1 Motivation und Einführung

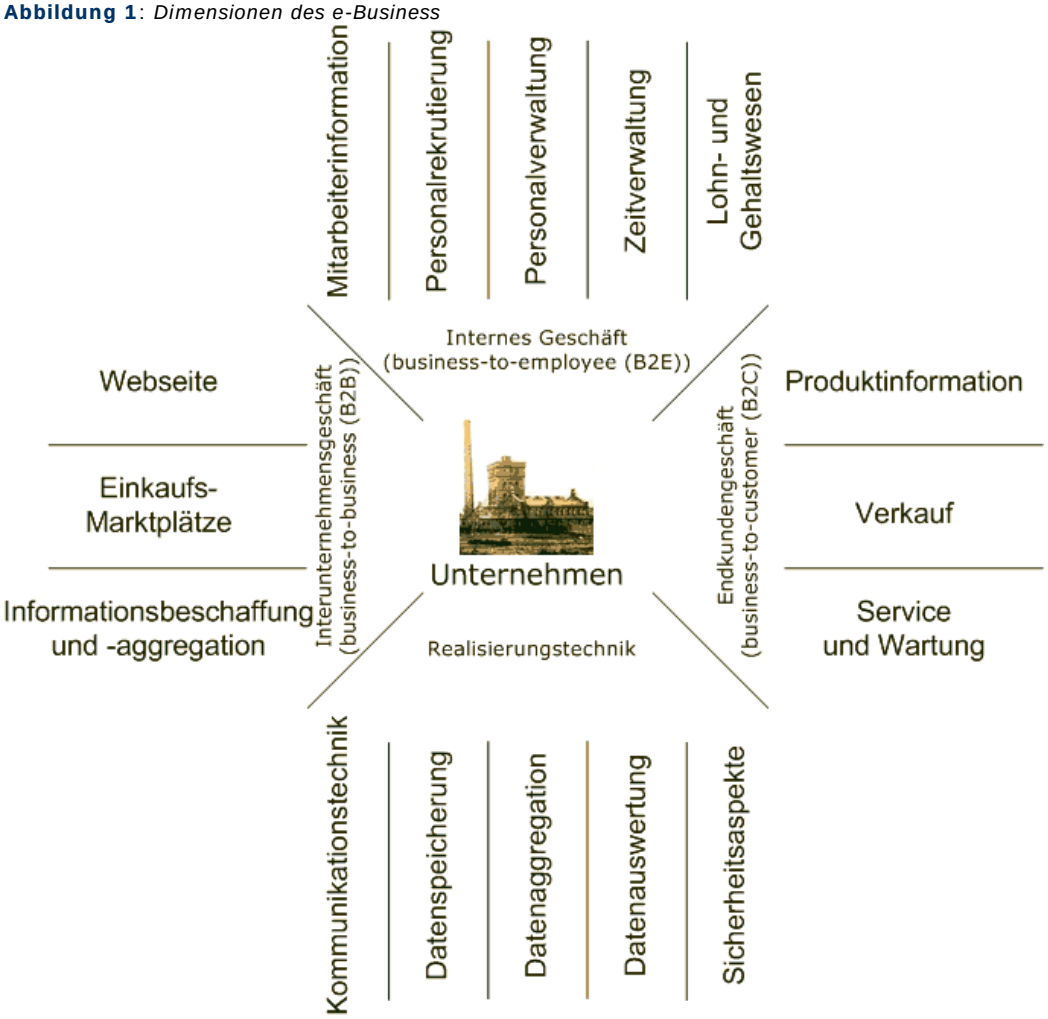
1.1 Was ist e-Business?

Der Begriff des *e-Business* als Abkürzung des englischsprachigen *electronic Business* hat sich inzwischen als Subsumption aller für ein Unternehmen wertschöpfenden Aktivitäten im Internet eingebürgert.
Die Sinngebung greift damit weiter als der historisch ältere Begriff *e-Commerce*, welcher ursprünglich ausschließlich Verkaufsaktivitäten bezeichnete. Inzwischen werden beide Terme jedoch nahezu synonym verwendet. Teilweise findet sich für den Teilbereich des internetgestützten Verkaufs von Waren und Dienstleistungen an Endkunden auch die Bezeichnung *e-tailing* (für *electronic retailing*) welcher jedoch nur einen Teilaspekt des e-Commercebegriffes abzudecken vermag.

Definition 1: e-Business
Electronic Business ist die Gesamtheit aller unternehmerischen Aktivitäten im Internet.

Gemäß dieser allgemeinen Definition werden sämtliche auf das Unternehmensziel gerichtete nach außen wirkende Aktivitäten als e-Business eingeordnet.
Gleichzeitig ergibt sich aus der Abstützung auf der Realisierungstechnik des Internets auch eine interne Sichtweise, sobald diese Technik innerhalb des Unternehmens zum Einsatz kommt.
Die Darstellung der Abbildung 1 unternimmt den Versuch der Einordnung der sich ergebenen Anwendungsdimensionen des e-Businessbegriffs.

Abbildung 1: *Dimensionen des e-Business*



(click on image to enlarge!)

Die naheliegendste Form des e-Business ist der Geschäftsverkehr mit dem (End-)Kunden, als dem typischen Konsumenten der durch ein Unternehmen zur Verfügung gestellten Güter und Dienstleistungen. Dieser Teilbereich wird mit dem Begriff *Business-to-Customer* (B2C) belegt.
In diese e-Businessvariante fallen alle Interaktionen zwischen Kunde und Unternehmen während des gesamten Lebenszyklus des angebotenen Produkts, angefangen von verkaufsfördernden Maßnahmen (Marketing) über den Verkaufs- bzw. Dienstleistungserbringungsakt selbst bis hin zur Abwicklung der Wartung, soweit nach Art des angebotenen Gutes elektronisch überhaupt möglich.

Entgegengesetzt zum durch ein Unternehmen produzierten ausgehenden Güter- und Dienstleistungsstrom verläuft die Beschaffung von nicht-menschlichen Produktionsfaktoren wie Roh-, Hilfs- und Betriebsstoffen sowie die Interunternehmenskommunikation. Dieser Teilbereich wird mit dem Begriff *Business-to-Business* (B2B) belegt.
In diese e-Businessvariante fallen die zwischen Unternehmungen ablaufenden elektronischen Kommunikationen. Die Spannbreite reicht hierbei von der kostenfrei nutzbaren statischen Präsentation des Güter- und Dienstleistungsangebots im Stile eines Katalogs über spezialisierte Marktplätze mit Angebots- und Nachfragefunktionalitäten bis hin zu Informationsdienstleistungen welche Zugriff auf die datenhaltenden Systeme des Geschäftspartners gewähren.

Die umfassende Betrachtung der zuvor ausgeklammerten Kommunikation mit potentiellen und bestehenden Mitarbeitern konstituiert die dritte Klasse der e-Businessanwendungen, welche auf die unternehmensinterne Kommunikation mit den Mitarbeitern fokussieren. Dieser Teilbereich wird mit dem Begriff *Business-to-Employee* (B2E) belegt.
Dieser Sparte werden alle elektronischen Informationsangebote an den Mitarbeiter, wie Auskunft über den aktuellen Gleitzeitstand, Adressstamm- sowie Gehaltsdaten, zugeordnet.

1.2 Relevante Techniken und ihre Einordnung

Orthogonal zu den drei Anwendungsdimensionen verdient die ebenfalls in Abbildung 1 dargestellte Realisierungstechnik Betrachtung.
Hierunter fallen gemäß Definition 1 alle sog. *Internettechniken*.

Dieser, in der Praxis nicht klar definiert und trennscharf gebrauchte Begriff umfaßt sowohl die Internetbasistechniken zur Datendarstellung und -übertragung als auch verschiedene Techniken zur Realisierung von Anwendungen, die über das Internet angesprochen und benutzt werden können.

Im wesentlichen zielen die eingesetzten Techniken auf die Lösung spezifischer Problemstellungen. Tabelle 1 stellt die im Rahmen der Vorlesung behandelten Techniken nebst den durch sie betrachteten Problemgebieten und einer Kurzcharakteristik zusammen.

Tabelle 1: Techniken: Einordnung und Kurzcharakterisierung

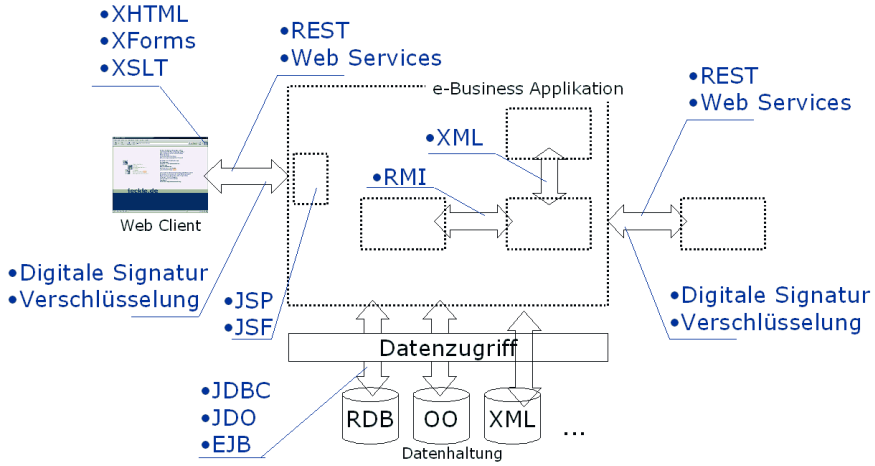
Problemdomäne	Technik	Charakteristik
Datendarstellung und -zugriff	XML	Generische Auszeichnungssprache zur Darstellung beliebiger Daten.
	XML-Namensräume	Syntaxmechanismus zur Unterscheidung von XML-Vokabularen.
	XML-Schema	Grammatiksprache zur Formulierung von XML-Vokabularen.
	XPath	Lokatorsprache zur Identikation von Knotenmengen in XML-Dokumenten.
Datenbankzugriff	JDBC	Durch SUN erarbeiteter Ansatz für den Zugriff auf tabellenartige Datenquellen. Zumeist für den Zugriff auf relationale Datenbanken benutzt.
	JDO	Mechanismus zur transparenten Persistierung von Java-Objekten in verschiedenen Datenspeichern.
	EJB	Durch SUN erarbeitete Komponententechnik. Hauptfocus in dieser Veranstaltung: Realisierung von Peristenz durch *Entity Beans*.
Funktionsintegration	JMS	Durch SUN für Java entwickelte Schnittstelle zur Verarbeitung asynchroner Nachrichten.
	RMI	Durch SUN für Java adaptierte Variante entfernter Funktionsaufrufe.
	REST	Interpretations- und Nutzungsvariante des HTTP-Protokolls zur Realisierung einfacher Web-Dienste.
Web Services	Ansatz zur Bereitstellung von Funktionalität über das Web mittels Nachrichtenaustausch und entfernter Funktionsaufrufe.	
Präsentationsaspekte	XHTML und XForms	Bekannteste Hypertextsprache und Ansatz zur Realisierung einfacher Web-basierter Eingabeoberflächen.
	JSP	Durch SUN erarbeiteter Ansatz zur dynamischen serverseitigen Erzeugung von Webseiten.

	JSF	Durch SUN erarbeiteter Ansatz zur vereinfachten Erstellung von GUI-basierten Web-Dialoganwendungen.
	XSLT	W3C-Standard zur Transformation von XML-Inhalten.
Sicherheitsaspekte	Schlüsselaustausch	Erzeugung und Verteilung geheimer und öffentlicher Daten, die den Zugriff auf gesicherte Daten gestatten.
	Leitungssicherheit	Bereitstellung transparenter Verbindungssicherung im Internet.
	Digitale Signatur	Sicherung von Datenkonsistenz, Glaubwürdigkeits des Ursprungs, Verbindlichkeit und Berechtigung.
	Verschlüsselung	Sicherung von Vertraulichkeit.

1.3 Architektur moderner e-Business Applikationen

Abbildung 2 ordnet die zuvor eingeführten Techniken in ein Architekturmodell für e-Business Applikationen ein.

Abbildung 2: *Architektur moderner e-Business Applikationen*



(click on image to enlarge!)

Das Architekturmodell zeigt die im Rahmen der Vorlesung behandelten Techniken als Bestandteil einer hypothetischen Architektur. Sie zeigt die bevorzugten Einsatzbereiche der Einzeltechniken und gibt damit bereits einen Ausblick auf die gegenwärtig in der Praxis etablierte Pragmatik.
Besonders deutlich wird dies anhand der dargestellten Positionierung des Remote-Method-Invocation-Mechanismus. Zwar kann dieser grundsätzlich auch zur systemübergreifenden Kommunikation herangezogen werden. Jedoch wird RMI aktuell vorwiegend für die Realisierung systeminterner Kommunikationsbeziehungen, beispielsweise innerhalb J2EE-basierter Applikationsserver, herangezogen. Dies liegt in zwei Grundfaktoren begründet. Zum einen ist nur ein Teil der verfügbaren e-Business-Systeme unter Nutzung der Programmiersprache Java realisiert, worauf die RMI-Anwendbarkeit faktisch beschränkt ist. Zum anderen ist der RMI inhärent zugrundeliegende Zugriff auf binäre Applikationsschnittstellen unter Sicherheitsrestriktionen als problematisch anzusehen.

2 Datendarstellung und -zugriff

2.1 Extensible Markup Language -- Strukturelle Grundkonzepte

Im Grunde besitzt die Geschichte der eXtensible Markup Language zwei Anfänge. Einerseits stellt XML die evolutionäre Fortentwicklung existierender generischer Auszeichnungssprachen dar; andererseits sind die

Hintergründe der Sprache XML so eng mit dem Aufkommen des World Wide Webs (WWW) verwoben, daß die Geschichte auch hier ihren Anfang nehmen könnte...

Der chronologischen Ordnung folgend sei zunächst die Entwicklung aus der Idee des *Hypertext* aufgerissen.
Die ersten Ideen zum Konzept des Hypertexts, als Plan zur Überwindung der Beschränkungen und Unzulänglichkeiten des klassischen textbasierten Publikationsmediums Papier, datieren zurück bis in die 1950er Jahre. Sie postulieren neben der nichtsequentiellen Organisation des Mediums auch zentrale Begriffe wie *Knoten*, *Link*, *Anker* und *Netz*. Ziel dieser Überlegungen war es, den auszudrückenden Inhalt von editorieller- und Präsentationsinformation wie Seitenzahlen, Fußnoten, Paginierung usw. zu trennen. Durch die nichtlineare Organisation soll es dem Leser freigestellt werden, auf welchen Pfaden er sich durch das Dokument bewegt.

Zur Realisierung dieser Bemühungen wird das Dokument mit weiteren Informationen angereichert, die jedoch für den Leser unsichtbar bleiben. Dieser Gedanke reicht zurück bis in die Anfänge des Buchdrucks. Dort sind formatierungsorientierte Auszeichnungssymbole, etwa für Fettdruck oder Unterstreichung, seit jeher bekannt. Vor dem Aufkommen der *what you see is what you get* Textverarbeitungssysteme waren diese bildlichen Symbole die einzige Möglichkeit zur Kommunikation präsentationsorientierter Information an den Schriftsetzer und Drucker.
Jedem Schüler ist bereits ein weiteres Beispiel einer editoriellen Auszeichnungssprache bekannt: Die graphischen Korrekturzeichen der Deutschlehrer. Auch sie liefern Informationen über den Inhalt, die nicht Bestandteil des Dokuments sind.

Voraussetzung für die angestrebte Flexibilisierung der Struktur eines Textes ist eine -- wie auch immer geartete -- technische Unterstützung. Seit den 60er Jahren wurden hierfür die aufkommenden elektronischen Rechenanlagen herangezogen. Eine der ersten Aktivitäten hierzu ist das von Ted Nelson initiierte (inzwischen legendäre) Xanadu-Projekt.

Zunächst erforderte die maschinelle Verarbeitung die Überarbeitung des Auszeichnungssymbolvorrates. Dies wurde notwendig, da eingesetzte Technik keine Unterstützung der alt-hergebrachten graphischen Auszeichungssymbole bot.
In einem ersten Entwicklungsschritt wurden daher die vormalig bildhaften Zeichen durch textuelle Pendants ersetzt und verallgemeinert. Beispielsweise: `Überschrift` zur inhaltlichen Kennzeichnung einer entsprechenden Textzeile.
Mit diesem Schritt erfolgte auch der Übergang zur formatierungsunabhängigen Auszeichnung, die bewußt auf die Beschreibung des späteren visuellen Aussehens der Information zugunsten einer neutralen deskriptiven Beschreibung der Semantik verzichtete.

In den 60er und 70er Jahren werden verschiedene Weiterentwicklungen der generischen Auszeichnungssprachen betrieben; u.a. bei der IBM durch das Team um Goldfarb, Mosher und Loire. Sie stellen 1969 unter dem Namen *Generalized Markup Language* einen Sprachvorschlag zusammen, der in der Folgezeit durch IBM kommerziell vermarktet wird.

Aus den GML-Aktivitäten bei IBM entwickelt sich die internationale Standardisierungsbewegung der *Standard GML* (SGML).
Durch sie wird eine Sprache festgelegt, welche die Definition eigener Sprachen erlaubt; daher auch der Begriff *Metasprache*. SGML bietet somit keinen feststehenden problemspezifischen Sprachumfang an, sondern eine Menge verschiedenster struktureller Konstrukte zur Formulierung von Dokumentgrammatiken.
In der Praxis wird der Einsatz einer mit Hilfe von SGML definierten Sprache oftmals plakativ zum *Einsatz von SGML* verkürzt, obwohl diese Begrifflichkeit lediglich den Erstellungsprozeß der Grammatik bezeichnet.

Mittels SGML definiert Tim Berners-Lee Mitte der 80er Jahre eine eigene Sprache zur vereinfachten Formulierung von Dokumenten, die er *HyperText Markup Language* (HTML) nennt. Hauptbeweggrund seiner Aktivitäten ist der Versuch den Dokumentenaustausch am Europäischen Kernforschungszentrum CERN rechnergestützt zu vereinfachen.
Die Eingangs erwähnten zentralen Hypertextkonzepte finden sich bereits in seinem ersten Sprachvorschlag wieder. Zur technischen Realisierung der Verknüpfung zwischen den Dokumenten mittels Ankern und Links definiert er den *Uniform Resource Locator* (URL), eine global eindeutige Adresse für beliebige Inhalte.

Seine Aktivitäten in Genf bilden die Keimzelle des Web.

In der Folgezeit, insbesondere im Zuge der Kommerzialisierung des Word Wide Web, entstehen verschiedene Revisionen der ursprünglichen HTML. Einige der Erweiterungen werden durch die beiden großen Web Browser Hersteller Microsoft und Netscape proprietär vorgenommen, um ihre Position am Markt zu stärken.
In der Konsequenz entstehen während des oft apostrophierten *browser war* teilweise inkompatible HTML-Dialekte. (Man denke nur an die Tags: `marquee` (nur Microsoft Internet Explorer) oder `layer` (nur Netscape Navigator))
Darüberhinaus entwickelt sich HTML zunehmend von einer Präsentations-orientierten Auszeichnungssprache zu einer semantischen. Dies bedeutet: während HTML in der ersten Grundform

zunächst überwiegend Elemente bot, durch die die Präsentation der Inhalte am Bildschirm festgelegt wurde (Beispiele: `b` für Fettdruck, `u` für Unterstreichungen oder `i` für Kursivschreibung), wurden später zunehmend semantische Elemente eingeführt. Durch sie wird die Bedeutung der ausgezeichneten Information ausgedrückt (Beispiele hierfür: `acronym` zur Kennzeichnung von Abkürzungen, `address` für Adressen oder `strong` zur besonderen Betonung einer Textpassage).

So wünschenswert die sukzessive Umgestaltung der HTML an die veränderten Bedürfnisse war, so aussichtslos waren die Bemühungen dennoch. Während bei den Präsentations-orientierten Elementen zunehmend Vollständigkeit hinsichtlich der Anwenderwünsche erzielt werden konnte, offenbaren sich die bisher erfolgten semantischen Erweiterungen als permanent inadäquat.
Letztlich war der Versuch, durch Standardisierung, semantische Erweiterungen in HTML einzubringen in doppelter Hinsicht zum Scheitern verurteilt:
1. birgt der Ansatz die Gefahr, die Elementmenge in unbekannte Größen zu erweitern
2. muß die Semantik jedes Tags definiert, abgestimmt und verabschiedet werden.

Aus diesen Gründen wurde seitens des W3C nach einer tragfähigeren Lösung gesucht. Unter Rückgriff auf die HTML-Wurzeln (als Anwendung der Metasprache SGML) wurde das Projekt *SGML for the Web* initiiert. Der letztendlich verabschiedete Vorschlag zur *eXtensible Markup Language* (XML) bildet konzeptionell eine Untermenge der Sprachmöglichkeiten von SGML. Konsequenterweise ist jedes XML-Dokument auch ein gültiges SGML-Dokument.

Die Abweichung zu SGML wird besonders aus den Entwicklungszielen für XML deutlich:

1. Einfache Nutzung im Internet.
   In Abkehr von den Hauptnutzung SGMLs als offline Dokumentationsformat wird die Untermengenbildung *XML* für die primäre Nutzung im Internet vorgenommen.
2. Unterstützung eines breiten Anwendungsspektrums.
   Auch hier soll die Untermengenbildung das Einsatzspektrum über die Hauptnutzung SGMLs als Format der technischen Dokumentation hinaus befördern.
3. SGML Kompatibilität.
   XML bildet eine echte Untermenge des ISO-Standards SGML, durch diesen Schritt kann jedes XML-Dokument auch als gültiges SGML-Dokument interpretiert und durch die entsprechenden SGML-Werkzeuge verarbeitet werden.
4. Einfache Applikationsentwicklung.
   Die Untermengenbildung wird im Hinblick auf eine gegenüber SGML deutlich vereinfachte Entwicklung von XML verarbeitenden Applikationen vorgenommen.
5. Minimierung optionaler Sprachmerkmale -- Idealerweise gleich Null.
   Auch dieses Ziel ist im Hinblick auf eine vereinfachte Applikationsentwicklung, aber auch eine einfachere Benutzbarkeit durch Menschen auf dem Wege der Komplexitätsreduktion zu interpretieren.
6. Lesbarkeit.
   Das entstehende Textformat soll für Menschen und Maschinen gleichermaßen les- und verstehbar sein.
7. Kompakte Spezifikation.
   Die erstehende XML-Spezifikation sollte deutlich weniger Umfang aufweisen als der SGML-Vorgängerstandard. Letztlich konnte die reine Seitenzahl von über 600 Seiten für die SGML-Spezifikation auf ungefähr 30 Seiten für XML reduziert werden.
8. Formaler und präziser Sprachentwurf.
   Um die schnelle Akzeptanz seitens der Anwender zu forcieren erachteten die Mitglieder der XML-Arbeitsgruppe die schnelle Verfügbarkeit von XML-Werkzeugen für essentiell. Aus diesem Grunde sollte der XML-Sprachentwurf möglichst leicht und eindeutig in XML-Werkzeuge zu implementieren sein.
9. Leichte Dokumenterstellung.
   Die Erstellung von korrekten XML-Dokumenten sollte idealerweise so einfach sein, daß hierfür keine speziellen Werkzeuge benötigt werden.
10. Nicht notwendigerweise knappes Markup.
    Kompaktheit und Effizienz hinsichtlich des Volumens eines XML-Dokuments war zu keinem Zeitpunkt eines der Hauptentwicklungsziele. Auf der Basis des XML-Information Sets ist es jedoch möglich beliebig kompakte Binärformate identischer Mächtigkeit zur die in der XML-Spezifikation vorgestellten Textnotation zu definieren.

XML stellt jedoch keine echte semantische Auszeichnungssprache dar, da durch die Metasprache lediglich eine Möglichkeit zur Formulierung eigener Syntax gegeben ist. Die Bedeutung der Elemente bleibt jedoch unberücksichtigt, und kann mittels XML nicht ausgedrückt werden.

Tabelle 2: Einige chronologische Eckdaten

Jahr	Ereignis
1945	Vannevar Bush diskutiert in seinem Artikel As We May Think ein persönliches Informationssystem mit Kommunikationsmöglichkeiten und Zugriff auf Bücher, Tonaufnahmen, etc. unter dem Namen *Memex*.
1967	William Tunnicliffe (Chairman des Graphic Communications Association (GCA) Composition Committee) schlägt aus seinen Erfahrungen bei der wiederholten Erstellung von Telephonkatalogen (*yellow pages*) vor, häufig auftretende strukturelle Elemente zu standardisieren.
September 1967	William Tunnicliffe (Vorsitzender der Graphic Communication Association) spricht sich auf einer Konferenz des *Printing Office* der Regierung von Kanada für die Separierung von Inhalt und Format aus.
Ende der 1960er Jahre	Stanley Rice, ein New Yorker Schriftsetzer, schlägt *editorial structure tags* vor. Der CGA-Direktor Norman Scharpf initiiert das Projekt *GenCode*.
1969	Charles Goldfarb, Edward Mosher und Raymond Lorie entwickeln bei der IBM die *Generalized Markup Language* (GML). Anwendungshintergrund war ein Projekt zur Integration von Informationssystemen für Anwaltskanzleien.
1970	Goldfarb formuliert zwei Grundprinzipien generalisierter Auszeichungssprachen: 1) Auszeichnungssprachen beschreiben die Dokumentstruktur, nicht die physischen Charakteristika wie Präsentation 2) Die Struktur der Auszeichnungssprache soll so gewählt sein, daß sie sowohl von Menschen als auch Maschinen interpretiert werden kann
1978	ANSI ruft *Computer Languages for the Processing of Text*-Komitee ins Leben. Ziel ist die Weiterentwicklung der GML zu einem nationalen US-Standard.
1980	• ANSI veröffentlicht ersten Entwurf einer standardisierten GML (SGML). • Tim Berners-Lee tritt seine Arbeit am Europäischen Kernforschungszentrum CERN an. Dort entwickelt er in der Folgezeit die (niemals veröffentlichte) Hypertextanwendung *Enquire*.
1983	Der International Revenue Service (IRS) und das US Verteidigungsministerium (DoD) übernehmen den sechsten Entwurf zur SGML (auch bekannt als GCA 101-1983).
1984	Die SGML-Arbeitsgruppe nimmt unter Schirmherrschaft der International Standardization Organization (ISO) als ISO/IEC JTCI/SC18/WG8 ihre Arbeit auf. Goldfarb dient als *technical leader* der ISO-Gruppe, sowie dem umorganisierten ANSI-Komitee X3V1.8.
1985	Norm-Entwurf zu SGML veröffentlicht.
15. Oktober 1986	ISO verabschiedet SGML als ISO 8879:1986.
März 1989	Berners-Lee schlägt mit dem Dokument *Information Management: A Proposal* ein SGML-basiertes Hypertext-System zum Informationsaustausch vor.
1990	Am Weihnachtstag nimmt das World Wide Web seinen Betrieb mit zwei Maschinen am CERN auf. Die notwendigen Implementierungen von HTML, HTTP und URL erfolgten durch Berners-Lee. Die erste WWW-Verbindung wird zwischen Berners-Lees Workstation und Robert Cailliaus' NeXT-Rechner aufgebaut. Ein Screenshot des ersten Web-Browsers NeXTStep-Implementierung des Browsers
1991	Beginn der turnusmäßigen Überarbeitungsphase von ISO 8879.
3. November 1992	Erster Entwurf zu HTML
Juni 1993	Einreichung des ersten HTML Entwurfs bei IETF.
Oktober 1994	Gründung World Wide Web Consortium
14. November 1996	Erster Entwurf zu XML vorgestellt
14. Januar 1997	Verabschiedung der HTML v3.2
1998	W3C gibt die erste Version von XML als Recommendation frei.
2000	• W3C gibt XHTML v1.0 -- die Reformulierung von HTML v4.01 zu einer XML-Anwendung -- frei. • W3C verabschiedet XML 2nd edition; sie integriert u.a. die XML Namespaces und behebt einige editorielle Fehler.
2. Mai 2001	Das W3C verabschiedet den XML Schema-Standard. Er geht an vielen Stellen deutlich über die ererbten SGML-Möglichkeiten hinaus, und markiert den Übergang von Präsentations-orientierten Strukturen hin zu Datenstrukturen.

Zum Abschluß dieser Einführung seinen die zehn Punkte zusammengestellt und kommentiert, die durch das World Wide Web Consortium als plakative Kurzcharakterisierung von XML veröffentlicht wurden:

1. XML steht für strukturierte Daten.
   Diese Aussage betont die Rolle von XML als Sprache um Sprachen zu erzeugen. Nicht XML wird innerhalb verschiedenster Applikationen direkt verarbeitet, sondern XML basierte Formate. So steht nicht die XML selbst für all diese Anwendungsdomänen, sondern die jeweiligen problemspezifischen XML-basierten Sprachen. XML selbst dient lediglich der Strukturierung der verschiedensten darzustellenden Daten.
   Gleichzeitig rückt durch Aussage die Rolle der XML als Datenformat in den Vordergrund und läßt so die Weiterentwicklung gegenüber den präsentationsorientierten Vorläufern deutlich werden.
   Die Vorlesungskapitel Strukturelle Grundkonzepte und XML Schemasprachen vermitteln einen Eindruck dieses Wandels und dokumentieren die Grundlagen des gegenwärtigen datenorientierten Einsatzes der XML.
2. XML sieht ein wenig wie HTML aus.
   Diese Aussage soll offenkundig einerseits den bisherigen HTML-verwendenden Web-Autoren den Einstieg in die XML schmackhaft werden lassen. Dennoch führt sie ein wenig von der Grundidee XMLs als generischer Auszeichnungssprache für beliebigste Anwendungen weg, indem sie den Blick auf HTML focussiert.
   Die -- im Grunde der Verwandschaft zu SGML geschuldete -- offensichtliche syntaktische Ähnlichkeit zu HTML wird bereits bei der Betrachtung der strukturellen Grundkonzepte deutlich.
3. XML ist Text, aber nicht zum Lesen.
   XML-Dokumente können sicherlich im wörtlichen Sinne „gelesen“ werden ... Die Aussage zielt jedoch auf den intendierten Einsatzzweck von XML: der Darstellung von Daten für den Austausch zwischen Maschinen. Unbenommen dessen kann XML selbstverständlich auch von Menschen gelesen und verstanden werden, wenngleich dies bei umfangreicheren XML-Dokumenten durchaus mühsam werden kann.
   Aufschluß über die textuelle Natur XMLs, insbesondere im Hinblick auf die Verwendung unterschiedlicher Alphabete, liefert das Kapitel strukturelle Grundkonzepte.
4. XML ist vom Design her ausführlich.
   Hiermit wird versucht dem häufig geäußerten Kritikpunkt der Platzzunahme XML-codierter Inhalte gegenüber klassischen Darstellungsweisen etwas pauschal entkräftend entgegenzutreten. Sicherlich geht das W3C in dieser Aussage nicht fehl, wenn die Entwicklung der Netzwerkbandbreiten, der CPU-Leistung und der Speicherkapazitäten berücksichtigt. Andererseits ist die Aufblähung der XML-formatierten Inhalte im Vergleich zu optimierten Binärformaten nicht von der Hand zu weisen, wird jedoch durch die mit der Verwendung von XML einhergehenden Vorteile mehr als ausgeglichen.
   Einen ersten Eindruck der Natur XML-codierter Inhalte liefert das Kapitel strukturelle Grundkonzepte. Dort finden sich auch Ansätze die bekannte XML-Syntax kompaktifiziert darzustellen ohne die Vorteile der generischen Auszeichnungssprache aufgeben zu müssen.
5. XML ist eine Familie von Techniken.
   Eine Aussage, die durch alle drei Kapitel der Vorlesung unterstrichen wird, die deutlich zeigen, daß XML nicht als isolierte Idee oder Technik anzusehen ist -- sondern erst im Zusammenspiel mit anderen XML-Standards und eingebettet in Applikationen und Infrastrukturen -- seine volle Wirkungsmächtigkeit entfalten kann.
6. XML ist neu, aber nicht so neu.
   Diese Bezugnahme soll nochmals unterstreichen, daß XML keineswegs den Anspruch erhebt eine vollkommen neue technische Errungenschaft zu sein, sondern vielfach bekanntes und erprobtes aus der Informatik wiederverwendet und im neuen Verwendungskontext weiterentwickelt.
   Diese Aussage wird durch die in den einzelnen Kapiteln dargebotenen Rückbezüge auf bereits bekannte Techniken und Lösungsformen untermauert.
7. XML überführt HTML in XHTML.
   Diese Aussage greift nochmals die Beziehung zwischen XML und HTML auf. Diesmal soll die Rolle von XML im Bezug auf die Weiterentwicklung von HTML zum XML-basierten Vokabular *XHTML* unterstrichen werden. So löst XML die Abhängigkeit zwischen SGML und HTML auf und reformuliert HTML auf der Basis von XML.
   Das Kapitel XHTML führt kurz in die Entwicklung der neuen HTML-Varianten auf Basis der XML ein und skizziert die vorgenommen Änderungen und zukünftige Erweiterungen dieser Hypertextsprache.
8. XML ist modular.
   Hierdurch wird unterstrichen, daß XML kein in sich geschlossenes monolithisches Gebilde darstellt, sondern einzelne Vertreter aus der Familie der XML-Sprachen wahlfrei zur Lösung konkreter Probleme herangezogen werden können. Ebenso wird die Sprachfamilie beständig an verschiedensten Stellen unabhängig voneinander weiterentwickelt, ohne einer zentralen Koordination zu bedürfen.
9. XML ist die Basis für RDF und das Semantic Web.
   Grundidee des Semantic Web ist die Weiterentwicklung des sichtbaren XHTML-basierten Webs unter Nutzung seiner datenorientierten Ergänzung XML zu einem Netz von Sinnzusammenhängen.
10. XML ist lizenzfrei, plattform- und herstellerunabhängig, und gut unterstützt.
    XML ist eine durch das World Wide Web Consortium herausgegebene Spezifikation, die kostenfrei über das Web bezogen werden kann und durch Interessierte ohne weitere Lizenzkosten in eigenen kommerziellen Produkten verwendet werden. Durch den Standardisierungsprozeß innerhalb des World Wide Web Consortiums wird sichergestellt, daß keine Ausführungsplattform bevorzugt wird und gleichzeitig keine Nachteile für Andere entstehen. Dies wird durch die herstellerunabhängige Organisation des Gremiums versucht zu garantieren, in dem zwar Hersteller Mitglied werden können,

die technischen Entscheidungen jedoch Arbeitsgruppen obliegen, die nicht durch eine Firma dominiert werden können.

Web-Referenzen 1: Vertiefende Informationen

- Artikel in der Online-Ausgabe des *Economist* über Ted Nelson -- *The Babbage of the web*
- COT1800 Public Networks, Lecture 8, Standard Generalised Markup Language
- Brief History of Document Markup
- XML, Element Types, DTDs, and All That
- Clark, J.: *Comparison of SGML and XML*

Web-Referenzen 2: Weiterführende Links

- Browser Timelines
- Browser Emulator

Definition 2: XML-Sprache

Eine Anwendung der Extensible Markup Language. Ein Vokabular, das aus Symbolen und der ihnen zugewiesenen Bedeutung (Semantik) gebildet wird, ergänzt um Regeln (grammatikalische Struktur und Gültigkeitsregeln für den Inhalt (z.B. Datentypen)) zur Kombination der Vokabularelemente.
Anwendungen einer so neu geschaffenen XML-Sprache *L* werden als XML-Dokumente, auch: *L*-Dokumente, bezeichnet.

Strukturelle Grundkonzepte

Die grundlegende XML-Syntax ist in der namensgebenden W3C-Recommendation der Extensible Markup Language definiert. Die Semantik der Metasprache wird hingegen durch den W3C-Standard des XML Information Set festgelegt.
Diese Spezifikationen beinhalten die grundlegenden Definitionen hinsichtlich Terminologie und Beziehung der verschiedenen möglichen Elemente eines XML-Dokuments. Im vorliegenden Teilkapitel werden beide Sprachaspekte grundlegend eingeführt und ein erstes Verständnis der XML vermittelt. Dabei wird in Form von Ausblicken auf nachfolgende Abschnitte der Bogen zu Grammatikdefinitionssprachen und weiterführenden Konzepten wie Namensräumen gespannt.
Zum leichteren Verständnis sind die aus der offiziellen Spezifikationen entnommenen formalen Grammatikdefinitionen der EBNF-Notation durch vereinfachte graphische Strukturdarstellungen ergänzt.

Definition 3: XML Dokument

Ein XML-Dokument ist ein Datenstrom (der nicht zwingend als Datei vorliegen muß), welcher den Strukturierungsprinzipien der eXtensible Markup Language genügt.

Definition 4: XML Information Set

Die Spezifikation des XML Information Sets definiert die Semantik der Metasprache XML, d.h. ihre zentralen Begriffe.
Gleichzeitig setzt es diese Begriffe in Beziehung und definiert so syntaxunabhängig die Struktur eines XML-Dokumentes.

Ausgehend von der Allgemeinheit der Aussage aus Definition 1 folgt, daß der Infoset neben seinem theoretischen Wert als Semantikdefinition zur XML auch zur Formulierung der Datenstrukturen, welche innerhalb eines XML-Prozessors vorliegen müssen, um beliebige XML-Dokumente verarbeiten zu können, herangezogen werden kann.
Daher läßt sich ein XML-Prozessor definieren als:

Definition 5: XML-Prozessor

Ein XML-Prozessor ist eine maschinelle Komponente (typischerweise: Software), die zum Lesen, Speichern und Verarbeiten eines XML-Dokuments eingesetzt wird.
Er erlaubt Zugriff auf den Inhalt und die Struktur des XML-Dokuments.

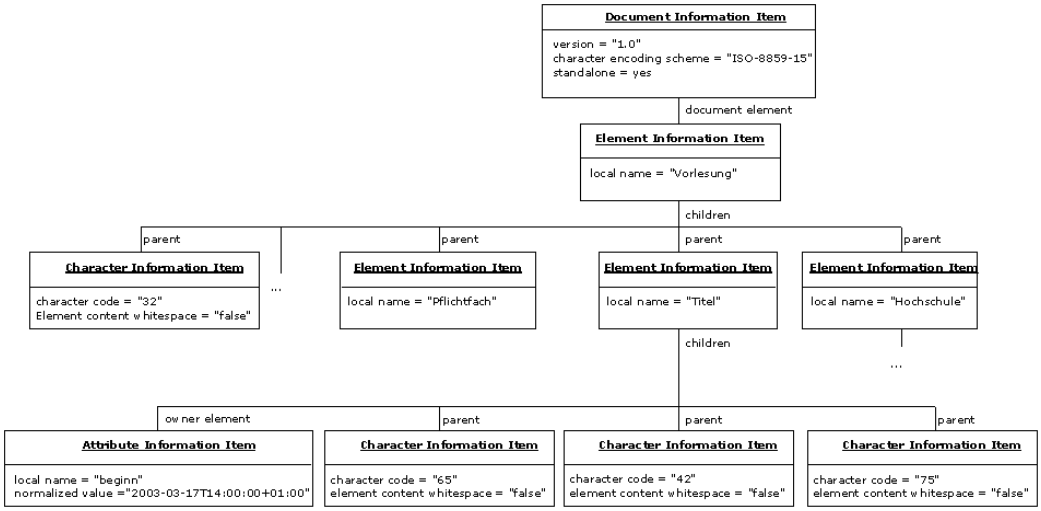
Die XML-Spezifikation faßt den XML-Prozessorbegriff etwas enger und beschränkt ihn lediglich auf Software-Module, die XML-Dokumente lesend verarbeiten. Konzeptionell spricht jedoch nichts gegen eine Umsetzung in Hardware, beispielsweise im Kontext eingebetter Systeme etc. (In XML-Spezifikation nachschlagen)
Ferner nimmt die XML-Spezifikation an, ein Prozessor operiere nicht eigenständig, sondern im integrierten Zusammenspiel mit einer Applikation.

Beispiel 1: Ein erstes XML-Dokument

```
(1)<?xml version="1.0" encoding="ISO-8859-15" standalone="yes"?>
(2)<Vorlesung>
(3)   <Pflichtfach/>
(4)   SS2003
(5)   <Titel beginn="2003-03-17T14:00:00+01:00">eBusiness-Engineering</Titel>
(6)   <Hochschule>Fachhochschule Furtwangen</Hochschule>
(7)   <Praktikum>Kein Übungsbetrieb</Praktikum>
(8)</Vorlesung>
```

Download des Beispiels

Das Beispiel zeigt ein erstes einfaches XML-Dokument, welches bereits die häufigst verwendeten Sprachelemente der XML versammelt.
Jedem XML-Dokument entspricht genau ein Information Set, der alle Informationselemente des Dokuments in Form einer Baumstruktur beinhaltet. Die nachfolgende Abbildung zeigt den Information Set des Beispiels in der Notation eines UML-Klassendiagramms. Dabei sind die einzelnen Knoten des Information Sets als Objekte (Klassensymbole mit unterstrichenem Klassennamen) und die Eigenschaften der Knoten als Attributwerte dargestellt.



Document Information Item

Jedes Information Set besteht genau aus einem *Document Information Item*. Dieses stellt den äußeren Rahmen des XML-Dokuments dar. Es beinhaltet dokumentbezogene Informationen, wie die verwendete XML-Version und das gewählte Codierungsschema innerhalb des Unicode-Systems.
Das Document Information Item enthält daher u.a. die Informationen des XML-Dokumentprologs in der erste Zeile jedes Dokuments. Das durch die öffnende Winkelklammer und ein Fragezeichen eingeleitete Konstrukt ist in der ersten Zeile des Beispiels 1 dargestellt. Innerhalb des Prologs findet sich die Zeichenkette `xml`, sowie die Bezeichner `version` und `encoding`. Beiden ist ein durch doppelte Hochkommata umschlossener Wert nachgestellt, `1.0` für `version`, bzw. `ISO-8859-15` für `encoding`. Beendet wird der Prolog wiederum durch ein Fragezeichen und die schließende Winkelklammer. Wird auf die Angabe des optionalen Prologs im Dokument verzichtet, so sind die daraus ableitbaren Angaben im Document Information Item nicht gesetzt.

Als weitere Eigenschaften verfügt jedes Document Information Item über eine geordnete Liste von Kindknoten. Darin ist genau ein Element Information Item enthalten, welches den Startknoten des XML-Dokuments verkörpert. Wegen seiner hervorgehobenen Bedeutung als Wurzel des Dokumentbaumes wird dieser Knoten auch als `Document Element` bezeichnet.
Zusätzlich kann die Liste Elemente vom Typ Processing Instruction Information Item enthalten. Sie dienen der Darstellung von Verarbeitungsanweisungen, die durch den XML-Prozessor interpretiert werden.
Im Kopfbereich vor `Document Element` plazierte XML-Kommentare werden durch Comment Information Items innerhalb der `children`-Liste dargestellt.

Zusammengefaßt enthält das Document Information Item folgende Informationen:

- **Kindknoten**: In der Reihenfolge des Auftretens im Dokument geordnete Liste. Sie enthält mindestens ein `Element Information Item`, welches in der Rolle des `Document Items` fungiert. Ferner je ein Element des Typs `Processing Instruction Information Item` für jede Processing Instruction die außerhalb des Wurzelements definiert ist und jeweils ein `Comment Information Item` zu jedem definierten Kommentar.
- **Document Element**: Ein Element des Typs `Element Information Item`, das auf den Wurzelknoten des Dokuments verweist.
- **Basis URI**: Lokation, falls bekannt, des XML-Dokuments in Form eines *Uniform Resource Identifiers* (URI) gemäß IETF RFC 2396.
- **Character Encoding Scheme**: Der Name der gewählten Codetabelle aus dem Unicode-Standard.
- **Standalone**: Legt -- als Boole'scher Wert (Zugelassene Belegungen: *yes*, *no*) -- fest, ob die im Dokument

gespeicherten Daten durch eine sie verarbeitende Applikation interpretiert und somit ergänzt oder verändert werden.
Häufigste Form dieses Interpretationsvorganges ist die Präsenz einer expliziten Grammatik in Form einer *Document Type Definition*, welche Gültigkeitsregeln für eine Familie von Dokumenten formuliert. Konsequenterweise bedeutet daher die Belegung mit *no*, daß die gespeicherten Daten entweder durch die Applikation interpretiert werden, dies ist beispielsweise bei der Auflösung von Entitäten der Fall, oder durch eine *DOCTYPE*-Deklaration ein Verweis auf die externe Dokumentgrammatik erfolgt. Die Angabe von *standalone* ist optional. Fehlt sie und ist gleichzeitig eine DOCTYPE-Deklaration im Dokument gegeben, so wird *standalone="no"* angenommen.
Im Beispiel ist die standalone-Deklaration auf *yes* gesetzt, d.h. es existiert explizit keine Dokumentgrammatik. (In XML-Spezifikation nachschlagen)

- **Version**: Die eingesetzte XML-Version. Dieser Wert wird aus dem Dokumentprolog übernommen.

Wie auch im Beispieldokument, bildet die erste Zeile den sog. *Prolog* eines jeden XML-Dokuments (In XML-Spezifikation nachschlagen) . Die Angabe der Version ist zwingend und derzeit auf die Konstante `1.0` fixiert. Die aktuelle XML-Spezifikation sieht als gültige Belegung der Versionsangabe ausschließlich die Zeichenkette `1.0` vor. Zukünftigen Weiterentwicklungen ist es jedoch freigestellt auch andere Revisionskennungen zu vergeben.
`encoding` leitet das zweite Namen-Wert-Paar ein. Die Deklaration ist innerhalb des Prologs optional, und kann daher auch unterbleiben. Die Zeichenkette der Encodingdeklaration benennt das Codierungsschema, welches für das so gekennzeichnete Dokument verwendet wurde. Es definiert den Satz der innerhalb des Dokumentes zugelassenen Zeichen fest.
Gemäß Produktion 22 der XML-Syntaxdefinition ist der gesamte Prolog optional.

Die Encoding-Deklaration hat folgendes Aussehen (In XML-Spezifikation nachschlagen) :

```
[80] EncodingDecl ::= S 'encoding' Eq ('"' EncName '"' | "'" EncName "'" )
[81] EncName      ::= [A-Za-z] ([A-Za-z0-9._] | '-')*
[3]  S            ::= (#x20 | #x9 | #xD | #xA)+
[25] Eq           ::= S? '=' S?
```

Die Festlegung der Produktion 80, sowie die der Produktion 23, stellt heraus, daß sich die Encodingdeklaration nicht auf die Prologzeile selbst auswirkt. Hier sind die beiden Zeichenketten `xml` und `encoding` in der Codierung UTF-8 oder UTF-16 Vorschrift.

Als Belegungen des `Encoding Namens` (`EncName`) sind beliebige Zeichensätze zugelassen. Der XML-Standard empfiehlt jedoch lediglich auf die durch die *Internet Assigned Numbers Authority* verwalteten zurückzugreifen (Dokument: Official Names for Character Sets) (In XML-Spezifikation nachschlagen) .
Die häufigsten praktisch eingesetzten Deklarationen sind die der ISO-8859 (*extended ASCII*)-Familie, sowie die der Unicode- und ISO-10646-Standards.
Die verschiedenen Abschnitte der ISO-8859 Familie werden als `ISO-8851-`*n* ausgedrückt, wobei *n* die Nummer des Abschnittes des zugehörigen ISO-Dokuments referenziert. Ferner können die durch JIS X-0208-1997 normierten asiatischen Zeichensätze als `ISO-2022-JP`, `Shift_JIS` und `EUC-JP` dargestellt werden.

```
<?xml version="1.0" encoding="Shift_JIS" standalone="yes"?>
<kougi>
  <title begin="二千一年三月二十一日三時三十分＋九時">選挙義務講義ＸＭＬ</title>
  <organization>アウグスブルグ大学コンピュータ・サイエンス過分、工学と家政と形成の大学</organization>
</kougi>
```

Unicode stellt einen Industriestandard (entwickelt u.a. durch Apple, HP, IBM, Microsoft und SUN) zur Darstellung verschiedenster Alphabete und graphischer Zeichen dar. Sein zunächst durch 16-Bit codierter Zeichenvorrat bot Raum für 65536 unterschiedliche Symbole.
Die seit 1991 laufenden Unicodebemühungen münden in die ISO-Norm zur Erweiterung des klassischen ASCII-Codes (ISO 646) als ISO-10646 *Universal Multiple-Octet Coded Character Set (UCS)*. Seit 1996 sind beide Standards synchronisiert und werden abgestimmt vorangetrieben.
UCS definiert zwei aufeinander aufbauende Codierungen: UCS-2 (16 Bit Umfang) und UCS-4 (32 Bit). Der bisherige Unicode-Standard ist voll kompatibel zu UCS-2 und durch diesen darstellbar.

Tabelle 3: Verschiedene Codierungen des Zeichens "A"

Codierung	Bitbreite	Binärdarstellung	Größe der Beispieldatei in Byte (ohne Berücksichtigung des XML-Prologs)	Bemerkung zum Meßwert
UTF-7	>= 7	100 0001	263	(encoding="UTF-7")
Extended ASCII, Latin-1 (ISO-8859-1)	8	0100 0001	258	(encoding="ISO-8859-1")

UTF-8	>= 8	0100 0001	259	(encoding="UTF-8") keine Byte Order Mark
UCS-2, Unicode	16	0000 0000 0100 0001	516	(encoding="UCS-2") keine Byte Order Mark
UTF-16 (big endian)	>= 16	0000 0000 0100 0001	516	(encoding="UTF-16") keine Byte Order Mark
UCS-4	32	0000 0000 0000 0000 0000 0000 0100 0001	1032	(encoding="UTF-8") keine Byte Order Mark
UTF-32	>= 32	0000 0000 0000 0000 0000 0000 0100 0001	1032	(encoding="UTF-32") keine Byte Order Mark

Die Zeilenumbrüche wurden in allen Fällen durch die Kombination von Wagenrücklauf und Zeilenvorschub ausgedrückt.

Die Tabelle stellt einige Codierungen zur Darstellung des Zeichens *A* zusammen.
Auffallend ist der große Platzbedarf der UCS-2 und -4 Codierungen. Insbesondere bei den „klassischen" ASCII-Symbolen werden hier (u.U. sehr viele) führende Nullbits erzeugt, die in der Konsequenz zu einer deutlichen Vergrößerung der Beispieldatei führen.
Daher wurde mit dem *UCS Transformation Format* (UTF) eine kompaktere Darstellung zum jeweiligen UCS-Set eingeführt. UTF-8 verwendet standardmäßig die ersten acht Bit zur Darstellung der bekannten ASCII-Zeichen

Anmerkung: Inzwischen existiert auch eine „UTF-32" genannte 32-Bit Ausprägung, diese ist jedoch identisch zu UCS-4, mit Ausnahme daß durch UTF-32 „nur" 2^{21}-Zeichen dargestellt werden können.
Die Dateigröße ist daher für das betrachtete Beispiel in dieser Darstellungsweise unverändert zu der des UCS-4-Encodings.

Der Größenunterschied zwischen der UTF-7 codierten Datei und der Latin-1 encodierten erklärt sich aus der Darstellung des Umlautes sowie des +-Zeichens, die beide nicht nicht im klassischen 7-Bit ASCII-Code enthalten ist. So wird *Ü* im Wort *Übungsbetrieb* des Beispieldokumentes durch die die Bytefolge `2B 41 4E 77 2D` dargestellt, während alle übrigen Zeichen durch ein einzelnes Byte ausgedrückt werden können.
UTF-8 ist in der Lage sämtliche Standard-ASCII-Zeichen durch jeweils genau ein Byte auszudrücken, wiederum für den Umlaut muß auf die 16-Bit-Darstellung des UCS-2 zurückgegriffen werden. Daher erhöht sich hier die Dateigröße um ein Byte.
Erwartungsgemäß beträgt der Umfang des UCS-2 codierten Dokuments exakt das Doppelte des 8-Bit Äquivalents der Latin-1-Darstellung.
Dasselbe gilt für die UTF-16-Variante, die für das vorliegende Beispiel unterschiedslos zu UCS-4 verläuft, da keinerlei Zeichen aus UCS-4 im Dokument auftreten.

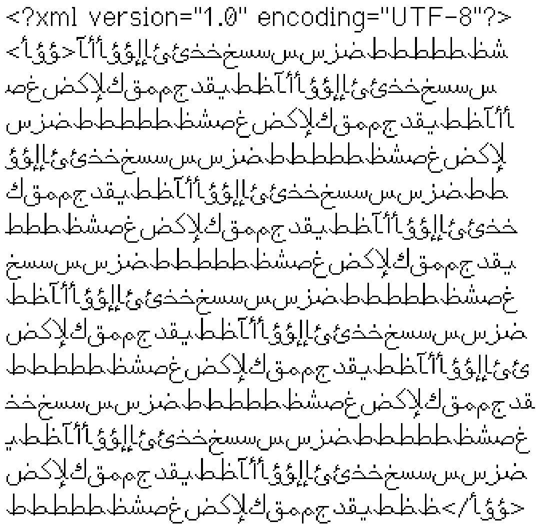
Die nachfolgende Tabelle stellt beispielhaft die Anwendung der UTF-8-Codierung zusammen:

Tabelle 4: UTF-8 Codierung

Unicode-Bereich	Bitbelegung
U-00000000 - U-0000007F:	0xxxxxxx
U-00000080 - U-000007FF:	110xxxxx 10xxxxxx
U-00000800 - U-0000FFFF:	1110xxxx 10xxxxxx 10xxxxxx
U-00010000 - U-001FFFFF:	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U-00200000 - U-03FFFFFF:	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U-04000000 - U-7FFFFFFF:	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Diese Mimik zeigt den Nachteil des UTF-*n*-Encodings deutlich: Die Darstellung nicht *n*-Bit darstellbarer Zeichen benötigt u.U. mehr Bitstellen als im Standard UCS-Code.
So wird beispielsweise das Zeichen mit der größtmöglichen Position (`7FFFFFFF`) in UTF durch sechs Byte encodiert, während UCS dieselbe Information mit den verfügbaren 32-Bit ausdrücken kann. Andererseits „verschwendet" die UCS-Darstellung für die niederwertigen Zeichen Bitstellen durch die führenden Nullen.

In der Praxis gilt es daher für das zu wählende Encoding einen möglichst guten Kompromiß zu finden: Im allgemeinen stellt das UTF-8-Encoding einen solchen dar, soweit überwiegend ASCII-Zeichen, und nur vereinzelt Sonderzeichen (hierzu zählen auch die deutschen Umlaute) eingesetzt werden.
Bei überwiegender Verwendung nicht in acht-Bit ASCII darstellbarer Zeichen (z.B. arabischer, chinesischer, etc.) erhöht die dann aufwendigere UTF-8-Codierung die Datenmenge.
So umfaßt die UTF-16-Darstellung des unten abgebildeten Beispieldokuments, welche in diesem Anwendungsfall identisch zu UCS-2 ist, 966 Bytes, während UTF-8 1299 Byte benötigt.

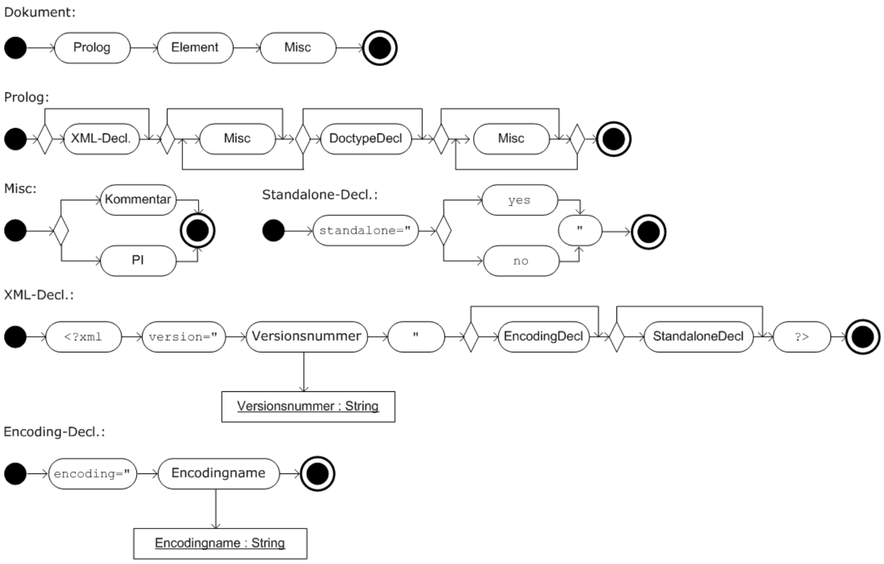


Achtung: Bereits durch die Unterstützung der beiden ISO-Zeichendarstellungen UTF-8 und UTF-16 ist die Konformität zum XML-Standard erfüllt! XML-Prozessorimplementierungen wird nicht abverlangt darüberhinausgehend weitere Darstellungen umzusetzen. (In XML-Spezifikation nachschlagen)

Wie bereits eingangs angemerkt, erklärt die XML-Spezifikation die Encodingdeklaration sowie den gesamten Prolog-Ausdruck als optionales Element (In XML-Spezifikation nachschlagen) .
Als Konsequenz geht dabei (auch) die Angabe des gewählten Encodings verloren.
Daher fordert der Anhang F der XML-Spezifikation *Autodetection of Character Encodings* bei einem von UTF-8 oder -16 abweichendem Codierungsschema die zwingende Angabe der XML-Deklaration (`<?xml ...`) (In XML-Spezifikation nachschlagen) .
Hintergrund dieser Maßnahme ist der Versuch anhand der damit bekannten fünf Zeichen das zugrundeliegende Encoding zu ermitteln.
Diese fünf Zeichen können als stabil angenommen werden, da Produktion 23 und 80 diese explizit von einem von UTF-8 oder -16 abweichenden Encoding ausnehmen.

Für Dokumente im deutschen Sprachraum, d.h. XML-Ströme die häuptsächlich aus den um die deutschen Umlaute ergänzten Standard-ASCII-Zeichen bestehen, hat es sich in der Vergangenheit eingebürgert den Zeichensatz latin-1 (`ISO-8859-1`) zu verwenden, um die Mehrbytedarstellung der Umlaute und weiterer Sonderzeichen in der UTF-Codierung zu umgehen.
Jedoch enthält der latin-1-Zeichensatz nicht das unter Unicode-Zeichennummer 20AC abgelegte Eurosymbol (_) welches zur Abkürzung des Währungsbegriffes der europäischen Gemeinschaftswährung verwendet wird.
Dieses Symbol wurde in die unter Nummer 15 veröffentlichte aktualisierte Fassung der Zeichensatzfamilie 8859 aufgenommen. Daher sollte bei der Erstellung von XML-Dokumenten generell darauf geachtet werden entweder `ISO-8859-15` als Codierung zu wählen oder auf die ohnehin ungleich flexiblere UTF-Codierung zurückzugreifen.

Die Darstellung der Abbildung 4 faßt die syntaktischen Elemente abgekürzt zusammen:



Web-Referenzen 3: Weiterführende Links
- Payer, M.: *UNICODE, ISO/IEC 10646, UCS, UTF*
- Kuhn, M.: *UTF-8 and Unicode FAQ*
- SC Unipad ein kostenfreier Unicode Editor

Element Information Item

Jedes XML-Dokument enthält mindestens ein *Element*, das `Document Element`.
Seine, wie auch die Grenzen aller anderen Elemente, werden durch die *Start*- und *Ende-Marke* (engl. *Tag*) markiert. Für den Sonderfall eines *leeren Elements* bildet die Start- auch zugleich die Ende-Marke. Als eine Konsequenz können diese Elemente keine weiteren Kindknoten besitzen.

Die XML-Spezifikation legt den Aufbau des Start-Tags wie folgt fest (In XML-Spezifikation nachschlagen) :

```
[40] STag      ::= '<' Name (S Attribute)* S? '>'
[41] Attribute ::= Name Eq AttValue
```

Mittels der Tag-Namen werden die Typen eines Dokumentes definiert. Sie werden später, in Verbindung mit einem Grammatikmechanismus wie XML-Schema, zur Gültigkeitsprüfung herangezogen.
Der Aufbau der Elementnamen ist ähnlich zu den aus den Programmiersprachen bekannten Regeln. Am Beginn muß ein Buchstabe, ein Unterstrich oder der Doppelpunkt stehen. Darauf können nahezu beliebige Zeichen folgen, die über ihre Unicoderepräsentation genau definiert sind.
Leerzeichen und sog. *white spaces* (vgl. Produktion 3 der XML-Spezifikation) wie Tabulatoren und Zeilenvorschübe sind nicht zugelassen. Desweiteren darf ein Elementname weder Auszeichnungssymbole, wie die öffnenden und schließenden Winkelklammern, enthalten, noch mit der Zeichenkette *XML* beginnen. Die Zeichenfolge *XML* ist -- in allen Schreibweisen -- für die Standardisierung reserviert und wird ausschließlich in W3C-Dokumenten verwendet.
Durch den Namespace Standard (siehe Abschnitt 1.3) wird dem Doppelpunkt, als Trennsymbol zwischen Namensraumkürzel und Elementnamen, eine besondere semantische Bedeutung zugeschrieben. Daher sollte -- obwohl er spezifikationsgemäß ein erlaubtes Zeichen darstellt -- von seiner Verwendung in Elementnamen abgesehen werden.

Oftmals wird -- insbesondere in der Praxis -- die existierende und notwendige Unterscheidung zwischen *Tag* und *Element* nicht getroffen.
Die Tags oder Marken drücken beschreibende Information über ein Element aus. Der durch den Tag ausgedrückte Elementname liefert somit lediglich deskriptive Information über die Natur des Elements. Hierzu können Worte einer natürlichen Sprache verwendet werden, jedoch auch beliebige andere identifizierende Zeichenketten. Üblicherweise sind jedoch sprechende Tags anzutreffen.

Über den Tag-Namen hinaus kann ein Startelement auch noch *Attribute* enthalten (Vgl. Produktion 41). Diese sind jedoch nicht vom Typ *Element* und werden daher im Abschnitt *Attribute Information Item* betrachtet.

Der Aufbau eines Elementnamens wird durch die Produktionen 4ff definiert (In XML-Spezifikation nachschlagen) :

```
[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
[5] Name     ::= (Letter | '_' | ':') (NameChar)*
[6] Names    ::= Name (S Name)*
[7] Nmtoken  ::= (NameChar)+
[8] Nmtokens ::= Nmtoken (S Nmtoken)*
```

Im Beispiel sind `Vorlesung`, `Titel` und `Hochschule` („normale“) Elemente, während `Pflichtfach` ein leeres Element darstellt.
Die Abbildung zeigt, daß auf der semantischen Ebene des Information Sets die syntaktische Unterscheidung zwischen Elementknoten mit Kindelementen und leeren Elementen des XML-Dokuments keine Berücksichtigung findet.

Eine Sonderstellung unter den Elementen eines Dokuments nimmt der ausgezeichnete Wurzelknoten ein, er wird auch durch das *Document Information Item* referenziert. Unterhalb dieses Knotens spannt sich der Dokumentbaum auf. Hierfür enthält jedes Element Information Item eine geordnete Menge (*children*) weiterer Elementknoten.
Die durch den Elementnamen verwirklichte Typisierung spiegelt sich im Information Set durch das Attribut *local name* wieder.

Darüberhinaus enthält jedes *Element Information Item* durch die Eigenschaft *namespace name* die Identifikation des Namensraumes, in dem dieses Element plaziert ist.
Das Namensraumkürzel, welches zur Identifikation eines Elements herangezogen wird, findet sich in der Eigenschaft *prefix*.
Der *local name* entspricht dem -- um Namensraumkürzel und trennenden Doppelpunkt gekürzten -- wiedergegebenen Elementnamen des XML-Dokuments.
Zusätzlich wird jeder Namensraum, der syntaktisch an die Attributdefinition angelehnt ist, in ein Element der ungeordneten Menge *namespace attributes* abgebildet, welche (nochmals) die Namensräume eines Elements beinhaltet.

Beispiel 2: Element mit deklariertem Namensraum

```
(1)...
(2)     <myNS:aParent xmlns:myNS="example.com">
(3)             <myNS:aElement/>
(4)     </myNS:aParent>
(5)...
```

Das Beispiel zeigt das leere Element `aElement` innerhalb des Elements `aParent`. Durch das Elternelement wird der Namensraum `example.com` deklariert und dem Kürzel `myNS` zugewiesen.
Gemäß den Prinzipien der Namensräume steht der auf dem Elternknoten deklarierte Namensraum auch in allen Kindknoten zur Verfügung. Daher enthält die Eigenschaft *in-scope namespaces* des Elements `aElement` auch die Namensräume der übergeordneten Elemente.
Das resultierende *Element Information Item* des Knotens `aElement` ergibt sich daher als (der Ausschnitt enthält nur die für das Beispiel relevanten Elemente):

```
local name    = aElement
namespace URI = example.com
prefix        = myNS
```

Nähere Ausführungen zur Bedeutung von Namensräumen und ihrer Verwendung finden sich im Abschnitt *Namensräume*.

Verweise auf die im Dokumentbaum nachfolgenden Knoten eines Elements werden in einer geordneten Liste *children* gesammelt. Ihre Inhalte sind sind vom Typ *Element Information Item*, *Character Information Item* und *Comment Information Item*.
Anhand der beiden Informationstypen *Element Information Item* und *Character Information Item* zeigen sich bereits die beiden Strukturierungsformen eines XML-Dokuments. Einerseits die durch die starke Verwendung von Elementen- und Attributen gekennzeichnete strukturierte Darstellung, andererseits die durch „eingestreuten“ Freitext entstehende charakteristische semistrukturierte Variante.
In beiden Fällen werden die textartigen Inhalte durch *Character Information Items* repräsentiert.
Das Beispiel zeigt die verschiedenen Auftretensformen exemplarisch. Der Inhalt der Elemente `title` und `organization` ist rein Zeichenketten-artig; jedoch mischt `vorlesung` strukturierten Inhalt (in Form der genannten Elemente) und unstrukturierte Information -- repräsentiert durch den Text `2002/03`.
Die XML-Spezifikation prägt für Zeichenketten-artige Inhalte, die optional durch *eingestreute* Elemente angereichert werden, den Begriff *mixed Content*.

children enthält jedoch keine Verweise auf die Attribute eines Elements. Diese sind durch die separate ungeordnete Menge *attributes* repräsentiert. Die Diskussion der als *Attribute Information Item* bezeichneten Mengenelemente findet sich im folgenden.

Die in der Abbildung dargestellte Beziehung *parent* verbindet jedes Element mit seinem übergeordneten.
Als Ziele dieser Referenz sind ausschließlich Ausprägungen von Document Information Item oder Element Information Item zugelassen.
Diese Festlegung untermauert nochmals die strikte Baumstruktur eines XML-Dokuments. Andernfalls müßte *parent* als Menge definiert werden.

Attribute Information Item

Das betrachtete Beispiel enthält, neben den Elementen, auch ein XML-Attribut.
Syntaktisch werden Attribute innerhalb eines Start-Tags plaziert und durch Namen-Wert-Paare ausgedrückt (In XML-Spezifikation nachschlagen) .

Der Information Set enthält folgende Eigenschaften zu jedem Attribut:

- **namespace name**: Namensraum des Attributs, falls definiert.
- **Lokaler Name**: Der um das eventuell definierte Namensraumkürzel bereinigte Attributname.
- **Präfix**: Namensraumkürzel des Namensraumes, innerhalb dessen das Attribut plaziert ist.
- **Normalisierter Wert**: Normalisierter Attributinhalt. Der Normalisierungsvorgang ist in Abschnitt 3.3.3 der XML-Spezifikation beschrieben (In XML-Spezifikation nachschlagen) .
  Unter anderem eliminiert er Zeilenumbrüche innerhalb des Attributinhalts und löst Entitätsreferenzen auf.
- **specified**: Boole'scher Wert, der angibt, ob das Attribut im XML-Dokument auftrat oder aufgrund einer Vorgabebelegung durch die DTD erzeugt wurde.
  Zur Ermittlung dieser Eigenschaft des Attribute Information Items ist die Definition und Referenzierung einer expliziten Grammatik notwendig.
- **Attributtyp**: Typ des Attributs. Zugelassene Belegungen sind: `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NMTOKEN`, `NMTOKENS`, `NOTATION`, `CDATA`, und `ENUMERATION`.
  Zur Ermittlung dieser Eigenschaft ist der Zugriff auf die DTD des Dokumentes notwendig. Ist dies nicht möglich, so ist der *attribute type* mit keinem Wert belegt.
- **Referenzen**: Handelt es sich bei dem Attribut um ein Referenzattribut (d.h. es ist als `IDREF(S)`,

`ENTITY`, `ENTITIES` oder `NOTATION` typisiert), so enthält diese Eigenschaft eine Verweisliste auf alle Auftreten des Attributwertes.

- **Eigentümerelement**: Bildet die Entsprechung zur *parent*-Eigenschaft des *Element Information Item*. Als solches enthält die Eigenschaft einen Verweis auf das Element, welches das Attribut beherbergt.

Im Vergleich zum *Element Information Item* erlaubt das Attribut keine weitere Unterstrukturierung (im XML-Sinne); insbesondere fehlen mengenwertige Eigenschaften zur Aufnahme der dann notwendigen Verweise. Stattdessen wird der gesamte Inhalt durch die Eigenschaft *normalized value* dargestellt. Daher dürfen innerhalb von Attributen keine (Meta-)Symbole wie die öffnende Winkelklammer auftreten, die als Starttags (miß-)interpretiert werden könnten (In XML-Spezifikation nachschlagen) .

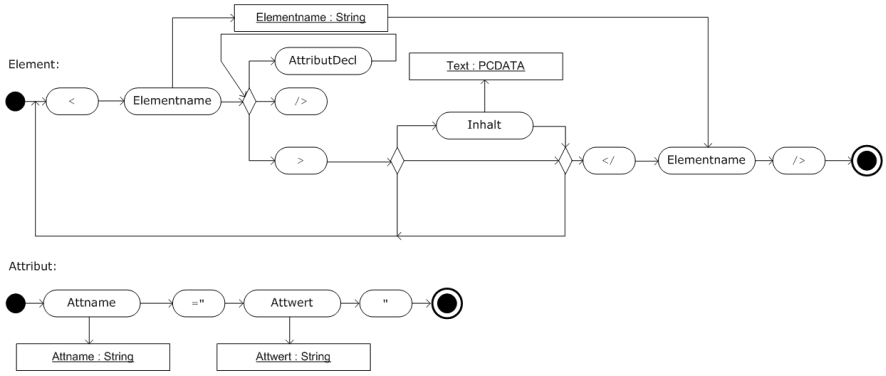
Auch die Form des Auftretens von Attributen innerhalb des definierenden Elements unterscheidet sich von der der Subelemente innerhalb eines Elements. Während Kindelemente durch die geordnete Liste *children* dargestellt werden, können Attribute (formalisiert in der ungeordneten Menge *attributes*) in beliebiger Reihenfolge angegeben werden, ohne die Dokumentsemantik zu verändern. Mehr noch, die Listenkonstruktion erlaubt das unterscheidbare mehrfache Auftreten desselben Elements. Diese Mimik ist für allgemeine Mengen, und damit für Attribute, nicht möglich.

Element vs. Attribut

Der Vergleich der Eigenschaften von Element und Attribut zeigt bereits, daß sich nicht weiter strukturierte Elemente auch durch Attribute darstellen ließen. Dies wirft innerhalb der Betrachtung der Syntax eines XML-Dokuments bereits die Frage nach der Organisation, und damit dem Entwurf, eines solchen auf.
Die bestehende XML-Spezifikation bleibt jedoch eine Anwendungs- oder Einsatzempfehlung zu dieser Fragestellung schuldig.
Aufgrund der inhärenten Einschränkungen der Attributprimitive bietet sich ihr Einsatz nur in einigen Sonderfällen an. Beispielsweise zur Darstellung deskriptiver Information über das enthaltende Element, die nicht Bestandteil der im XML-Dokument dargestellten Information ist. Hierbei kann es sich um Informationen höherer Ordnung, sog. Metainformation handeln.

Generell bieten sich Elemente immer dann an, wenn eine weitere Unterstrukturierung des Inhaltes gewünscht oder vielleicht zukünftig notwendig ist. Die Darstellungsform als Attribut würde in diesem Fall eine strukturelle Umorganisation des XML-Vokabulars erfordern, da die Spezifikation keine Unterstrukturierungsmöglichkeit für Attribute vorsieht.
Darüberhinaus gestatten Attribute keine Wiederverwendung in verschiedenen Bedeutungskontexten, da sie syntaktisch an das umgebende Element gebunden sind. Diese Einschränkung wird zwar durch die Einführung des Standards *XML Schema* weitgehend gemildert, jedoch nicht die zuvor genannte Mächtigkeitseinschränkung. Zusätzlich stellen Attribute die einzige Möglichkeit zur Typisierung des Inhaltes dar solange DTDs verwendet werden. Dieser Punkt dürfte jedoch durch den wachsenden Praxiseinsatz der XML Schemata immer mehr an Bedeutung verlieren.

Die Darstellung der Abbildung 5 faßt die syntaktischen Elemente abgekürzt zusammen:



Character Information Item

Die Betrachtung der Attribut- und Elementknotentypen im Information Set zeigt bereits die zwei grundlegenden Arten der Informationsdarstellung eines XML-Dokumentbaumes.
Die Eigenschaft *normalized value* des *Attribute Information Item*s kapselt den im XML-Dokument angegebenen Inhalt direkt im Informationsknoten. Der Datentyp der Eigenschaft ist für alle Dokumenttypen fixiert angebbar, da keine weitere Unterstukturierung von Attributen erfolgen kann. Entgegensetzt hierzu verläuft die Argumentationslinie für Elemente. Ihr Inhaltsmodell kann eine freie Mischung aus Zeichenketten-Daten und weiteren Elementen aufweisen. Die Länge der Zeichenketten ist hierbei nicht näher festgelegt. Daher können diese im minimalen Falle nur aus einem einzelnen Zeichen bestehen. (In XML-Spezifikation nachschlagen) .
Innerhalb des Information Sets eines Dokuments werden alle Zeichen im Rumpf eines Elements als Ausprägungen des *Character Information Item*s dargestellt.

Jedes Character Information Item stellt das im Dokument gegebene Zeichen gemäß ISO 10646-Codierung in der Eigenschaft *character code* dar. Die Werte können hierbei jedoch nur in den durch die Spezifikation vorgegebenen Grenzen variieren (In XML-Spezifikation nachschlagen) . Darüberhinaus genügt bereits die

Unterstützung der UTF-8 und -16-Darstellung zur Erfüllung der Spezifikationsanforderungen an konforme Prozessoren.
Häufig werden white-spaces (Leerzeichen, Tabulator, Zeilenvorschub, Wagenrücklauf) zur besseren visuellen Strukturierung des XML-Dokumentes eingesetzt. So enthält das Beispieldokument jeweils nach der schließenden Marke einen Zeilenvorschub. Unter Datengesichtspunkten handelt es sich hierbei jedoch um keine verwertbare Information. Die Angabe der Berücksichtigung bzw. Vernachlässigung im XML-Dokument existierender white-spaces kann in der DTD gesetzt werden. Ist keine solche Deklaration gesetzt oder existiert keine explizite Grammatik, so hat die Eigenschaft *element content whitespace* keinen Inhaltswert.
Der als `parent`-Eigenschaft realisierte Verweis auf das beherbergende Elternelement bildet den Abschluß der Eigenschaften des *Character Information Items*.
Im betrachteten Beispiel sind unterhalb der Elemente `organization` und `title` *Character Information Element*-Ausprägungen plaziert. Die Darstellung zeigt diese als Objekte (Unterhalb des `organization`-Knotens wurde aus Übersichtlichkeitsgründen auf die Darstellung verzichtet).

Eine Sonderrolle kommt den Zeichen zu, die auch als Metasymbole der Auszeichnungssprache dienen. Sie dürfen daher nicht in XML-Dokumenten auftreten.
Bei diesen Zeichen handelt es sich um die beiden Winkelklammern, die einfachen und doppelten Anführungszeichen sowie das *Kaufmanns-Und*. Um eine Fehlinterpretation zu vermeiden existieren hierfür vordefinierte Textersetzungsmuster.
Jeder spezifikationskonforme XML-Prozessor berücksichtigt diese Symbole und gibt sie in der korrekten Darstellung an die Applikation weiter; damit sind diese Fluchtsymbole (engl. *escape characters*) aus Applikationssicht vollkommen transparent.

Tabelle 5: Vordefinierte Textersetzungsmuster

Entitätsreferenz	Ausgedrücktes Zeichen
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Web-Referenzen 4: Weiterführendes ... Die in XHTML v1.0 vordefinierten Entitäten
Latin-1 Entities
Special Entities
Symbole

Comment Information Item

Zur Dokumentation steht innerhalb jedes XML-Dokuments die von SGML ererbte Kommentierungssyntax zur Verfügung.
Die Spezifikation erlaubt die Anbringung von Kommentaren an zwei Stellen im XML-Dokument:

- Nach dem Prolog. (In XML-Spezifikation nachschlagen)
- An jeder beliebigen Stelle des Inhalts, außerhalb von Markup-Symbolen. (In XML-Spezifikation nachschlagen)

Nicht erlaubt sind demnach Kommentare in Tags, d.h. innerhalb geöffneter Winkelklammern.
Dergleichen gilt für Kommentare selbst, was geschachtelte Kommentare verbietet.

Produktion 15 der XML-Spezifikation legt die Struktur wie folgt fest:

```
[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

Als Konsequenz sind innerhalb von Kommentaren alle Zeichen, auch Metasprachensymbole, zugelassen. Somit ist das beliebige „auskommentieren“ von Dokumentteilen möglich.
Als zentrale Einschränkung dürfen (aus SGML-Kompatibilitätsgründen) keine zwei aufeinanderfolgenden Trennstriche (*hyphen-minus*, ISO 10646 #x2D) innerhalb eines Kommentars auftreten, da diese fehlerhafterweise als Beginn des Kommentarendes interpretiert würden.

Der gesamte Inhalt eines Kommentars wird als uninterpretierte Zeichenkette in der Eigenschaft *content* des *Comment Information Item*s abgelegt.
Zusätzlich verweist jeder Kommentar über die bekannte *parent*-Eigenschaft auf seinen Elternknoten. Wie bereits durch die beiden Einsatzformen angedeutet, kann es sich hierbei ausschließlich um ein *Document Information Item* oder ein *Element Information Item* handeln.

Beispiel 3: Verschiedene Kommentarstrukturen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<Root>
(3)     <!-- this is a comment -->
(4)     <ElementA>
(5)             <ElementB>
(6)                     <!--
(7)                     <ElementC/>
(8)                     <ElementD att1="..."/>
(9)                     -->
(10)            </ElementB>
(11)    </ElementA>
(12)</Root>
```

Das Beispiel zeigt verschiedene Einsätze von Kommentaren. Zunächst eine einzeilige Anmerkung, die nur verschiedene Zeichen versammelt. Im Anschluß einen mehrzeiligen Kommentar, der auch XML-Strukturen beinhaltet. Ein prozessierender Zugriff auf den Kommentarinhalt ist jedoch nicht vorgesehen, und wird durch gängige Parser und APIs zumeist nicht unterstützt.

Processing Instruction Information Item

Im Gegensatz zu den prinzipiell in beliebigem Freitext formulierbaren Kommentaren, die üblicherweise zur Kommunikation mit einem menschlichen Leser des XML-Dokuments dienen, zielt die *Processing Instruction* und das zugehörige Element des Information Sets auf Kommentare, welche einen maschinellen Verarbeiter des XML-Dokuments, den XML-Prozessor, betreffen.

Im Grunde genommen läuft die Anreicherung eines XML-Dokuments mit Verarbeitungsinformation der Idee einer deskriptiven Auszeichnungssprache entgegen ...
Jedoch wurde für die XML beschlossen, nicht zuletzt aus Kompatibilitätsgründen zu SGML, dieses Sprachmerkmal beizubehalten. Eine mögliche weitere Erklärung könnte das syntaktische Aussehen der XML-Deklaration innerhalb des des Dokumentprologs sein. Ihre in Produktion 23ff festgelegte Struktur stellt eine Anwendung der Processing Instruction dar, auch wenn dies innerhalb der Spezifikation nicht explizit formuliert wird.

Die Syntax einer *Processing Instruction* lautet:

```
[16] PI       ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

Eine Processing Instruction wird demnach immer durch eine öffnende Winkelklammer und ein folgendes Fragezeichen eingeleitet. Daran schließt sich die Benennung der Applikation an, für die diese Instruktion eingefügt wurde. Optional können weitere Zeichen -- ausgenommen der Kombination aus Fragezeichen und schließender Winkelklammer -- folgen.
Das adressierte System kann beliebig identifiziert werden, jedoch ist die Zeichenkette *XML* in allen Variationen ausgeschlossen.
Unbedachterweise verbietet die Spezifikation jedoch nicht die Bildung von Namen, die *XML* als Präfix nutzen ... Jedoch sollte von der Nutzung solcher Konstruktionen abgesehen werden, da sie zur Verwirrung der (menschlichen) Leser beitragen.

Wie Kommentare auch können Processing Instructions an beliebiger Stelle innerhalb des XML-Dokuments auftreten: Vor Beginn des Wurzelelements sowie im Rumpf jedes Elements. Nicht gestattet ist ihre Angabe in Elementnamen und Attributen.
Ergänzend sei angemerkt, daß die Angabe von Processing Instructions auch innerhalb der *Document Type Definition* erfolgen kann. (siehe *Document Type Definition Information Item*).

Beispiel 4: Verschiedene Processing Instructions

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<?mySystem value="42"?>
(3)<root>
(4)   <?System2?>
(5)   <elementA>
(6)     <?System3 a="1" anotherValue?>
(7)  </elementA>
(8)</root>
```

Download des Beispiels

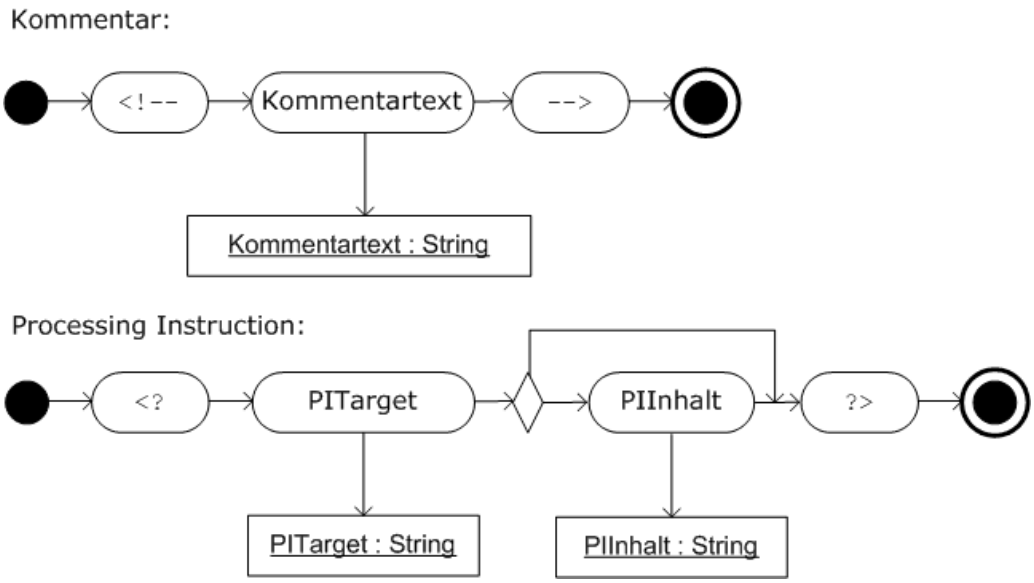
Übung 1: Processing Instructions

Begründen Sie mit Hilfe der XML-Spezifikation warum Processing Instructions nicht innerhalb von Elementen und Attributen zugelassen sind.
Hinweis: Es gibt mehr als eine Begründung!

Das *Processing Instruction Information Item* enthält die angesprochene Zielapplikation als Namen innerhalb der Eigenschaft *target*.
Der weitere Inhalt der Deklaration wird uninterpretiert als Zeichenkette in die Eigenschaft *content* übernommen.
Neben einem Verweis auf die Basis-URI der Processing Instruction wird durch *parent* das Elternelement -- entweder ein Knoten des Typs *Document Information Item* oder *Element Information Item* -- referenziert.

Zur Formalisierung der Identifikation der Zielapplikation empfiehlt die XML-Spezifikation die Verwendung des Sprachmittels *Notation*.

Die Darstellung der Abbildung 6 faßt die syntaktischen Elemente abgekürzt zusammen:



Namespace Deklaration Information Item

Jedem im XML-Dokument definierten Namensraum ist ein *Namespace Deklaration Information Item* zugeordnet. Es enthält die notwendigen syntaktischen Details zur Identifikation des Namensraumes:

- **prefix**: Das gewählte Präfix des Namensraumes, bzw. leer falls es sich um den Vorgabenamensraum handelt.
- **namespace name**: Der Name des Namensraumes, an den das Präfix gebunden ist.

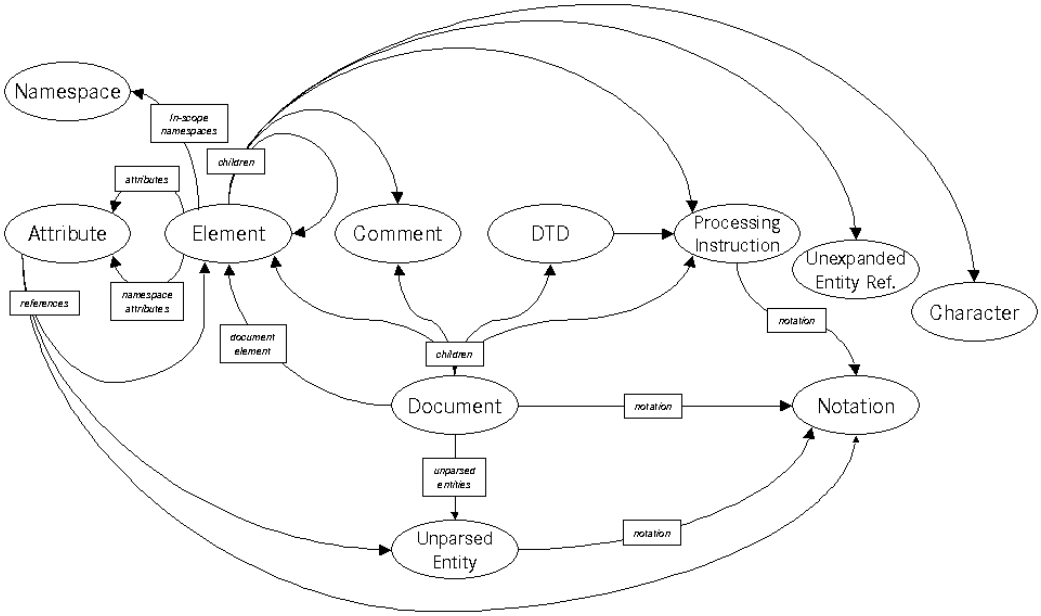
Beispiel 5: Beispiel eines Dokuments mit Namensräumen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<root>
(3)     <elementA>...</elementA>
(4)     <elementB xmlns="http://www.fh-furtwangen.de">...</elementB>
(5)     <elementC xmlns:abc="http://www.xyz.com">
(6)             ...
(7)             <abc:elementD/>
(8)     </elementC>
(9)</root>
```

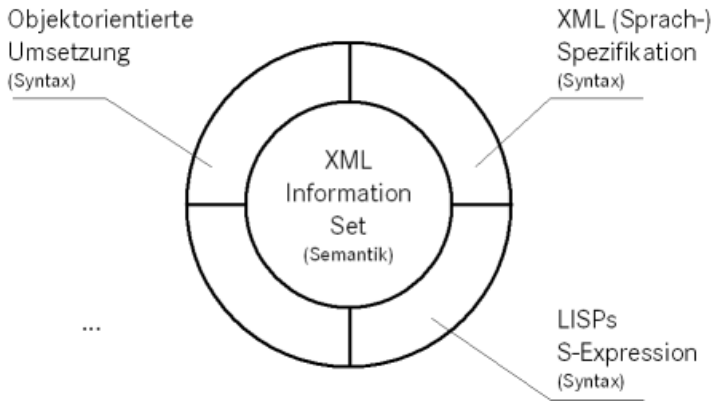
Für das Beispiel lauten die Namensräume wie folgt:

Elementname	Namensraum
root	(Das Element befindet sich im leeren Namensraum)
elementA	(Das Element befindet sich im leeren Namensraum)
elementB	http://www.fh-furtwangen.de
elementC	(Das Element befindet sich im leeren Namensraum)
elementD	http://www.xyz.com

Eine ausführliche Betrachtung zur Verwendung von Namensräumen findet sich im entsprechenden Abschnitt.



Die Graphik der Abbildung 7 stellt alle diskutierten Elemente des Information Sets in der Übersicht mit ihren Beziehungen dar. Zur Veranschaulichung wurde eine einfache Graphenstruktur gewählt, die alle Informationseinheiten als Knoten (darstellt als Ellipsen) und alle zugelassenen Beziehungen als gerichtete Kanten zwischen diesen enthält. Zusätzlich ist an die Kanten die Art der Beziehung angetragen.
Den Ausgangspunkt der baumartigen Struktur eines XML-Dokuments bildet die im Zentrum abgebildete Primitive `Document Information Item`, die alle weiteren Inhalte eines Dokuments über die `children`-Kante als Kindknoten enthält. Ferner fällt in dieser Darstellung besonders auf, daß lediglich `Element Information Items` über weitere Kindknoten verfügen und so die charakteristische XML-Struktur herausbilden. Alle übrigen Primitive dienen überwiegend als Blattknoten des Baumes.



Die Graphik der Abbildung 8 setzt die durch den Infoset-Standard definierte Semantik und die darauf aufsetzenden Syntaxen in Beziehung. Der XML-Basisstandard definiert hierbei nur eine von mehreren möglichen Syntaxen zur Darstellung von Infoset-Ausprägungen. Ebenso denkbar wäre der Einsatz anderer Darstellungen gleicher Mächtigkeit wie beispielsweise der S-Expression aus LISP oder objektorientierte Umsetzungen.

Auf Basis der Definitionen des Information Sets läßt sich ein beliebiges XML-Dokument, welches den Strukturierungsprinzipien des Infosets folgt, als *wohlgeformt* (*well-formed*) charakterisieren.

Definition 6: Wohlgeformtes XML-Dokument

Ein textartiges Objekt, dessen Inhalt folgenden Anforderungen genügt:

- Das XML-Dokument nutzt eine DTD, oder enthält die Deklaration `standalone="yes"`
- Zu jedem Start-Tag existiert genau ein Ende-Tag. Bei leeren Elementen können diese zu einem Tag zusammenfallen.
- Korrekte Elementschachtelung, d.h. Elemente überlappen einander nicht.
- Genau ein Wurzelelement.
- Alle Attributwerte sind in einfachen oder doppelten Anführungszeichen.
- Kein Start-Tag (oder Tag der ein leeres Element einleitet) enthält zwei oder mehr Attribute desselben Namens.
- Keine Kommentare oder Processing Instructions innerhalb

von Tags.
- Kommentare beginnen und enden mit genau zwei Bindestrichen.
- Die Sonderzeichen < und & treten nicht innerhalb von Elementinhalten oder Attributwerten auf.

siehe XML-Spezifikation

Der Textstrom des Beispiels 6 zeigt ein nicht-wohlgeformtes XML-Dokument, welches gegen eine Reihe der in Definition 6 verstößt:

Beispiel 6: Ein nicht wohl-geformtes XML-Dokument

```
(1)<?xml version="1.0"?>
(2)<root>
(3)     <elementA att=a oder b>
(4)             <elementB> iff a<b ==> ...
(5)     </elementA>
(6)     <elementC att1="42" att1="3.14">
(7)             <elementD <?do-something?> >
(8)     </elementC>
(9)             </elementD>
(10)    <!---- dies ist nicht erlaubt ---->
(11)</root>
```

Download des Beispiels

So findet sich in Zeile 3 ein nicht in die erforderlichen Anführungszeichen eingeschlossener Attributwert.
Der textuelle Elementinhalte des in Zeile 4 geöffneten Elements `elementB` enthält ein öffnendes Winkelklammersybol, welches um Fehler während des Einlesevorganges zu vermeiden durch die alternative Zeichensequenz `<` hätte ersetzt werden müssen. Darüberhinaus fehlt das korrekte schließende Tag zum Öffnenden.
Innerhalb des Elements `elementC` der Zeile 6 wird zweifach ein identisch benanntes Attribut definiert.
Im öffnenden Tag des in Zeile 7 definierten Elements `elementD` findet sich eine -- dort nicht zugelassene -- Processing Instruction.
Überdies überlappen sich die Elementgrenzen der Elemente `elementC` und `elementD` und zusätzlich wird der in Zeile 10 plazierte Kommentar nicht durch die erforderlichen genau zwei Bindestriche eingegrenzt.

2.2 XML-Namensräume

Namensräume

Die XML-Namensräume wurden schon verschiedentlich erwähnt. Sie bilden die wichtigste, und offensichtlichste Weiterentwicklung der XML-Urspezifikation seit ihrer Veröffentlichung.
Trotz ihrer engen Beziehung zum XML-Kernstandard bildet die Recommendation *Namespaces in XML* eine eigenständige Spezifikation. Aufgrund der engen syntaktischen Beziehung zum XML-Standard und der großen praktischen Bedeutung, sowie des Einflusses auf die weitere Entwicklung verschiedenster Sekundärstandards und XML-Sprachen, werden die Namensräume explizit in der Neuauflage des XML-Standards berücksichtigt. Einen Beleg hierfür bildet die Anmerkung zu Abschnitt 2.3 *Common Syntactic Constructs*. Dort wird von der -- laut Syntaxproduktion 5 erlaubten -- Verwendung des Doppelpunktes in Elementnamen abgeraten. Dies geschieht, um Mehrdeutigkeiten, oder schlichtweg der Verwirrung des Anwenders, vorzubeugen, da es sich beim Doppelpunkt um ein Symbol besonderer Bedeutung innerhalb der Namensraumdeklarationen handelt.

Warum Namensräume?

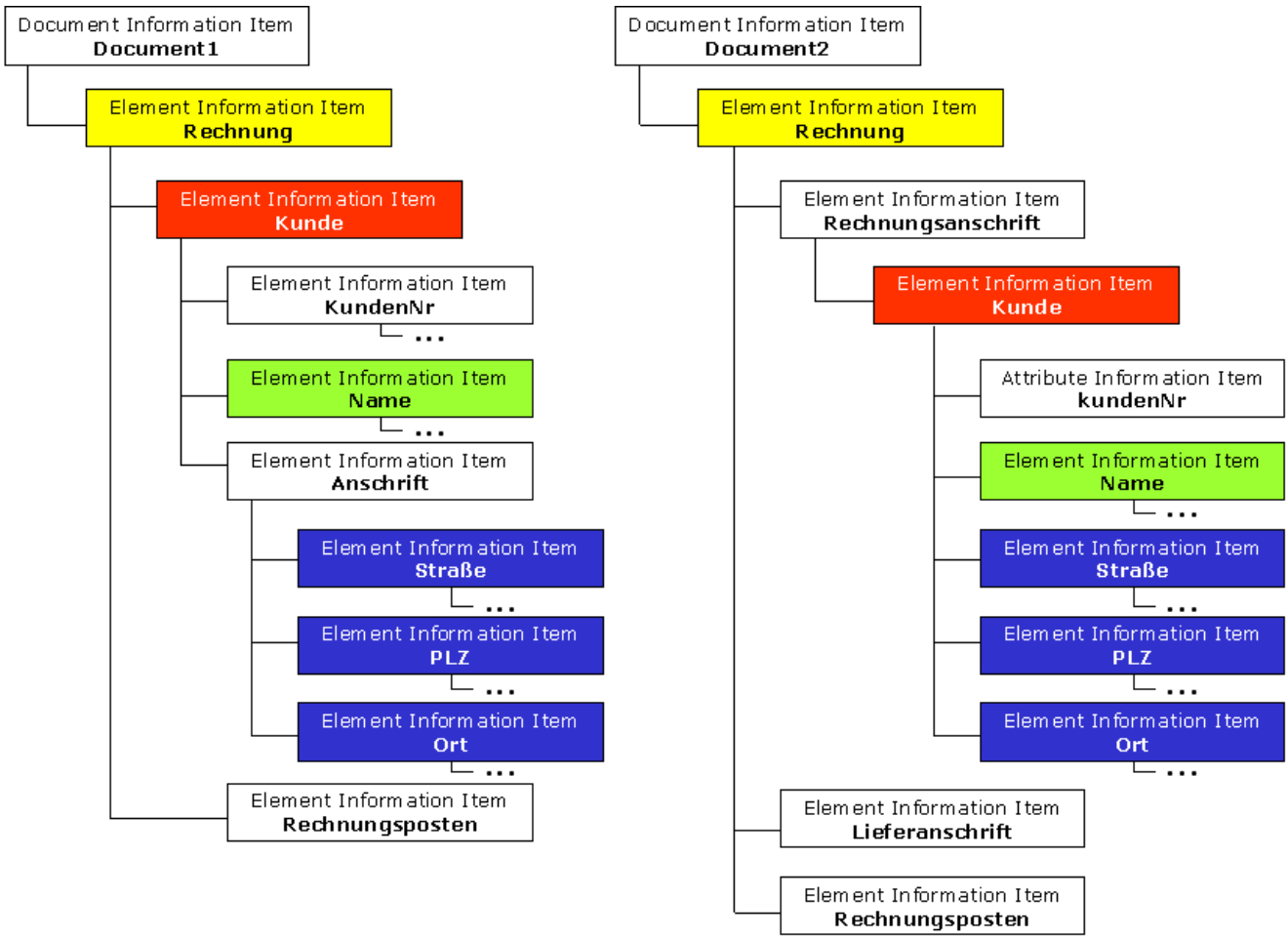
Die breite Entwicklung immer neuer XML-Sprachen führt zwangsläufig zu Mehrfachentwicklungen für ähnliche oder identische Problemstellungen. Technisch betrachtet äußerst sich dies -- bei natürlichsprachlicher Benennung der Elemente -- durch die Verwendung identischer Bezeichner in verschiedenen XML-Sprachen. Hierbei bilden die verschiedenen Sprachen Anwendungskontexte, innerhalb derer die Bezeichner, durch Einbezug der Anwendungssemantik, eindeutig sind; andernfalls kann unterstellt werden, daß bereits durch die Sprachentwicklung andere Benennungskonventionen gewählt worden wären.
In der Konsequenz der Verfügbarkeit verschiedenster XML-Sprachen für beliebige Anwendungsbereiche entsteht der (berechtigte) Wunsch existierende Sprachfragmente in eigene Sprachen zu integrieren, um so zeitraubenden und vielfach fehleranfälligen Mehrfachentwicklungen vorzubeugen. Jedoch tritt bei diesem Integrationsszenario die u. U. kontextabhängige Elementeindeutigkeit zu Tage.
Das Beispiel zeigt zwei Dokumente identischen Informationsumfanges, die lediglich strukturell differieren.

Beispiel 7: Ein Rechnungsdokument

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<Rechnung>
(3)   <Kunde>
(4)      <KundenNr>4711</KundenNr>
(5)      <Name>Max Mustermann</Name>
(6)      <Anschrift>
(7)         <Straße>Musterplatz 1</Straße>
(8)         <PLZ>12345</PLZ>
(9)         <Ort>Musterstadt</Ort>
(10)      </Anschrift>
(11)   </Kunde>
(12)   <Rechnungsposten>
(13)    ...
(14)    </Rechnungsposten>
(15)</Rechnung>
```

Beispiel 8: Eine alternative Rechnungsstruktur

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<Rechnung>
(3)     <Rechnungsanschrift>
(4)             <Kunde kundenNr="4711">
(5)                     <Name>Max Mustermann</Name>
(6)                     <Straße>Musterplatz 1</Straße>
(7)                     <PLZ>12345</PLZ>
(8)                     <Ort>Musterstadt</Ort>
(9)             </Kunde>
(10)    </Rechnungsanschrift>
(11)    <Lieferanschrift>
(12)   ...
(13)   </Lieferanschrift>
(14)    <Rechnungsposten>
(15)    ...
(16)    </Rechnungsposten>
(17)</Rechnung>
```



Die beiden Bäume mit *Information Set*-Ausprägungen zeigen die Struktur der Beispieldokumente. Dabei sind Knoten die den selben Inhalt repräsentieren mit identischen Farben unterlegt, unabhängig davon um welchen Knotentyp es sich handelt. Die *Character Information Item* Knoten wurden aus Übersichtlichkeitsgründen weggelassen und durch Punkte angedeutet, sie sind jedoch für die vorliegende Betrachtung nicht von Interesse.

Einige der Elemente und Attribute werden in beiden Dokumenten mit gleichen Inhalten verwendet; z.B. `Name`, `Ort` oder `PLZ`. Dies äußert sich in identischen Teilbäumen unterhalb der Information Set-Knoten welche diese XML-Elemente repräsentieren. Hieraus läßt sich ableiten, daß die beiden vorgestellten Sprachen an den genannten Stellen keine strukturelle Differenz aufweisen.
Dagegen unterscheiden sich die Kindknoten der Elemente `Rechnung` und `Kunde` hinsichtlich ihrer Struktureigenschaften. So folgt im ersten Beispieldokument auf das `Rechnung`-Element direkt der `Kunde`, während im zweiten XML-Dokument zunächst ein Element mit dem Namen `Rechnungsanschrift` erwartet wird.
Dergleichen gilt für die Kindelemente des `Kunden`. Im zweiten Beispieldokument wird die diesem Element untergeordnete Kundennummer durch ein Attribut (`kundenNr`) dargestellt. Dagegen codiert das erste Beispiel diese Information direkt in den Elementinhalt.

Solange die beiden Dokumente in unterschiedlichen Anwendungswelten (Unternehmen o. ä.) verwendet werden, ist der gewählte Ansatz nicht problematisch. Bedenklich wird er jedoch in mindestens zweierlei Hinsicht:
Zunächst bei der „Mischung“ der beiden Dokumente. Dieser Wunsch tritt bei praktischen Problemstellungen häufig auf, wenn es um die Übernahme von XML-codierten Daten in ein anderes XML-Dokument geht. In der Konsequenz folgt das entstehende Zieldokument nicht mehr den Strukturierungsregeln eines der Ausgangsdokumente; mithin entsteht eine neue Dokumentstruktur, deren Regeln nicht explizit dokumentiert sind.
Eine weitaus größere Herausforderung stellt die Zusammenfassung und Veröffentlichung von XML-Strukturen in sog. Schemabibliotheken oder Datenbanken dar. Hier werden zwar die Dokumente nicht vereinigt, jedoch offenbart sich die gleiche Anwendungsdomäne (z.B. Rechnungsverwaltung, Stücklisten, Produktstrukturen) als problematisch, da sie die XML-Strukturen in direkte Konkurrenz treten läßt. In Zeiten immer stärker werdenden ökonomischen Flexibilisierungsdruckes erweist sich dies als äußerst kontraproduktiv, im Hinblick auf eine angestrebte Standardisierung. Die offene Konkurrenz verschiedener *Dialekte* innerhalb einer Domäne verzögert damit oft die Entscheidung zum Einsatz eines Sprachformates.

Einen anderen interessanten Anwendungsfall stellt der ausdrückliche Wunsch nach der Einbettung fremder Sprachelemente dar. Diese Form der Wiederverwendung knüpft an das durch öffentlich verfügbare XML-Formate eröffnete Anwendungsfeld an. Da nicht in jedem Fall ein alle Anforderungen erfüllendes existierendes XML-Format ermittelt werden kann, jedoch verschiedene vorhandene Formatteile des gewünschten Umfanges abdecken, entsteht der Wunsch nach einer selektiven Weiterverwendung. Ein bekanntes Beispiel bilden Freitexte in beliebigen XML-Sprachen, welche auf Teile des (X)HTML-Sprachumfanges zurückgreifen. Gleichzeitig ist damit die Semantik der Elemente durch den zugehörigen W3C-Standard festgelegt. XHTML selbst stellt ein interessantes Anwendungsbeispiel für die gemeinsame Verwendung verschiedener XML-Sprachen in einem Dokument dar. So können Web-Seiten neben den bekannten Textstrukturen (XHTML) auch mathematische Symbole und Formeln (in der XML-Sprache MathML) und Vektorgraphiken (in der XML-Sprache SVG) enthalten.
Als Nebeneffekt der Wiederverwendung existierender XML-Sprachen verringern sich mögliche Fehlerquellen, was in der Konsequenz zur Erhöhung der Qualität der entstehenden Sprachen führt.

Zusammenfassend lassen sich die (Hinter-)Gründe der Namensraumeinführung wie folgt darstellen:

- Wiederverwendung bestehender (fremder) XML-Strukturen in eigenen Dokumenten.
- Wunsch nach breiteren Standards.
- Verringerung des Designaufwandes.
- Nutzung bereits gesammelter Designerfahrung.
- Zusammenführung verschiedener XML-codierter Inhalte (*heterogeneous content syndication*).

Definition 7: Namensräume

XML-Namensräume stellen eine XML-basierte Syntax zur Verfügung um Element- und Attributnamen eines Vokabulars eindeutig zu identifizieren und so Bedeutungsüberschneidungen durch gleichbenannte Elemente- oder Attribute in zu unterscheidenden Vokabularen auszuschließen. XML-Namensräume bilden damit die notwendige Voraussetzung zur freien dezentralen Entwicklung eigener Vokabulare ohne die Möglichkeit einer späteren Syndikatisierung zu verlieren.

Konzept der Namensräume:
Die Recommendation *Namespaces in XML* definiert die Syntax und Semantik der Namensräume. Ihr Konzept wurde rund ein Jahr nach Verabschiedung der ersten XML-Version eingeführt. Daher wurde der Kompatibilität mit bereits existierenden XML-Dokumenten große Priorität eingeräumt.

Grundidee der Namensräume ist es, die Element- und Attributnamen dergestalt zu erweitern, daß (auch nach Vereinigung beliebiger Dokumente wieder) eineindeutige Bezeichner entstehen. Dies könnte durch anwenderdefinierte Erweiterungen geschehen, sie trügen jedoch wiederum die Gefahr in sich, daß sie unbeabsichtigt mehrfach benutzt würden.
Daher scheidet der unkoordinierte Einsatz solcher Namenserweiterungen aus. Jegliche Koordination

bedingt jedoch inhärent eine zentrale Vergabestelle zur Registrierung der vergebenen Namen, die über die Eindeutigkeit wacht und Mehrfachnutzungen unterbindet.
Die Einführung einer solchen Stelle hätte jedoch einen unüberschaubaren Verwaltungsaufwand bedeutet, den das W3C nicht zu leisten im Stande wäre. Man nehme nur als Vergleich das Vergabeverfahren von Einträgen des Internet Domain Name Systems (DNS), welches bereits dezentral durch die einzelnen nationalen Domain-Registrars gehandhabt wird. Der dort anzutreffende Aufwand hätte sich für XML-Namensräume potenziert, legt man pro Domainadresse mehrere Namensräume zugrunde.

Ziel des W3C war es, durch die Namensräume einen gleichermaßen mächtigen als auch leicht zu handhabenden und zu administrierenden Identifikationsmechanismus zu etablieren. Offenkundig wird diesem Anspruch nur ein (überwiegend) dezentraler, aber dennoch die Eineindeutigkeit garantierender, Ansatz gerecht.
Diesen Anforderungen genügt das aus IETF RFC 2396 bekannte Namensschema der *Uniform Resource Identification* (URI) (später aktualisiert in IETF RFC 2732). Es kombiniert zentrale und dezentrale Elemente in der Handhabung, und ermöglicht so -- trotz Existenz und Pflege einer zentralen Registratur -- größtmögliche Flexibilität in der Anwendung. Der bekannteste Einsatz von URI-Namen ist der im World-Wide-Web allgegenwärtige *Uniform Ressource Locator* (URL) (IETF RFC 1738); einer Untermenge der URI.
Die zentrale Komponente findet sich im Domainnamen verwirklicht. Er ist entweder durch die IP-Adresse (konkret: IPv4-Adresse; im Falle des RFC 2732: der IPv6-Adresse) oder deren literaler Repräsentation gegeben. Unterhalb der Domainebene kann durch deren Verwalter eine beliebige Strukturierung vorgenommen werden. Die verschiedenen Ebenen werden dabei durch ISO-10646/ASCII #x2F „/“ voneinander abgetrennt.
Wie auch bereits bei URLs notwendig, ist das Schema (*URI scheme*) (z.B. `http`) zwingend mitanzugeben.

Trotz der Möglichkeit XML-Namensräume durch URLs zu identifizieren handelt es sich dabei nicht die Bezeichnung einer Internetquelle. Die verwendete Zeichenkette dient ausschließlich Benennung der im Namensraum versammelten XML Element Information Items und Attribute Information Items.
Die Auflösung des Namensraumbezeichners durch einen XML-Prozessor ist nicht vorgesehen.

Nachfolgend ist die in definierte Syntax einer URI wiedergegeben. Sie wurde behutsam an die in der XML-Spezifikation verwendete BNF-Notation (In XML-Spezifikation nachschlagen) angepaßt, ohne jedoch die Produktionen in ihrer Struktur zu verändern.

```
[URI1]  URI-reference ::= (absoluteURI | relativeURI)? ("#" fragment)?
[URI2]  absoluteURI   ::= scheme ":" ( hier_part | opaque_part )
[URI3]  relativeURI   ::= ( net_path | abs_path | rel_path ) [ "?" query ]
[URI4]  hier_part     ::=  ( net_path | abs_path ) ("?" query)?
[URI5]  opaque_part   ::= uric_no_slash uric?
[URI6]  uric_no_slash ::= unreserved | escaped | ";" | "?" | ":" | "@" |
                          "&" | "=" | "+" | "$" | ","
[URI7]  net_path      ::= "//" authority abs_path?
[URI8]  abs_path      ::= "/" path_segments
[URI9]  rel_path      ::= rel_segment abs_path?
[URI10] rel_segment   ::= (unreserved | escaped |
                          ";" | "@" | "&" | "=" | "+" | "$" | "," )+
[URI11] scheme        ::= alpha (alpha | digit | "+" | "-" | "." )*
[URI12] authority     ::= server | reg_name
[URI13] reg_name      ::= ( unreserved | escaped | "$" | "," |
                          ";" | ":" | "@" | "&" | "=" | "+" )+
[URI14] server        ::= ((userinfo "@")? hostport)?
[URI15] userinfo      ::= ( unreserved | escaped |
                          ";" | ":" | "&" | "=" | "+" | "$" | "," )*
[URI16] hostport      ::= host (":" port)?
[URI17] host          ::= hostname | IPv4address
[URI18] hostname      ::= ( domainlabel "." )* toplabel (".")?
[URI19] domainlabel   ::= alphanum | alphanum *( alphanum | "-" ) alphanum
[URI20] toplabel      ::= alpha | alpha (alphanum | "-" )* alphanum
[URI21] IPv4address   ::= digit+ "." digit+ "." digit+ "." digit+
[URI22] port          ::= digit*
[URI23] path          ::= (abs_path | opaque_part)?
[URI24] path_segments ::= segment ("/" segment)*
[URI25] segment       ::= pchar* (";" param)*
[URI26] param         ::= pchar*
[URI27] pchar         ::= unreserved | escaped |
                          ":" | "@" | "&" | "=" | "+" | "$" | ","
```

```
[URI28] query         ::= uric*
[URI29] fragment      ::= uric*
[URI30] uric          ::= reserved | unreserved | escaped
[URI31] reserved      ::= ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
                          "$" | ","
[URI32] unreserved    ::= alphanum | mark
[URI33] escaped       ::= "%" hex hex
[URI34] hex           ::= digit | "A" | "B" | "C" | "D" | "E" | "F" |
                          "a" | "b" | "c" | "d" | "e" | "f"
[URI35] digit         ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                          "8" | "9"
[URI36] uric_no_slash ::= unreserved | escaped | ";" | "?" | ":" | "@" |
                          "&" | "=" | "+" | "$" | ","
```

Die Produktionen `alphanum`, `lowalpha` sowie `upalpha` zur Konstruktion der alphanumerischen Namen wurden aus Übersichtlichkeitsgründen weggelassen.

Neben einigen anderen gängigen URI-Varianten stellt das nachfolgende Beispiel einige der möglichen syntaktisch korrekten URIs zusammen, die für die späteren Betrachtungen von Interesse sind.

Beispiel 9: Gültige URIs

```
(1)http://www.wi.fh-furtwangen.de
(2)http://meinrechner.wi.fh-augsburg.de
(3)mailto:mario@jeckle.de
(4)ftp://ftp.shareware.com
(5)http://www.jeckle.de/xml/vorlesung/script.htm#Namespaces
(6)#EinfuehrungUndUeberblick
(7)urn:oasis:names:specification:docbook:dtd:xml:4.1.2
(8)urn:oid:1.3.6.1.2.1.27
(9)org.omg/standards/UML
```

Exkurs: URIs, URLs, URNs ...

Vielfach wird in der Praxis die Abgrenzung der im Internet gebräuchlichen Adressierungs- und Identifikationsmechanismen nicht trennscharf vollzogen.
Darüberhinaus trat im Laufe der Entwicklung eine merkliche Bedeutungsverschiebung insbesondere zwischen der *Uniform Resource Identifikation* und den als WWW-Adressen genutzten *Uniform Resource Locators* ein.

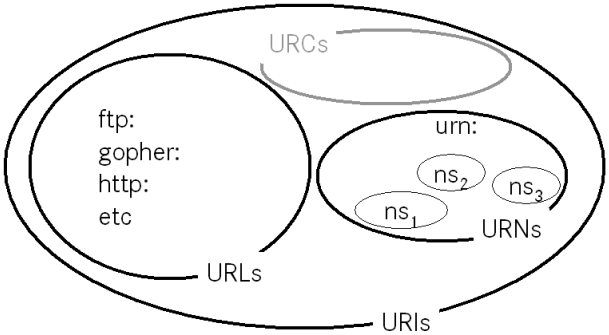
Gegenwärtig wird die Begriffsabgrenzung wie in Abbildung 10 schematisch dargestellt vollzogen:

- *Uniform Resource Identification* (URI) dient als abstrakter Oberbegriff eineindeutig identifizierbarer Web-Ressourcen.
  Konzeptionell sind URIs:
  - über Zeit und Raum eindeutig
  - für Menschen leicht zu merkend
  - mit keinerlei Registrierungskosten verbunden
  - unabhängig von der tatsächlichen Lokalisation der so identifizierten Ressource

  Der URI-Raum zerfällt in die disjunkten Bezeichnerschemata URL, URN und URC.
- *Uniform Resource Location* (URL) bezeichnet den physischen Aufenthaltsort einer Ressource, etwa den Ablageort einer HTML-Seite.
  Beispiele:
  - `http://www.jeckle.de/vorlesung/xml/script.html`
  - `http://www.wi.fh-furtwangen.de/`
  - `mailto:mario@jeckle.de`
  - `ftp://example.org/aDirectory/aFile`
  - `news:comp.infosystems.www`
  - `tel:+1-816-555-1212`
  - `ldap://ldap.example.org/c=GB?objectClass?one`
  - `urn:oasis:SAML:1.0`
- *Uniform Resource Name* (URN) bezeichnen den eineindeutigen Namen einer beliebigen Resource. Für die URN existiert kein Auflösungsmechanismus durch den die physische Lokation ermittelt werden könnte. URNs dienen daher ausschließlich der eindeutigen Benennung!
  Syntaktisch folgt auf das definierte Kürzel `urn` eine Zeichenkette welche eine Weiterklassifikation der Ressource gestattet. Hierdurch wird eine weitere Partitionierung des URN-Raumes erzielt. Diese *namespace ID* genannten Zeichenketten unterliegen einem globalen Registrierungszwang um ihre Eindeutigkeit zu gewährleisten. Diese Unterstrukturierungen sind in der Abbildung als ns_1 bis ns_3 benannt.

Beispiele:
- urn:oasis:names:specification:docbook:dtd:xml:4.1.2
- urn:oid:1.3.6.1.2.1.27

- *Uniform Resource Citation* (URC) schlägt die Brücke zwischen Lokationsbezeichnung und reiner Benennungskonvention. Eine URC verweist in eine Metadatenstruktur welche die physischen Ressourcen-Aufenthaltsorte katalogisiert.
  Dieser URI-Typ ist jedoch gegenwärtig kaum verbreitet.



Web-Referenzen 5: Weiterführende Links
•URIs, URLs, and URNs: Clarifications and Recommendations
•The Anatomy of an URL

Verwendung von Namensräumen:
Am naheliegendsten wäre nach der Zielsetzung der Verwendung von URIs zur eindeutigen Benennung von XML-Element- und Attributnamen, die URI direkt vor dem XML-Bezeichner zu plazieren, evtl. separiert durch ein Trennsymbol wie den Doppelpunkt „:“.
Hieraus entstünden dann, auf jeden Fall eindeutige, Element- und Attributnamen wie beispielsweise für das erste Beispieldokument dieses Kapitels (die URI `http://www.example.com/sales` werde zur Identifizierung verwendet):

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)     <http://www.example.com/sales:Rechnung>
(3)             <http://www.example.com/sales:Kunde>
(4)                     <http://www.example.com/sales:KundenNr>4711</http://www.example.com/
sales:KundenNr>
(5)                     <http://www.example.com/sales:Name>Max Mustermann</http://www.
example.com/sales:Name>
(6)                     <http://www.example.com/sales:Anschrift>
(7)                             <http://www.example.com/sales:Straße>Musterplatz 1</http://
www.example.com/sales:Straße>
(8)                             <http://www.example.com/sales:PLZ>12345</http://www.example.
com/sales:PLZ>
(9)                             <http://www.example.com/sales:Ort>Musterstadt</http://www.
example.com/sales:Ort>
(10)                    </http://www.example.com/sales:Anschrift>
(11)            </http://www.example.com/sales:Kunde>
(12)    <http://www.example.com/sales:Rechnungsposten>
(13)            ...
(14)    </http://www.example.com/sales:Rechnungsposten>
(15)</http://www.example.com/sales:Rechnung>
```

Bei entsprechender Nachbearbeitung des zweiten Beispieldokumentes mit einem anderen URI-identifizierten Namensraum, entstehen eindeutige Element- und Attributnamen, die nicht mehr kollidieren.

Jedoch verstößt diese Lösung gegen die in Produktion 5 der XML-Spezifikation formulierte syntaktische Einschränkung. Sie erlaubt das in URIs elementare Pfadtrennersymbol („/“) (aus den URI-Produktionen 8, 24 und 31) nicht in XML-Namen (#x2F findet sich nicht in den in Produktion 85 aufgeführten Unicode-Blöcken).
Die Integration der Namensräume auf diesem Weg hätte daher eine Modifikation der XML-Spezifikation nach sich gezogen. Diese erweiternde Aufweichung der zugelassenen Namen für Elemente und Attribute hätte jedoch mit der Kompatibilität zu SGML gebrochen, und somit eine der Grundforderungen der XML-Entwicklung verletzt.
Darüberhinaus ist die Spezifikation vollständiger URIs für Menschen „unhandlich“ und reduziert die Lesbarkeit der entstehenden XML-Dokumente.

Als Ausweg und pragmatischer Kompromiß zwischen eineindeutigen Namenspräfixen und Lesbarkeit wurde daher ein zweistufiges Verfahren eingeführt. Es erlaubt die Zuordnung von URIs zu Präfixen. Dieser Vorgang wird als „Bindung“ bezeichnet.
Diese Präfixes können Attributen oder Elementen vorangestellt werden, um sie in bestimmte

Namensräume zu übernehmen.
Für die Präfixe gelten dieselben Bildungsgesetze wie für die Element- und Attributnamen. Im Einzelnen legt die *Namespace Recommendation* fest: (im XML-Namespace-Dokument nachschlagen)

```
[NS7] Präfix     ::= NCName
[NS4] NCName     ::= (Letter | '_') (NCNameChar)*
[NS5] NCNameChar ::= Letter | Digit | '.' | '-' | '_'
                   | CombiningChar
                   | Extender
```

Anmerkung: Die rechten Seiten der Produktionen beziehen sich entweder auf die dargestellten Definitionen des Namespace-Standards oder auf Syntaxregeln der XML-Recommendation.

Die Bindung einer URI an ein -- gemäß Produktion NS7 frei wählbares -- Präfix geschieht durch das reservierte Attribut `xmlns`.
Die Syntax hierfür wird mit

```
[NS2] PräfixedAttName ::= 'xmlns:' NCName
```

angegeben.

Nach der Bindung der URI an das Präfix kann dieses jedem Element oder Attribut vorangestellt werden, um es in den Namensraum zu übernehmen.
Hierdurch verändert sich die Produktion `Name` aus der XML-Spezifikation zum *qualifizierten Namen*, der durch die Voranstellung des Präfixes entsteht. Der rechts vom trennenden Doppelpunkt folgende Elementname stellt den lokalen Namen (innerhalb des Namensraumes dar). Dieser lokale Name darf *keinen* Doppelpunkt mehr enthalten; insofern schränkt Produktion NS8 in Verbindung mit NS4 die Festlegung der Produktion 5 der XML-Spezifikation ein.

```
[NS6] QName     ::= (Präfix ':')? LocalPart
[NS8] LocalPart ::= NCName
```

Während der Verarbeitung eines XML-Dokuments, das Namensräume nutzt, ersetzt ein XML-Prozessor jedes Auftreten eines deklarierten Präfixes transparent durch die gebundene URI.
Prozessoren, welche die Namensraum-Spezifikation unterstützen, werden als *namespace aware* bezeichnet. Alle anderen Prozessoren treffen die durch `NS6` eingeführte Unterscheidung zwischen *Präfix* und *LocalPart* eines qualifizierten Namens nicht und betrachten die Kombination aus Präfix und Element- bzw. Attributnamen als Bezeichner. Die Präfix-URI-Bindung durch das `xmlns:...`-Attribut wird hierbei als gewöhnliches XML-Attribut betrachtet und führt daher zu keinen Validierungsfehlern. (Die Einschränkung der Produktion 5, ein Name dürfe nicht mit der Zeichenfolge `(('X'|'x') ('M'|'m') ('L'|'l'))` beginnen, stellt in der XML-Spezifikation lediglich einen Hinweis dar.)

Semantisch bildet die durch `xmlns` eingeleitete Deklaration ein *Pseudoattribut*, da es für die maschinelle Verarbeitung vorbehalten und mit festelegter Bedeutung ausgestattet ist, welche durch den XML-Dokumentautor nicht verändert werden kann.
Zusätzlich werden Namensraumdeklarationen durch Programmiersprachenschnittstellen nicht den gewöhnlichen Attributen gleichgestellt betrachtet, sondern nehmen, wie auch im Information Set, dort eine Sonderstellung ein.

Anmerkung: Auf Webseiten und in Mailinglisten finden sich manchmal Formulierungen der Struktur `{`*`namespaceName`*`}`*`elementName`* (z.B. `{http://www.w3.org/2001/XMLSchema}element` oder `{http://www.w3.org/1999/XSL/Transform}template`).
Hierbei handelt es sich um eine zwar geläufige, aber *nicht spezifikationskonforme Schreibweise*!
Sie dient lediglich dazu, das prinzipiell beliebig wählbare Präfix einzusparen und den gewählten Namensraum hervorzuheben.
Strukturen dieses Stils sind jedoch keine gültigen XML-Dokumente!

Angewendet auf das betrachtete Beispiel läßt sich die URI `http://www.example.com/sales` an das Präfix `myNS1` binden. Diese Bindung steht im definierenden Element (local name: `rechnung`) und allen untergeordneten zur Verfügung.

Beispiel 10: Dokument mit W3C-konformen Namensräumen

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<myNS1:Rechnung xmlns:myNS1="http://www.xyz.com/sales">
(3)   <myNS1:Kunde>
(4)      <myNS1:KundenNr>4711</myNS1:KundenNr>
(5)      <myNS1:Name>Max Mustermann</myNS1:Name>
(6)      <myNS1:Anschrift>
(7)         <myNS1:Straße>Musterplatz 1</myNS1:Straße>
(8)         <myNS1:PLZ>12345</myNS1:PLZ>
(9)         <myNS1:Ort>Musterstadt</myNS1:Ort>
(10)      </myNS1:Anschrift>
(11)   </myNS1:Kunde>
(12)   <myNS1:Rechnungsposten>
(13)    <!--...-->
(14)    </myNS1:Rechnungsposten>
(15)</myNS1:Rechnung>
```

Download des Beispiels

Hinweis: Für das Attribut `xmlns` kann keine Namensraumdeklaration angegeben werden; es ist spezifikationsgemäß an keinen Namensraum gebunden.

Die Deklaration des Namensraumes mit der Präfixbindung kann auf beliebige hierarchisch höhergeordnete Elemente ausgelagert werden. In der Praxis hat es sich aus Übersichtlichkeitsgründen durchgesetzt, alle in einem XML-Dokument benutzten Namensräume mit ihren Präfixen zu Beginn des Dokuments im Wurzelelement zu definieren.
Das nachfolgende Beispiel zeigt dies anhand eines XHTML-Dokuments, das neben Elementen der Hypertextsprache auch mathematische Formeln und Vektorgraphiken enthält.

Beispiel 11: Ein XHTML-Dokument mit MathML- und SVG-Inhalten

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml"
(3)            xmlns:mml="http://www.w3.org/TR/REC-MathML"
(4)            xmlns:svg="http://www.w3.org/2000/svg">
(5)     <xhtml:head>
(6)             <xhtml:title>XHTML Dokument, mit MathML- und SVG-Inhalten</xhtml:title>
(7)     </xhtml:head>
(8)     <xhtml:body>
(9)             <xhtml:h1>Eine Û¢erschrift</xhtml:h1>
(10)            <mml:math>
(11)            <mml:mrow>
(12)                    <mml:mi>x</mml:mi>
(13)                    <mml:mo>=</mml:mo>
(14)                    <mml:mfrac>
(15)                            <mml:mrow>
(16)                                    <mml:mrow>
(17)                                            <mml:mo>-</mml:mo>
(18)                                            <mml:mi>b</mml:mi>
(19)                                    </mml:mrow>
(20)                                    <mml:mo>&PlusMinus;</mml:mo>
(21)                                    <mml:msqrt>
(22)                                            <mml:mrow>
(23)                                                    <mml:msup>
(24)                                                            <mml:mi>b</mml:mi>
(25)                                                            <mml:mn>2</mml:mn>
(26)                                                    </mml:msup>
(27)                                                    <mml:mo>-</mml:mo>
(28)                                                    <mml:mrow>
(29)                                                            <mml:mn>4</mml:mn>
(30)                                                            <mml:mo>&InvisibleTimes;</
mml:mo>
(31)                                                            <mml:mi>a</mml:mi>
(32)                                                            <mml:mo>&InvisibleTimes;</
mml:mo>
(33)                                                            <mml:mi>c</mml:mi>
(34)                                                    </mml:mrow>
(35)                                            </mml:mrow>
(36)                                    </mml:msqrt>
(37)                            </mml:mrow>
(38)                            <mml:mrow>
(39)                                    <mml:mn>2</mml:mn>
(40)                                    <mml:mo>&InvisibleTimes;</mml:mo>
(41)                                    <mml:mi>a</mml:mi>
(42)                            </mml:mrow>
(43)                    </mml:mfrac>
```

```
(44)            </mml:mrow>
(45)            </mml:math>
(46)            <svg:svg width="4cm" height="8cm">
(47)                    <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm"/>
(48)            </svg:svg>
(49)    </xhtml:body>
(50)</xhtml:html>
```

Download des Beispiels

Definition 8: Namensraumidentifikation
Jeder XML-Namensraum wird durch eine gültige URI identifziert. Diese URI dient ausschließlich der Benennung, daher muß sie nicht auf eine gültige Ressource verweisen.

Überschreiben des Vorgabe-Namensraums:
Aus den Beispielen ist leicht ersichtlich, daß die explizite Angabe des definierten Präfixes für jedes Element eines Namensraumes platzraubend und für die Zuordnung aller Elemente eines Teilbaumes zum selben Namensraum redundant und -- wegen des zusätzlichen Spezifikationsaufwandes -- unpraktikabel ist. Die mehrmalige explizite redundante (identische) Angabe des identifizierenden Präfixes bildet zusätzlich noch eine potentielle Fehlerquelle hinsichtlich Übertragungsfehlern und reiner Tippfehler bei manuell erstellten XML-Dokumenten.

Eine einfache Kompaktifizierungsvariante greift auf die aus den Programmiersprachen geläufigen Regeln für Namensräume zurück. Dort beinhaltet ein explizit geöffneter Block alle enthaltenen Elemente bis zum Blockendesymbol und faßt sie so zu einem Gültigkeitsbereich zusammen.
Dieses Prinzip läßt sich leicht auch auf XML-Dokumente, die immer eine streng hierarchische Baumstruktur aufweisen, anwenden.

Hierzu wird das `xmlns`-Attribut leicht modifiziert eingesetzt. Wird es *ohne nachfolgendes Präfix* und unter Weglassung des separierenden Doppelpunktes verwendet, so definiert es einen *Vorgabenamensraum* (*default namespace*). Dieser umfaßt neben dem Element, welches das Attribut beinhaltet, auch alle Kindelemente. Eine Ausnahme hiervon bilden untergeordnete Elemente, die explizit durch Präfix oder Redefinition des Vorgabenamensraumes einem anderen Namespace zugeordnet werden.

Das nachfolgende Beispiel zeigt dies für das bereits mit Namenräumen versehene Rechnungsdokument

Die syntaktische Definitionsform der Namensraumüberschreibung als XML-(Pseudo-)Attribut stellt hierbei sicher, daß für ein Element keine mehrmalige Überschreibung des Vorgabenamensraumes vorgenommen werden kann, da in diesem Falle das Attribut `xmlns` mehrfach im selben Elementkontext auftreten müßte, was der XML-Basisspezifikation widerspräche.

Beispiel 12: Rechnungsdokument mit überschriebenem Vorgabenamensraum

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<Rechnung xmlns="http://www.xyz.com/sales">
(3)   <Kunde>
(4)      <KundenNr>4711</KundenNr>
(5)      <name>Max Mustermann</Name>
(6)      <Anschrift>
(7)         <Straße>Musterplatz 1</Straße>
(8)         <PLZ>12345</PLZ>
(9)         <Ort>Musterstadt</Ort>
(10)     </Anschrift>
(11)  </Kunde>
(12)  <Rechnungsposten>
(13)           <!--...-->
(14)   </Rechnungsposten>
(15)</Rechnung>
```

Download des Beispiels

Durch die Definition des Vorgabenamensraumes für das Element `rechnung` und all dessen Kindelemente wird derselbe Effekt erreicht wie durch die Präfixangabe im vorangegangenen Beispiel.
Diese Schreibweise stellt lediglich eine Abkürzung der expliziten Qualifizierung jedes einzelnen XML-Namens dar. Insbesondere führt die mehrmalige Redefinition des Vorgabenamensraumes *nicht* zu kaskadierten Namensräumen. Jeder Namensraum ist von allen umgebenden unabhängig definiert.
So kann das Dokument des XHTML-Beispiels auch dahingehend verändert werden, daß die Namensräume erst an der Stelle im Dokument deklariert werden, an der sie auch benötigt werden.

Beispiel 13: Ein XHTML-Dokument mit MathML- und SVG-Inhalten, unter Verwendung überschriebener Vorgabenamensräume

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<html xmlns="http://www.w3.org/1999/xhtml">
(3)     <head>
(4)             <title>XHTML Dokument, mit MathML- und SVG-Inhalten</title>
(5)     </head>
(6)     <body>
(7)             <h1>Eine Überschrift</h1>
(8)             <math xmlns="http://www.w3.org/1998/Math/MathML">
(9)             <mrow>
(10)                    <mi>x</mi>
(11)                    <mo>=</mo>
(12)                    <mfrac>
(13)                            <mrow>
(14)                                    <mrow>
(15)                                            <mo>-</mo>
(16)                                            <mi>b</mi>
(17)                                    </mrow>
(18)                                    <mo>+-</mo>
(19)                                    <msqrt>
(20)                                            <mrow>
(21)                                                    <msup>
(22)                                                            <mi>b</mi>
(23)                                                            <mn>2</mn>
(24)                                                    </msup>
(25)                                                    <mo>-</mo>
(26)                                                    <mrow>
(27)                                                            <mn>4</mn>
(28)                                                            <mo> </mo>
(29)                                                            <mi>a</mi>
(30)                                                            <mo> </mo>
(31)                                                            <mi>c</mi>
(32)                                                    </mrow>
(33)                                            </mrow>
(34)                                    </msqrt>
(35)                            </mrow>
(36)                            <mrow>
(37)                                    <mn>2</mn>
(38)                                    <mo> </mo>
(39)                                    <mi>a</mi>
(40)                            </mrow>
(41)                    </mfrac>
(42)            </mrow>
(43)            </math>
(44)            <svg xmlns="http://www.w3.org/2000/svg" xmlns:svg="http://www.w3.org/2000/
svg" svg:width="4cm" svg:height="8cm">
(45)                    <ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm"/>
(46)            </svg>
(47)    </body>
(48)</html>
```

Download des Beispiels

Die Namensraumpräfixe können durch den Anwender frei vergeben werden. Sie dienen lediglich der abkürzenden Schreibweise und sind für die Namensraumauflösung unerheblich.
Daher werden zwei Elemente oder Attribute als gleich betrachtet, wenn sie lexikalisch in Namen und Namensraumidentifier übereinstimmen. Hierbei ist es unerheblich, ob der Namensraum explizit durch Präfixangabe oder durch Überschreiben des Vorgabenamensraumes definiert wurde.
Die Elemente der XML-Dokumente aus den Beispielen 14 und 15 befinden sich alle ausnahmslos im Namensraum `http://www.example.com`.

Beispiel 14: Namensraumpräfixe 1

```
(1)<abc:ElementA xmlns:abc="http://www.example.com"
(2)                             xmlns:xyz="http://www.example.com">
(3)     <ElementB xmlns="http://www.example.com">
(4)             <ElementC/>
(5)     </ElementB>
(6)     <xyz:ElementB>
(7)             <abc:ElementC/>
(8)     </xyz:ElementB>
(9)</abc:ElementA>
```

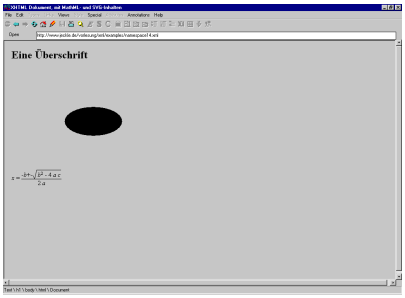
Download des Beispiels

Beispiel 15: Namensraumpräfixe 2

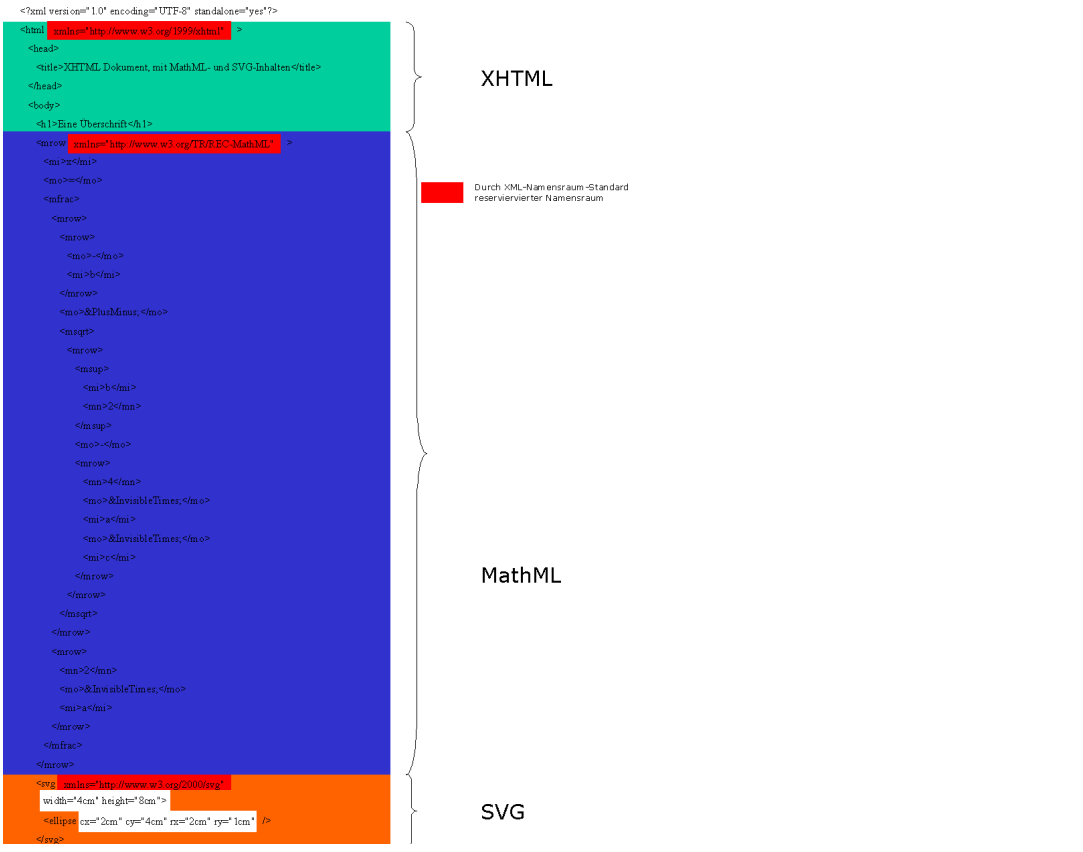
```
(1)<ElementA xmlns="http://www.example.com"
(2)             xmlns:myNamespace="http://www.example.com">
(3)   <foo:ElementB xmlns:foo="http://www.example.com">
(4)      <myNamespace:ElementC/>
(5)   </foo:ElementB>
(6)   <ElementB xmlns="http://www.example.com">
(7)      <myNamespace:ElementC/>
(8)   </ElementB>
(9)</ElementA>
```

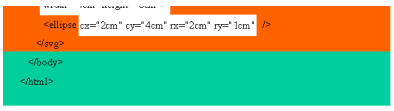
Download des Beispiels

Die Abbildung zeigt das Beispieldokument in der Darstellung des W3C-Browsers Amaya.



Im Beispieldokument wird der Vorgabenamensraum dreimal, entsprechend der verschiedenen verwendeten XML-Sprachen, neu gesetzt. So wird auf `html` und alle direkt untergeordneten Elemente der URI-identifizierte Namensraum `http://www.w3.org/1999/xhtml` angewendet. head, `title` und body sowie dessen Kindelemente finden sich demnach, da sie keinen eigenen Namensraum definieren, ebenfalls im so definierten Vorgabenamensraum.
`mrow` als hierarchisch tieferstehendes Element redefiniert den Namensraum zu `http://www.w3.org/TR/REC-MathML`. Daher werden das Element `mrow` sowie all dessen Kindelemente (im Beispiel: `ellipse`) auch diesem zugeordnet.
Die Attribute `width`, `height`, `cx` , ... verfügen über kein explizites Namensraumpräfix und sind daher dem leeren Namensraum zugeordnet.
Auf den MathML-Namensraum folgend wird der Vorgabenamensraum zu `http://www.w3.org/2000/svg` redefiniert. Auch hier gelten dieselben Regeln, d.h. der überschriebene Vorgabenamensraum erstreckt sich auf alle Kindelemente.
Mit dem schließenden Tag svg endet auch dessen Namensraum. Alle folgenden Elemente befinden sich wieder im umgebenden Namensraum, der zu Beginn des Dokuments mit `http://www.w3.org/1999/xhtml` festgelegt wurde.
Die nachfolgende Graphik stellt die Namensräume nochmals farblich hervorgehoben dar.
Ein weiteres Beispiel findet sich in der Namespace-Recommendation.





Der XML-Namensraumstandard des W3C sieht die beiden im Vorhergehenden diskutierten Varianten exklusiv zueinander vor. D.h. für ein Element, welchem bereits durch Präfixangabe eine Namensraumzuordnung gegeben wurde, kann nicht zusätzlich der Vorgabenamensraum überschrieben werden. Deklarationen der Form `<xyz:abc xmlns="..." ...>` sind widersprüchlich; und daher illegal. (in der XML-Namespace Recommendation nachschlagen)

Das abschließende Beispiel 16 zeigt die Verwendung zweier Vokabulare (SVG und MathML), die beide ein mit `set` benanntes Element definieren.
Durch die Deklaration der jeweiligen Namensräume unterscheiden sich die qualifizierten Namen, die dem (gleichnamigen) Elementnamen die Namensraum-URI voranstellen.

Beispiel 16: Namensräume im realen Einsatz

```
(1)<?xml version="1.0"?>
(2)<document>
(3)     <svg xmlns="http://www.w3.org/2000/svg">
(4)             <g transform="translate(100,100)">
(5)                     <text id="TextElement" x="0" y="0" style="font-family:Verdana;
font-size:35.27; visibility:hidden">
(6)                     It's alive!
(7)                             <set attributeName="visibility" attributeType="CSS"
to="visible" begin="3s" dur="6s" fill="freeze"/>
(8)                     </text>
(9)             </g>
(10)    </svg>
(11)
(12)    <math xmlns="http://www.w3.org/1998/Math/MathML">
(13)            <set>
(14)                    <ci> b </ci>
(15)                    <ci> a </ci>
(16)                    <ci> c </ci>
(17)            </set>
(18)    </math>
(19)</document>
```

Download des Beispiels

Präzedenz des explizit zugeordneten Namensraumes:
Eine explizit durch Präfixzuordnung vorgenommene Namensraumfestlegung besitzt Präzedenz gegenüber dem evtl. überschriebenen Vorgabenamensraum.
Findet daher für ein Element sowohl die Überschreibung des Vorgabenamensraumes, als auch gleichzeitig die Namensraumfestlegung durch explizite Präfixzuordnung statt, so wird das Element demjenigen Namensraum zugeordnet, der durch die URI identifiziert wird, an den das Präfix gebunden ist.
Dies gilt insbesondere auch dann, wenn ein und dasselbe Element sowohl über ein Präfix, als auch eine Überschreibung des Vorgabenamensraumes verfügen.
Das XML-Dokument aus 17 illustriert dies beispielhaft. So wird `ElementA` -- durch Überschreibung des Vorgabenamensraumes -- dem Namensraum `urn:namspaces:Namespace1` zugeordnet und diese Festlegung auch an das Kindelement `ElementB` weitergegeben.
Das Kindelement `ElementC` hingegen überschreibt die Vorgabe des Elternelements durch explizite Präfixangabe und ist daher dem durch `urn:namespace:Namespace2` identifizierten Namensraum zugeordnet.
Für `ElementD` findet sich sowohl eine Namensraumdefinition, welche durch Überschreiben des Vorgabenamensraumes zu `urn:namespace:Namespace3` stattfindet, als auch eine Präfix-gebundene Definition an den Namensraum `urn:namespace:Namespace2`. Gemäß der Präzedenz der expliziten Festlegung durch Präfix wird `ElementD` jedoch ausschließlich dem Namensraum zugeordnet, an den das angegebene Präfix `ns1` gebunden ist. Im Beispiel ist dies die URI `urn:namespace:Namespace2`.

Beispiel 17: Präzedenzregel

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<ElementA xmlns="urn:namspaces:Namespace1"
(3)                     xmlns:ns1="urn:namespace:Namespace2"
(4)                     xmlns:ns2="urn:namespace:Namespace3">
(5)     <ElementB/>
(6)     <ns1:ElementC/>
(7)     <ns1:ElementD xmlns="urn:namespace:Namespace3"/>
(8)</ElementA>
```

Download des Beispiels

Aufheben der Namensraumzuweisung:
Durch Überschreibung des Vorgabenamensraumes mit der Zeichenkette leeren Inhalts -- formal der Zuweisung der leeren URI als Namensraumidentifikator -- kann eine bestehende Namensraumdefinition aufgehoben werden. Als Resultat entsteht eine Situation identisch zu einem Dokument ohne festgelegte Namensräume, d.h. die Elemente finden sich im leeren Namensraum.

Beispiel 18: Aufheben von Namensraumdeklarationen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<Adressen>
(3)     <table xmlns="http://www.w3.org/TR/REC-html40">
(4)             <tr>
(5)                     <td>Name</td>
(6)                     <td>Adresse</td>
(7)             </tr>
(8)             <tr>
(9)                     <td>
(10)                            <Vorname xmlns="">Max</Vorname>
(11)                       <Nachname xmlns="">Mustermann</Vorname>
(12)                    </td>
(13)                    <td>
(14)                            <Straße xmlns="">Musterstr. 1</Straße>
(15)                            <PLZ xmlns="">12345</PLZ>
(16)                            <Ort xmlns="">Musterstadt</Ort>
(17)                    </td>
(18)            </tr>
(19)    </table>
(20)</Adressen>
```

Das Beispiel 18 zeigt die notwendigen Deklarationen zur Aufhebung der Vorgabenamensraumdefinition. So wird zwar für das Element `table` und alle seine Kindelemente der Vorgabenamensraum auf `http://www.w3.org/TR/REC-html40` gesetzt, dies jedoch für die Kindelemente `Vorname`, `Nachname`, `Straße`, `PLZ` und `Ort` durch die Festlegung `xmlns=""` explizit für das jeweilige Element aufgehoben.

Die Aufhebung von definierten Namensräumen kann ausschließlich durch die Überschreibung des Vorgabenamensraum erfolgen. Eine Bindung der leeren URI an ein Präfix zur späteren Verwendung ist nicht zugelassen.

Namensräume für Attribute:
Abweichend von der Mimik für Elemente, dort wirkt sich ein überschriebener Vorgabenamensraum auch immer auf die Kindelemente aus, wird eine Namensraumdeklaration auf Elementebene nicht auf Attribute propagiert.
Diese Festlegung der Spezifikation mag insbesondere unter Kenntnis der Baumstruktur der Infosets, welche Attribute und Elemente gleichermaßen als Kindknoten der beherbergenden Elementinformationseinheit darstellt, verwundern. Eine mögliche Begründung dieser Asymmetrie mag in der besonderen Rolle der Attribute zur Informationsdarstellung liegen. So wird teilweise damit argumentiert, daß Attribute üblicherweise unabhängig vom aktuell umgebenden Element sein sollten und daher nur zur Darstellung von Daten herangezogen werden sollten, die nicht über einen direkten Bezug zum sie umgebenden Element verfügen.
In der Konsequenz müssen Attribute immer explizit mit einem Namensraumpräfix versehen werden, um sie einem Namensraum zuzuordnen.
Beispiel 19 zeigt die Anwendung der Namensräume auf Attribute. So befinden sich weder das Attribute `att1` des Elements `ElementB`, noch dasjenige von `ElementD` in einem Namensraum. Das mit dem Wert `XYZ` versehene Attribut `att2` des Elements `ElementC` wird hingegen -- aufgrund des explizit angegebenen Präfixes -- dem Namensraum `http://www.example.com/NS2` zugeordnet.
Ferner illustriert `ElementC` die Rolle der Namensräume als Bestandteil des identifzierenden Namens von Elementen und Attributen. Aufgrund der Interpretation des Namensraumes als Benennungsbestandteil darf das `att2` benannte Attribut mehrfach auftreten, da die Zuhilfenahme des Namensraumes die eindeutige Identifikation gestattet.

Beispiel 19: Namensräume für Attribute

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<Wurzelelement>
(3)     <ElementA xmlns:NS1="http://www.example.com/NS1" xmlns:NS2="http://www.example.com/
NS2">
(4)             <ns2:ElementB att1="...">
(5)                     <ElementD att1="..." xmlns="http://www.example.com/NS3">
(6)                             <ElementC att2="ABC" NS2:att2="XYZ"/>
(7)                     </ElementD>
(8)             </ns2:ElementB>
(9)     </ElementA>
(10)</Wurzelelement>
```

Definition 9: Namensraumvererbung

Namensräume, die durch Überschreiben des Vorgabenamensraumes zugewiesen werden wirken sich ausschließlich auf Elemente und deren direkte oder transitive Kindelemente aus, sofern diese den Namensraum nicht wieder verändern.
Namensräume, die durch explizite Präfixangabe zugewiesen werden, wirken sich ausschließlich auf dasjenige Element aus vor dessen Name das Präfix plaziert ist.
Namensräume für Attribute werden ausnahmslos durch explizite Präfixangabe festgelegt und gelten ausschließlich für das Attribut selbst.

Ausgehend von der Vererbungsregel für Namensräume, sowie der Präzedenz expliziter Präfixangaben lassen sich daher folgende Auswertungsregeln definieren:

Ein Element befindet sich in demjenigem Namensraum ...

1. ... an den das vorangestellte Präfix gebunden ist.
   Verfügt das Element über kein Namensraumpräfix, so befindet es sich in demjenigen Namensraum ...
2. ... der auf diesem Element durch Überschreibung des Vorgabenamensraumes definiert wurde.
   Findet für dieses Element keine Überschreibung des Vorgabenamensraumes statt, so befindet es sich in demjenigen Namensraum ...
3. ... der für das Elternelement gilt, sofern er dort Vorgabenamensraum ist.
   Man beachte: Das *gilt* im vorangehenden Satz umschließt sich nicht nur die Überschreibung des Vorgabenamensraumes im direkten Elternelement, sondern auch eine dort geltende Namensraumüberschreibung die in dessen Elternelement oder dessen Elternelement ... stattfand.
   Findet in keinem der Elternelemente eine Überschreibung des Vorgabenamensraumes statt, so befindet sich das Element in demjenigen Namensraum ...
4. ... der leer ist (d.h. im leeren Namensraum).

Ein Attribut befindet sich in demjenigem Namensraum, der durch explizite Präfixangabe festelegt wurde.

Internationale URIs und Namensraumidentifikatoren:
Die Berücksichtigung von Zeichen, die in XML v1.1 zugelassenen, deren Nutzung in den klassischen URIs nach RFC 2396 bzw. RFC 2732 jedoch untersagt ist, führt zur Einführung des neuen Begriffes des *Internationalized Resource Identifier*s (IRI). Diese Neuschöpfung stellt im Kern eine URI-Fassung dar innerhalb der Leerzeichen sowie diverse Sonderzeichen zulassen sind. Diese internationalisierten Identifikatoren werden durch einen im Spezifikationsentwurf festgelegten Algorithmus in syntaktisch korrekte URIs umgewandelt.
Beispiel 20 zeigt gültige IRIs und jeweils dahinter in Klammern angegeben die daraus resultierende URI-Darstellung.

Beispiel 20:

```
(1)http://www.{iri-}example.com (http://www.%7Biri-%7Dexample.com)
(2)mailto:marc léon@example.org (mailto:marc%20l%E9on@example.org)
```

Kompatibilität zu älteren Dokumenten:
Elemente, für die weder ein expliziter Namensraum durch Präfix definiert ist, noch ein Namensraum von einem Elternelement übernommen werden kann, sind einem leeren Namensraum zugeordnet; konzeptionell entspricht dies einem NULL-Präfix.
Somit befinden sich alle Elemente, die keinem Namensraum angehören, automatisch in einem gemeinsamen Namensraum, der an keine URI gebunden ist.

Zusammenfassend gelten somit folgende Prinzipien:

- Jede beliebige URI kann an eigendefinierte Präfixe gebunden werden.
  Insbesondere kann dieselbe URI an verschiedene Präfixe gebunden werden, und dasselbe Präfix in verschiedenen Kontexten für verschiedene URIs stehen.
- Jedes Element übernimmt den überschriebenen Vorgabenamensraum seines Elternelements, sofern es keinen eigenen definiert.
- Elemente oder Attribute ohne Namensraumzuordnung (die auch keine von ihrem hierarchisch höherstehenden Element übernehmen) befinden sich im Standardnamensraum.
- Attribute ohne Namensraumprefix befinden sich im leeren Namensraum.

Web-Referenzen 6: Weiterführende Links

- XML-Namespace Recommendation
- Namespace Recommendation in deutscher Übersetzung
- Namespace Tutorial @ Zvon.org
- Tim Bray: Namespaces by Example
- Hintergrundartikel: Namespaces in XML Adopted by W3C
- (Tutorial) Simon St. Laurent: Namespaces in XML
- Roland Bourret: XML Namespaces FAQ

2.3 XML-Schema

XML Schema

Neben den in der Vergangenheit zur Sprachdefinition verwendeten Document Type Definitions ist in jüngerer Zeit ein alternativer Ansatz in den Blickpunkt des Interesses gerückt: die XML-Schemasprachen. Sie setzen die Emanzipation der Metasprache XML von ihrer Vorgängersprache SGML fort. Bereits in engem zeitlichem Bezug zur Veröffentlichung der XML-Recommendation wurde mit *XML Data* ein erster Ansatz vorgestellt. In der Zwischenzeit fanden verschiedene konkurrierende Vorschläge ein breites Interesse. Übereinstimmende Zielsetzung aller verschiedenen vorgeschlagenen Schemasprachen ist die Schaffung eines Sprachdefinitionsmechanismus, der die Dokumenten-orientierten Strukturen und Inhaltsmodelle der DTD überwindet.
An die Spitze der Bemühungen setzte sich eine Arbeitsgruppe des W3C zur Definition einer XML-Schemasprache, unter Berücksichtigung der bekanntesten und verbreitetsten Vorschläge. Durch sie wurde im Mai 2001 der *XML Schema*-Standard des W3C veröffentlicht.

Der Begriff *Schema* ist der im Datenbankumfeld gebräuchlichen Terminologie entlehnt. Dort bezeichnet er Informations- oder Datenmodelle als Konstruktionsvorlage oder Dokumentation eines Datenbankdesigns. Hierzu muß ein Schema nicht unbedingt in einer graphischen Datenmodellierungssprache vorliegen, sondern kann beispielsweise auch die Tabellenstruktur einer relationalen Datenbank bezeichnen.

Zur Notwendigkeit einer Schemasprache:
Zum Zeitpunkt der Konzeption der Metasprache SGML war das Anwendungsfeld klar umrissen und im wesentlichen auf die Digitalisierung vormals papiergestützter Dokumentation festgelegt. Daraus erklärt sich auch die Mächtigkeit der Document Type Definition, der angebotenen Grammatiksprache zur Darstellung der Dokumentstrukturen.
Insbesondere war weder die Daten-orientierte Verwendung von SGML, noch die rund 30 Jahre später einsetzende Weiterentwicklung (eigentlich: Reduktion) zur eXtensible Markup Language abzusehen.
Die inzwischen eingesetzte breite Anwendung von XML-Sprachen zur Darstellung beliebiger Inhalte läßt jedoch die Beschränkungen und Unzulänglichkeiten des DTD-Mechanismus für diesen Anwendungen offenkundig werden.

Nachfolgend sind einige der durch Nutzung des DTD-Mechanismus zur Beschreibung Daten-intensiver Strukturen induzierten Einschränkungen zusammengestellt:

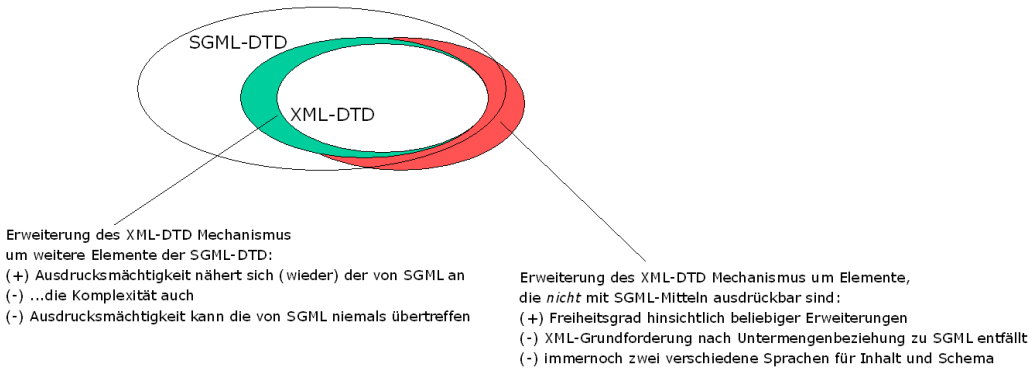
- Unzureichende Datentypunterstützung.
  DTDs erlauben für Elemente nur die vier Inhaltsmodelle: `child elements`, `PCDATA`, *mixed content* sowie das leere Inhaltsmodell `EMPTY`.
- Unzureichende Strukturierungsunterstützung.
  Zwar erlauben die Operatoren zur Steuerung der Auftrittshäufigkeit („?“, „+“ und „*“) einzelner Kindelemente die Codierung beliebiger Kardinalitäten. Allerdings sind die hierfür anzuwendenden Prinzipien teilweise umständlich in der Schreibung, und daher fehlerträchtig. Die Lesbarkeit der entstehenden DTD wird entscheidend gesenkt.
- Keine Unterstützung von Wiederverwendbarkeit.
  Während Elementstrukturen (zumindest) innerhalb der definierenden DTD beliebig wiederverwendet werden können, sind Attribute immer an das umgebende Element gebunden .
  Eine Nutzung in anderen Dokument-Typ-Definitionen als der definierenden ist nicht vorgesehen.
- Starres Typsystem.
  Das angebotenen Typsystem kann durch den Anwender nicht erweitert werden.
- Keine Unterstützung von Namensräumen.
  Namensräume können in der DTD nicht angegeben werden.
- Nur rudimentärer Referenzierungsmechanismus.
  Die offerierten Verknüpfungsmechanismen sind ausschließlich Dokument-lokal möglich und gestatten keine Differenzierung hinsichtlich der Semantik des eindeutig identifizierten oder referenzierten Elements.
- DTD-Syntax ist nicht XML.
  Die von SGML übernommene Syntax der DTD bildet eine eigene Sprache, die von Werkzeugen gesondert zu implementieren ist.

Technische Ansätze:
Prinzipiell lassen sich die in der Vergangenheit vorgeschlagenen Ansätze zur Definition einer Schemasprache in vier Kategorien unterscheiden:

1. Orientierung am bestehenden DTD-Mechanismus.
   Erweiterungen des bestehenden Mechanismus um zusätzliche Sprachelemente.
2. Orientierung an der programmiersprachlichen Interpretation.
   Versuch XML und ein Ausführungsmodell möglichst eng zu koppeln.
3. Orientierung an Wissensdarstellungen
   Interpretation des Schemas einer XML-Sprache als Wissen über die Sprache.
4. XML-Sprachen zur Inhaltsbeschreibung.
   Da XML i.A. zur Beschreibung beliebigster Informationen herangezogen werden kann, ist die Verwendung auch für die Beschreibung von XML-Strukturen denkbar.

Die naheliegendste Option dürfte die Erweiterung des bestehenden DTD-Sprachumfanges bilden. Durch geeignete Modifikationen und Ergänzungen ließen sich alle, mit Ausnahme der letzten, identifizierten Unzulänglichkeiten beheben.
Konzeptionell lassen sich zwei Erweiterungsvarianten aufzeigen. Zunächst die Möglichkeit, die XML-DTDs um Elemente der ursprünglichen SGML-DTD zu erweitern. In der Konsequenz nähert sich XML, positiv formuliert, wieder der Ausdrucksmächtigkeit der Ursprache SGML an. Negativ formuliert, kann jedoch XML auf diesem Wege niemals Inhaltsstrukturen ausdrücken, die nicht durch SGML ausdrückbar sind, da die Mächtigkeit des SGML-DTD-Mechanismus eine natürliche Obergrenze der Erweiterbarkeit darstellt.
Zusätzlich ist anzumerken, daß ein solcher Ansatz der ursprünglichen Intention der XML-Entwicklung -- ein leichter einsetzbares SGML zu schaffen -- entgegenläuft.
Eine der bekannten Ideen zur Erweiterung des DTD-Mechanismus stellt *Datatypes for DTDs* (DT4DTD) dar.
Alternativ zur Erweiterung hin zur SGML-Mächtigkeit ließe sich der bestehende XML-DTD-Mechanismus um neue zusätzliche Konstrukte anreichern, die nicht Bestandteil der SGML-DTD-Syntax sind. Dieser Ansatz böte den Vorteil, den Vorgängerstandard nicht berücksichtigen zu müssen und beliebige Erweiterungen in Syntax und Semantik einbringen zu können. Allerdings würde damit eine zentrale Forderung der XML-Entwicklung, die sich bereits im Abstract der XML-Recommendation findet, nicht berücksichtigt: die Untermengenbeziehung zu SGML. Durch eine Erweiterung, welche über die SGML-Mächtigkeit hinausreicht, würden legale (*well formed* und sogar *valid*) XML-Dokumente entstehen, die keine gültigen SGML-Dokumentinstanzen wären.

Die nachfolgende Graphik veranschaulicht die beiden Erweiterungsoptionen und die Argumente der geführten Diskussion.



Die im zweiten Punkt angedeutete Umsetzung ist durch eine programmiersprachliche Verarbeitung der XML-Dokumente motiviert. Aus Sicht dieser Anwendungsfacette ist ein Schemamechanismus idealerweise so ausgelegt, daß er die transparente Umsetzung in Applikationsdatenstrukturen ermöglicht. Dahinter steht der Wunsch, den *impedance mismatch*, mithin den zu leistenden Abbildungsaufwand zwischen XML-Konstrukten und Datenstrukturen, möglichst gering zu halten.
Beispielsweise greift der -- durch den Einsatz im e-Commerce-System der Firma CommerceOne bekannt gewordene -- Vorschlag *Schema for Object-Oriented XML* (SOX) zur Definition der notwendigen Semantik der angebotenen Schemaprimitiven auf die bekannte plattformunabhängige Programmiersprache Java zurück.
Die aktuelle Version der Schemasprache SOX, die zur Definition der XML-Sprache xCBL eingesetzt wird, findet sich unter xCBL.org.

Der dritte technische Ansatz weist auf eine alternative Interpretation der XML-Grammatikstruktur hin. So spiegelt ein Schema auch immer *Wissen* über Struktur und Inhalt eines betrachteten Problembereichs wieder.
Der bekannteste Vorschlag -- die *Document Content Description* (DCD) -- nutzt zur Definition der Wissensstrukturen eines XML-Dokuments das *Resource Description Framework* (RDF) des World Wide Web Consortiums.
Der Ansatz hat sich durch Referenzimplementierungen durchaus als tragfähig und, wegen der RDF-basiertheit, als allgemein verwendbar erwiesen. Jedoch liegt hierin auch die offensichtlichste Limitierung. RDF als Metasprache der Schemasprache legt bereits eine gewisse Strukturierung aller Schemata zugrunde, da jedes gültige DCD-Schema definitionsgemäß ein RDF-Dokument darstellt. Ebenso ist die Semantik der eingesetzten RDF-Elemente bereits durch diese Spezifikation vorgegeben. Beide Punkte zusammengenommen offenbaren eine ausgeprägte Abhängigkeit von den weiteren RDF-Aktivitäten des World Wide Web Consortiums, die bisher nicht auf die Interdependenz von Schemasprache und Wissensbeschreibungsformat ausgerichtet ist.
Positiv fällt an DCD die Verwendung von XML zur Beschreibung von XML-Sprachen auf, womit auch die letzte der erhobenen Anforderungen zu erfüllen wäre.
Die Verknüpfung von RDF mit DCD als Schemasprache birgt allerdings ein potentielles Problem hinsichtlich der Validierbarkeit der entstehenden Strukturen. Durch den Rückgriff von DCD auf RDF entsteht bei der Angabe eines Schemas für RDF ein transitiver Zirkelschluß. In der Konsequenz wird zur Validierung eines XML-Dokuments, welches einer mittels DCD-formulierten Grammatik folgt, neben dem eigentlichen DCD-Schema des Dokuments auch das DCD-Metaschema und dessen Semantik-liefernde RDF-Beschreibung benötigt.

Diese Beschränkung mildert die vierte Familie von XML-Schemasprachen ab. Sie umfaßt die meisten

Vorschläge, die alle als eigenständige XML-Sprachen ausgelegt sind; daher definieren sie ein eigenständiges XML-Vokabular zur Darstellung der benötigten XML-Strukturen, sowie die zugehörige Semantik.
In der Folge sind sie für die Meta-Schemaebene selbstbeschreibend. Das bedeutet das Schema eines Schemas kann durch sich selbst validiert werden. Da dieser Validierungsschritt statisch nur einmal erfolgen muß, kann er durch Schemawerkzeuge vorweggenommen werden.
In dieser Kategorie sind die meisten der bisher vorgeschlagenen Schemadialekte einzuordnen.

Die größte Bedeutung haben kontextfreie reguläre Sprachen zur Spezifikation von XML-Sprachstrukturen erlangt.
Eine Sprache dieses Typs entwickelt auch die W3C-Arbeitsgruppe zur Definition eines XML-Schemasprachstandards. Insbesondere berücksichtigt diese Aktivität explizit die Vorgängersprachen XML Data, DCD, SOX sowie Document Definition Markup Language. Die erwähnten konkurrierenden Vorschläge unterscheiden sich semantisch lediglich in Nuancen, bieten dem Anwender jedoch teilweise (optisch) stark unterschiedliche Konstrukte zur Syntaxspezifikation an.

Einen strukturell unterschiedlichen Ansatz verfolgt die durch Rick Jelliffe vorgeschlagene Sprache *Schematron*. Sie interpretiert ein Schema als Sammlung von Regeln, denen ein gegebenes Dokument genügen muß, um als gültig akzeptiert zu werden. Dies erlaubt die Formulierung mächtiger konktextsensitiver Einschränkungen, die während des Validierungsvorganges geprüft werden.
Die Umsetzung dieser Schemasprache setzt auf den XML-Standards XPath und XSLT auf.

W3Cs XML-Schema:
Jenseits aller existierenden verschiedenen Sprachvorschläge kommt dem W3C-Standard der *XML Schema Description Language* (XSD) die größte praktische Bedeutung zu.
Tim Berners-Lee verkündete in der Eröffnungsrede der WWW-Konferenz in Hong Kong am 2. Mai 2001 die Verabschiedung als Recommendation. Gleichzeitig deutete er bereits weitere Schema-Aktivitäten des World Wide Web Consortiums an.
XML-Schema bildet zusammen mit XML v1.0 2nd edition und den Namensräumen die Basis aller weiteren W3C-XML-Sprachstandards.

Aus formalen Gründen ist nicht mit dem Ersatz der DTD durch Schema zu rechnen. Jedoch werden mittelfristig neu entwickelte XML-Sprachen keine Grammatiken mehr in der Syntax der DTD entwickeln, sondern direkt Schemata definieren.

XSD bildet eine vollständig in XML-Syntax formulierte kontextfreie reguläre Grammatik zur Formulierung beliebiger XML-Strukturen ab. Hierbei handelt es sich um die bekannten Grundprimitive *Element* und *Attribut*
Gleichzeitig wurde, neben zahlreichen anderen Neuerungen, die Kommentarsyntax für Schemata neu definiert.

Inhaltlich gliedert sich der XSD-Sprachvorschlag in zwei große Teilbereiche: Part 1: Structures zur Definition von Inhaltsmodellen für Elemente, Attributstrukturen und wiederverwendbaren Strukturen und Part 2: Datatypes zur Festlegung diverser inhaltlicher Charakteristika wie Datentypen und konsistenzgarantierende Einschränkungen.
In beiden Teilen werden XML-Namensräume explizit berücksichtigt. Konzeptionell rekonstruiert XSD-Part1 zunächst die bekannte Mächtigkeit der DTD um so die evolutionäre Weiterentwicklung bestehender XML-Sprachen zu ermöglichen.
Der zweite Teil der XSD-Spezifikation definiert ein eigenständiges Typsystem, das neben der naheliegenden Verwendung im ersten Teil der Schemasprache XSD auch in anderen W3C-Arbeitsgruppen Verwendung findet. Inhaltlich baut auch Part2 auf den in der DTD definierten Typen auf und erlaubt zunächst direkt ihre Angabe in Schemata. Darauf aufbauend wird eine Fülle verschiedenster Typen angeboten, die an die verschiedenen verfügbaren Typsysteme aus den Programmiersprachen, Datenbanken und internationalen Standards angelehnt sind.
Alle durch XSD definierten Elemente, d.h. alle Primitive zur Definition eines eigenen Schemas, befinden sich im Namensraum `http://www.w3.org/2001/XMLSchema`, der üblicherweise an das Präfix `xsd` gebunden wird. Elemente und Attribute aus XML-Schema, die in Instanzdokumenten verwendet werden könne sind im Namensraum `http://www.w3.org/2001/XMLSchema-instance` (übliches Präfix `xsi`) organisiert.
Wegen des Umfanges der offiziellen Schemadokumente wird zusätzlich durch das W3C ein Part 0: Primer herausgegeben. Er stellt die beiden XSD-Teile in der Zusammenschau an Beispielen dar.

Schemareferenz:
Jedes XML-Schema bildet als XML-Dokument eine eigenständige Speichereinheit, üblicherweise eine Datei. Die Verbindung zwischen Schema und beschriebenem Dokument wird durch das in der XSD-Spezifikation vordefinierte Attribut `schemaLocation` bzw. `noNamespaceSchemaLocation` definiert. Eines dieser Attribute muß zwingend im Wurzelelement des XML-Dokuments angegeben werden.
Legt das Schema keinen Namensraum für die enthaltenen Deklarationen fest, d.h. alle darin deklarierten Elemente befinden sich im Vorgabenamensraum, so findet sich die Schemareferenz in `noNamespaceSchemaLocation`; andernfalls in `schemaLocation`.
Das nachfolgende Beispiel zeigt die Deklaration:

Beispiel 21: Definition einer Schemareferenz

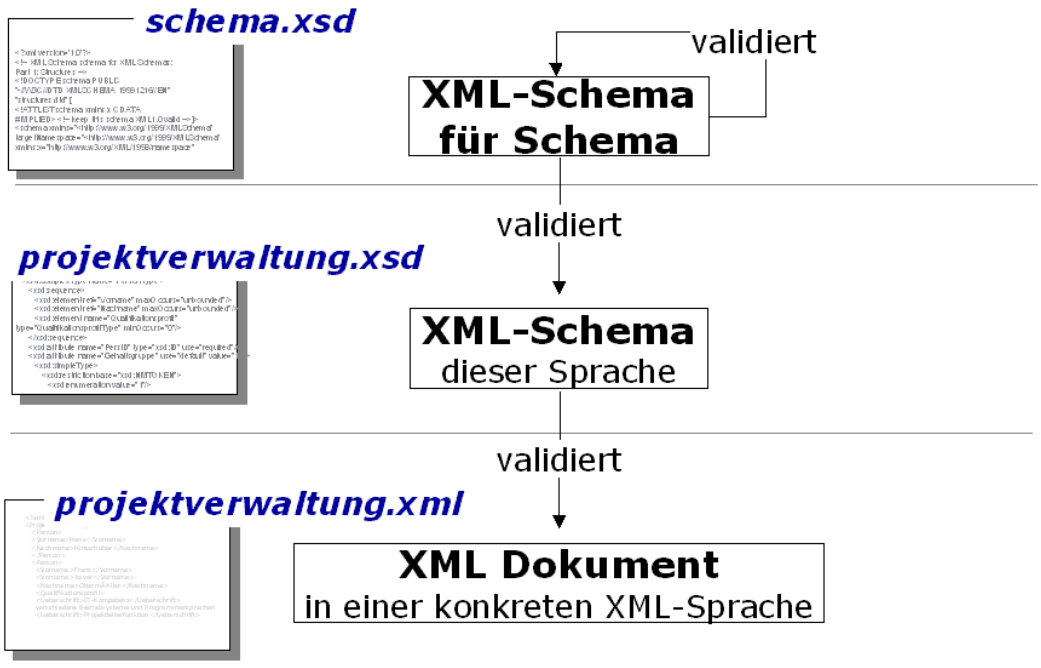
```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<ProjektVerwaltung
(3)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)     xsi:schemaLocation="http://www.jeckle.de/vorlesung/xml/examples/projektverwaltung.
xsd">
(5)     ...
```

Im Beispiel wird zunächst der XML-Schema-Instanzen-Namensraum an das Präfix `xsi` gebunden. Dies ermöglicht die Einbindung von Elementen und Attributen aus der Schemaspezifikation in das eigene Dokument.
Als erste Nutzung eines solchen Elements aus XSD wird das Attribut `schemaLocation` im Wurzelelement mit der URI des Schemas als Wert belegt. Die Deklaration des XSI-Namensraumes ist daher zwingend.
Die angegebene URI kann zur Ermittlung des Schemas für Validierungszwecke durch einen XML-Prozessor genutzt werden.

Aufbauend auf dem Begriff der Wohlgeformtheit definiert XML-Schema den der *Schemagültigkeit* als höhere Qualitätsstufe eines XML-Vokabulars:

Definition 10: Gültigkeit hinsichtlich eines Schemas
Ein XML-Dokument heißt *gültig hinsichtlich eines Schemas* (*schema valid*), wenn es über ein Schema verfügt, und konform zu diesem aufgebaut ist.

Aufgrund der Realisierung der Schemasprache als XML-Sprache ist jedes Schema auch ein XML-Dokument. Daher eröffnet sich die Möglichkeit, das Schema selbst durch ein Schema zu beschreiben. Dieses *Schema für Schema* -- auch *Metaschema* genannte -- XML-Dokument erlaubt die Validierung (im Sinne der *schema validness*) jedes Schemas. Damit erfüllt sich eine der Anforderungen an den Schemamechanismus: die Validierbarkeit der erstellten Schemata selbst, was für DTDs nicht gegeben war. In der praktischen Anwendung zeigt sich dies in der Möglichkeit, erstellte Schemata mit denselben Werkzeugen zu analysieren, verarbeiten und zu prüfen, die auch für Instanzdokumente verwendet werden.
Da das Metaschema selbst wiederum ein XML-Dokument ist, folgt, daß hierfür auch ein Schema angegeben werden kann. Die XML-Standardisierung hat hier -- nicht zuletzt um eine unendliche Reihung zur Validierung notwendiger Schemata zu vermeiden -- den Ansatz gewählt, das Schema für Schema durch sich selbst zu beschreiben.
Die Abbildung stellt die getroffenen Aussagen und Validierungsbeziehungen nochmals graphisch zusammen.



Die Schema-Definition:
Wuzelknoten jedes XSD-Dokuments ist das Element Information Item `schema`. Alle Definitionen eines Schemas sind direkte Kindknoten dieses Elements oder dessen Kindknoten.
Durch die Attribute des `schema`-Elements werden verschiedene Eigenschaften festgelegt, die für alle im Schema definierten Elemente und Attribute gelten.

Zunächst wird durch eine Reihe von Attributen das Verhalten des Schemas in Bezug auf Namensräume festgelegt. Als Besonderheit eines XML-Schemas fällt hier die ständige Berücksichtigung von mindestens zwei Namensräumen ins Auge. Während ein Schema mit Elementen des Schemanamensraumes aufgebaut wird, trifft es zeitgleich Aussagen über einen zweiten Namensraum -- den Namensraum des Vokabulars

für das das Schema erstellt wird. Dieser Namensraum wird *Zielnamensraum* (*target namespace*) genannt. Daher findet sich im Attribut `targetNamespace` die URI des Zielnamensraumes. In diesen Namensraum werden automatisch alle durch das Schema deklarierten Elemente und Attribute übernommen. Als Konsequenz müssen diese in jedem Schema-gültigen XML-Dokument im entsprechenden Namensraum auftreten. Hierbei wird nicht zwischen expliziter Namensraumdeklaration durch ein gebundenes Präfix und impliziter Deklaration durch Überschreiben des Vorgabenamensraumes unterschieden.
Durch Angabe der Attribute `elementFormDefault` und `attributeFormDefault` kann der durch `targetNamespace` implizierte Namensraumzwang für das XML-Instanzdokument gelockert werden. Wird der Wert der beiden Attribute auf `unqualified` gesetzt, so können die Attribute auch außerhalb des Zielnamensraumes auftreten. Dies entspricht auch dem Vorgabeverhalten.

Definition von Elementen:
Als Obermenge der Ausdrucksmächtigkeit der DTD unterstützt auch XSD die Inhaltsmodelle

- unstrukturierter Inhalt
- beliebig (jedes Element kann als Kindelement angegeben werden)
- leer (keine Kindelemente)
- explizit angegebene Kindelemente
- gemischter Inhalt (gleichzeitiges Auftreten von Elementen und unstrukturiertem Text)

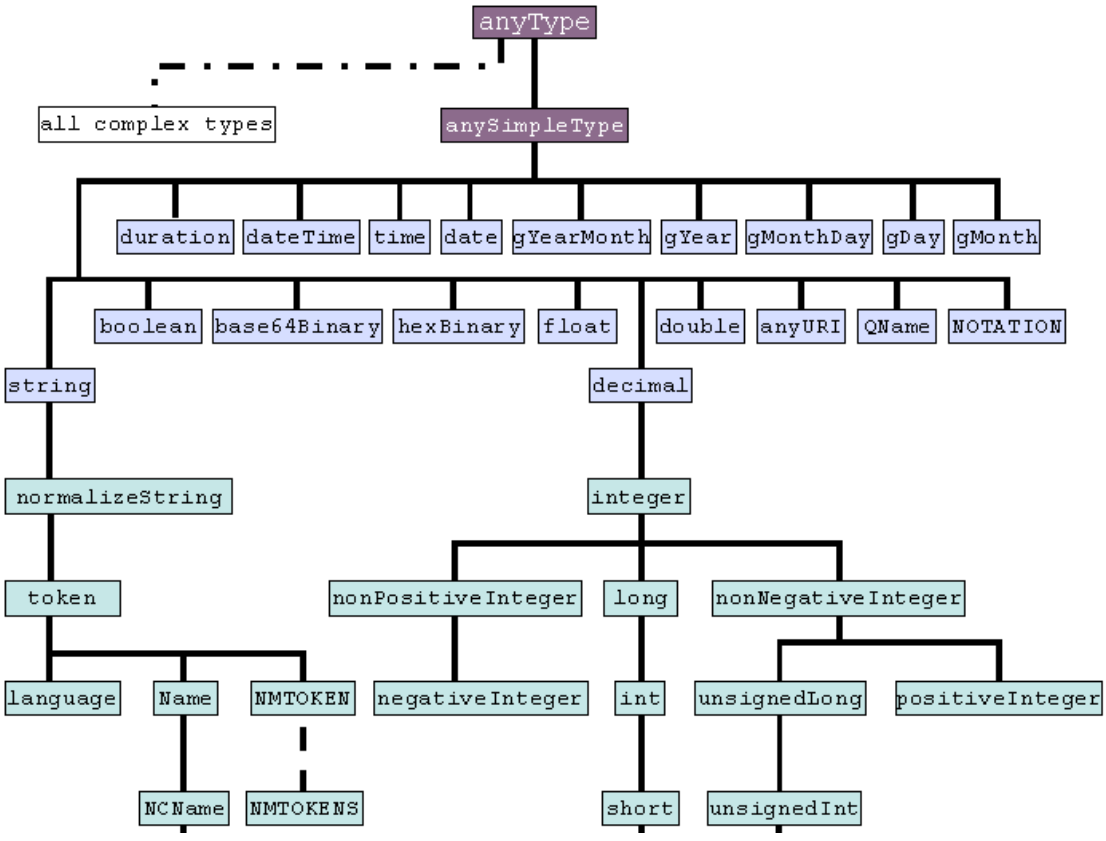
Generell wird jedes Element durch das XSD-Element `element` ausgedrückt.

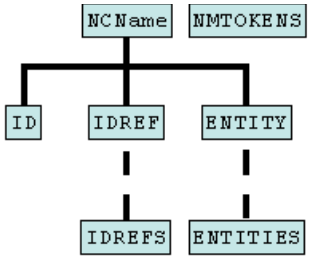
Während die DTD für unstrukturierten Inhalt ausschließliche uninterpretierte Zeichenketten unterstützt, wird die Ausdrucksmächtigkeit durch XML-Schema deutlich gesteigert.
XML-Schema Part 2 definiert insgesamt 44 Primitivtypen. Darunter finden sich die bereits in der DTD angebbaren Element- und Attributtypen, sowie eine Fülle Neuer.

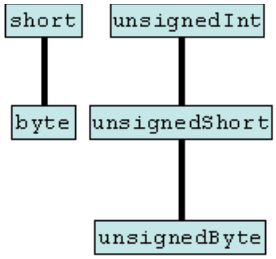
Im Kern zerfallen die XSD-Typen in drei Typklassen:

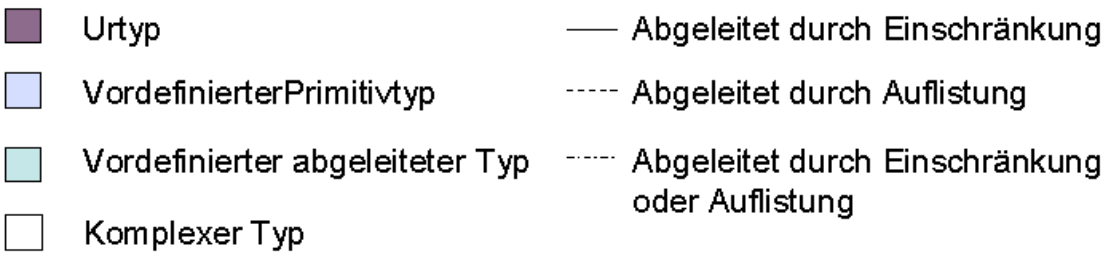
- Atomare und aggregierte Typen
  Atomare Typen bestehen aus unteilbaren Werten. Der Begriff der *Unteilbarkeit* soll dabei auf die nicht erkennbare Unterstrukturierung (`string` besteht zwar aus einzelnen Zeichen) bzw. falls erkennbar, dem nicht explizit unterstützten Zugriff auf die Komponenten (`date` bietet keine Zugriffsmöglichkeit auf die Komponenten Tag, Monat und Jahr), verweisen.
  Die aggregierten Typen teilen sich in Listen-artige und Vereinigungstypen (*Union*). Listen bestehen dabei aus einer geordneten Menge atomarer Typen.
  Vereinigungstypen erlauben hingegen die Verknüpfung Typ-verschiedener Datentypen zu einem neuen.
- Primitive und abgeleitete Typen
  Primitive Datentypen existieren unabhängig von anderen Datentypen, während abgeleitete Datentypen von der Definition ihres Elterntyps abhängig sind.
- Vorgegebene und anwenderdefinierte Typen
  Die vorgegebenen Typen sind diejenigen, die in XML-Schema Part2 beschrieben sind. Alle weiteren Typen eines Schemas sind durch den Anwender definierte abgeleitete Typen.

Durch Erweiterungs- und Aggregationsmechanismen ergibt sich das in der nachfolgenden Abbildung dargestellte Typsystem.









Die Tabelle stellt die angebotenen Typen mit einigen Beispielen dar:

Tabelle 6: Typen in XSD-Schema Part 2

Typname	Beispiel	Bemerkung
string	Hello
World	Jedes beliebige Unicode Symbol gemäß XML-Syntaxproduktion 2
normalizedString	&#20Hello&#20;World	Jedes beliebige Unicode Symbol außer Zeilenvorschub, Wagenrücklauf und Tabulatoren normalizedString ist eine einschränkende Spezialisierung des Typs string
token	Hello World	Jeder normalizedString, unter Weglassung führender, abschließender und mehrfacher Leerzeichen (#x20), sowie Zeilenvorschüben (#xA) und Tabulatoren (#x9). token ist eine einschränkende Spezialisierung des Type normalizedString
Name	aName, _helloWorld, :notAGoodIdea	XML Name gemäß Syntaxproduktion 5. Name ist eine einschränkende Spezialisierung des Typs token
QName	xsd:element, element	Durch Namensraumpräfix qualifizierter Name gemäß Produktion 6 der XML Namespace Recommendation
NCName	aName, _anotherName, X	Name, der keinen Doppelpunkt enthält (*non colonized name*), gemäß Produktion 4 der XML Namespace Recommendation
decimal	-1.23, 12678967.543233, +100000.00, 210	Wertebereich: $i*10^{-n}$, mit i, n aus integer, n>=0 Ein Prozessor muß mindestens 18 Dezimalstellen unterstützen
long	-9223372036854775808, ... -1, 0, 1, ... 9223372036854775807	Wertebereich: 2^{63} <= long <= 2^{63}-1 long ist eine einschränkende Spezialisierung des Typs integer
int	-2147483648, ... -1, 0, 1, ... 2147483647	Wertebereich: -2^{31} <= int <= 2^{31}-1 int ist eine einschränkende Spezialisierung des Typs long
short	-32768, ... -1, 0, 1, ... 32767	Wertebereich: -2^{15} <= short <= 2^{15}-1 short ist eine einschränkende Spezialisierung des Typs int
byte	-128, ...-1, 0, 1, ... 127	Wertebereich: -2^{7} <= byte <= 2^{7}-1 byte ist eine einschränkende Spezialisierung des Typs short

integer	...-1, 0, 1, ...	Wertebereich: entspricht der mathematischen Menge der ganzen Zahlen (Z) integer ist eine einschränkende Spezialisierung des Typs decimal
positiveInteger	1, 2, ...	Wertebereich: entspricht der mathematischen Menge der natürlichen Zahlen (N) positiveInteger ist eine einschränkende Spezialisierung des Typs nonNegativeInteger
negativeInteger	... -2, -1	Wertebereich: {..., -2, -1}, die unendliche Menge der negativen Zahlen negativeInteger ist eine einschränkende Spezialisierung des Typs nonPositiveInteger
nonNegativeInteger	0, 1, 2, ...	Wertebereich: 0 <= nonNegativeInteger nonNegativeInteger ist eine einschränkende Spezialisierung des Typs integer
nonPositiveInteger	... -2, -1, 0	Wertebereich: {..., -2, -1, 0} die unendliche Menge der negativen Zahlen, und die Null nonPositiveInteger ist eine einschränkende Spezialisierung des Typs integer
unsignedLong	0, 1, ... 18446744073709551615	Wertebereich: 0 <= unsignedLong <= 2^{64}-1 unsignedLong ist eine einschränkende Spezialisierung des Typs nonNegativeInteger
unsignedInt	0, 1, ...4294967295	Wertebereich: 0 <= unsignedInt <= 2^{32}-1 unsignedInt ist eine einschränkende Spezialisierung des Typs unsignedLong
unsignedShort	0, 1, ... 65535	Wertebereich: 0 <= unsignedShort <= 2^{16}-1 unsignedShort ist eine einschränkende Spezialisierung des Typs unsignedInt
unsignedByte	0, 1, ... 255	Wertebereich: 0 <= unsignedByte <= 2^{8}-1 unsignedByte ist eine einschränkende Spezialisierung des Typs unsignedShort
float	-1E4, 1267.43233E12, 12.78e-2, 12, INF	32-Bit-Zahl mit einfacher Genauigkeit gemäß IEEE 754-1985. Wertebereich: $m * 2^{e}$, wobei m und $e$integer-Elemente mit $m <= 2^{24}$, und $-149 <= e < 104$ sind.
double	-1E4, 1267.43233E12, 12.78e-2, 12, INF	64-Bit-Zahl mit doppelter Genauigkeit gemäß IEEE 754-1985. Wertebereich: $m * 2^{e}$, wobei m und $e$integer-Elemente mit $m <= 2^{53}$, und $-1075 <= e < 970$ sind.
boolean	true, false, 1, 0	Unterstützung der klassischen zweiwertigen Logik
time	13:20:00-05:00, 13:20:00.000	Uhrzeit, die täglich wiederkehrt, ausgedrückt im Format gemäß ISO 8601
date	2004-05-25	Datumsformat: CCYY-MM-DD, gemäß ISO 8601
gYear	1999, 2001, 2004	Darstellung von Jahren des gregorianischen Kalenders gemäß ISO 8601
gYearMonth	2004-05	Darstellung eines Monats eines bestimmten Jahres des gregorianischen Kalenders gemäß ISO 8601

gDay	----05, ----31	Darstellung eines wiederkehrenden Tages eines Monats gemäß ISO 8601
gMonthDay	--31-12, --01-01	Darstellung eines wiederkehrenden gregorianischen Datums, gebildet aus Tag Monat und Monat im Format --MM-DD, gemäß ISO 8601
gMonth	--03, --12	Monatsformat: --MM-- gemäß ISO 8601
dateTime	2004-05-25T05:37:59.000+02:00	Zeitpunkt, ausgedrückt durch Datum und Uhrzeit; beide gemäß ISO 8601 codiert.
duration	P1Y2M3DT10H30M12.3S Zeitraum von einem Jahr, zwei Monaten, drei Tagen, zehn Stunden, 30 Minuten und 12,3 Sekunden	Nach Größe (Signifikanz) geordnete Koordinate im sechs-dimensionalen Raum aus Jahr, Monat, Tag, Stunde, Minute und Sekunde. Formatdefinition laut ISO 8601
base64Binary	SGVsbG8gd29ybGQhCg==	Base64-Darstellung eines beliebigen Binär-interpretierten Inhaltes gemäß IETF RFC 2045
hexBinary	0FB7	Hexadezimale Darstellung beliebiger Binär-interpretierter Inhalte
anyURI	http://www.jeckle.de	Jede gemäß IETF RFC 2396 bzw. IETF RFC 2732 gültige URI
language	en-GB, en, de-de	Sprachcodierung gemäß IETF RFC 1766 und XML Recommendation language identification. Die Identifikationsnamen werden durch ISO 639 sowie ISO 3166 definiert. language ist eine einschränkende Spezialisierung des Typs token
ID	test, XYZ	XSD-Darstellung des DTD-Typen ID. Zugelassen sind alle Ausprägungen der Namespaceproduktion 4 (NCName). ID ist eine einschränkende Spezialisierung des Typs NCName
IDREF	test, XYZ	XSD-Darstellung des DTD-Typen IDREF. Zugelassen sind alle Ausprägungen der Namespaceproduktion 4 (NCName). IDREF ist eine einschränkende Spezialisierung des Typs NCName
IDREFS	test1 test2 test4, test3 test5	XSD-Darstellung des DTD-Typen IDREFS. Zugelassen sind Listen aus white space separierten Ausprägungen der Namespaceproduktion 4 (NCName). IDREFS ist eine nichtleere Aufzählung von IDREF-Ausprägungen
ENTITY		XSD-Darstellung des DTD-Typen ENTITY. Zugelassen sind alle Satzformen, die der Produktion NCName der XML-Namensräume entsprechen und als ungeparste Entität definiert sind. ENTITY ist eine einschränkende Spezialisierung des Typs NCName

ENTITIES		XSD-Darstellung des DTD-Typen ENTITIES. Zugelassen sind Listen aus white space separierten Ausprägungen des Typs ENTITY. ENTITIES ist eine nichtleere Aufzählung von ENTITY-Ausprägungen
NOTATION		XSD-Darstellung des DTD-Typen NOTATION. Zur Verwendung dieses Typs in einem Schema muß eine Ableitung von NOTATION durch den Anwender definiert werden.
NMTOKEN	US, Deutschland	XSD-Darstellung des DTD-Typen NMTOKEN. Ausprägungen dieses Typs müssen konform zur Produktion 7 der XML-Spezifikation sein. NMTOKEN ist eine einschränkende Spezialisierung des Typs token
NMTOKENS	US UK Aus, Ger	XSD-Darstellung des DTD-Typen NMTOKENS. Zugelassen sind Listen aus white space separierten Ausprägungen des Typs NMTOKEN. NMTOKENS ist eine nichtleere Aufzählung von NMTOKEN-Ausprägungen
anyType	1, 2.3, aGVsb, 06b8f45, test&#20; für&#20;anyType&#0A; <sentence>the quick brown <animal>fox</animal>...</sentence>	Allgemeinster Datentyp. Konzeptionell bildet er die Vereinigung aller angebotenen XSD-Typen.

Die einfachste Form zur **Definition eines Elements mit unstrukturiertem typisierten Inhalt** lautet:

```
<xsd:element
                    name="elementName"
                    type="typeName"/>
```

XSD definiert ferner folgende Charakteristika für Elemente, die durch Attribute der Elementdeklaration ausgedrückt werden:

- **abstract**: falls auf `true` gesetzt, darf ein solches Element nicht in einem XML-Dokument auftreten. Es kann ausschließlich zur Strukturierung des Schemaentwurfs eingesetzt werden und als Basis von Spezialisierungen dienen.
  Vorgabewert ist `false`
- **block**: erlaubt die Kontrolle der Verwendung abgeleiteter Typen. Zugelassene Belegungen beliebige Kombinationen aus `extension`, `restriction` und `substitution` oder der Einzelwert `#all`.
  Ist das Attribut auf `restriction` gesetzt, so dürfen keine (einschränkend) abgeleiteten Typen Ausprägungen des Originaltyps in Instanzdokumenten ersetzen. Dasselbe gilt für `extension` oder als Substitutionsgruppen deklarierte Typen (`substitution`-Belegung).
  Der Wert `#all` versammelt alle möglichen Varianten und verbietet generell die Ersetzung eines Elements durch andere.
- **default**: Vorgabebelegung des Inhalts durch eine beliebige Zeichenkette, die konform zum gewählten Typ ist.
  Dieser Wert wird durch den XML-Prozessor an die Applikation gemeldet, wenn kein Wert im Dokument angegeben wird.
- **final**: verhindert die Ableitung von Typen. Die zulässigen Belegungen sind mit denen für `block` identisch, nur daß durch dieses Attribut die Vererbungsmechanismen bereits auf Schemaebene verboten werden, während `block` ihre Nutzung im Instanzdokument einschränkt.
- **fixed**: erlaubt die konstante Wertbelegung.
- **form**: legt fest, ob das Element im Instanzdokument mit Namensraumpräfix erscheint.
  Zulässige Belegungen: `qualified` (Namensraumpräfix muß angegeben werden) und `unqualified`.
- **id**: erlaubt die eineindeutige Kennzeichnung eines Elements durch eine Schema-weit eindeutige Zeichenkette.
- **minOccurs**: Minimalkardinalität, d.h. Mindestzahl zulässiger Vorkommen dieses Elements. Der Attributinhalt ist ein Element aus nonNegativeInteger.
  Das Attribut ist optional, und wird bei fehlender Angabe mit dem Vorgabewert `1` belegt.
- **maxOccurs**: Maximalkardinalität, d.h. Höchstzahl zulässiger Vorkommen dieses Elements. Der Attributinhalt ist entweder ein Element aus nonNegativeInteger oder die Zeichenkette unbounded zur Kennzeichnung beliebig vieler Auftreten.

Das Attribut ist optional, und wird bei fehlender Angabe mit dem Vorgabewert 1 belegt.

- **name**: Unqualifizierter Name des Elements, konform zur NCName-Produktion der Namensraum-Spezifikation.
- **nillable**: Erlaubt Null-Werte im Instanzdokument, die Semantik ist dabei an die in relationalen Datenbanksystemen verwirklichte angelehnt.
  Die Belegung ist entweder `true` oder `false`, was auch als Vorgabe bei Fehlen dieses Attributs angenommen wird.
- **ref**: Referenz auf eine andere Elementdeklaration zur Übernahme der dort spezifizierten Definitionen.
- **substitutionGroup**: Name einer Gruppe von Elementen, die anstatt des aktuellen Elements im Instanzdokument auftreten dürfen.
- **type**: Ein durch Schema Part 2 vordefinierter Typ, oder jeder beliebige anwenderdefinierte.

Nachfolgend sind einige Elementdeklarationen für unstrukturierten Inhalt versammelt

Beispiel 22:

```
(1)<element name="geburtsdatum" type="xsd:date"/
>
(2)<element name="pi"
(3)     type="xsd:double"
(4)     fixed="3.141592653"
(5)     block="#all"
(6)     final="#all"/>
(7)<element name="vorname"
(8)     type="xsd:token"
(9)     minOccurs="1"
(10)    maxOccurs="unbounded"/>
(11)<element name="artikelNummer"
(12)    type="xsd:NCName"
(13)    form="qualified"/>
```

Die Deklaration `geburtsdatum` definiert ein XML-Element des Typs date zur Darstellung eines Datums. Weitere Festlegungen sind nicht getroffen, daher wird das Element mit `minOccurs` und `maxOccurs` 1 belegt, wodurch es als zwingend anzugebend (*mandatory*) und skalar (d.h. nicht mengenwertig) ausgewiesen wird.
`pi` legt die gleichnamige mathematische Konstante fest. Als Datentyp wurde double, eine Gleitkommazahl mit doppelter Genauigkeit gewählt. Als konstante Belegung wird durch das `fixed` Attribut der entsprechende Zahlenwert festgelegt. Daher muß eine Vorgabebelegung durch das Attribut `default` nicht erfolgen; gemäß Schema-Spezifikation darf sie sogar nicht erfolgen, `fixed` und `default` schließen sich gegenseitig aus. Um eine weitere Spezialisierung des Elements durch Vererbung oder Aggregation zu verhindern wird der Wert von `block` auf `#all` gesetzt, wodurch die Teilnahme an allen Typbildungsmechanismen unterbunden wird.
Die Definition für `vorname` nutzt als Datentyp den token, der automatisch mehrfache, führende und abschließende Leerzeichen sowie sonstige Formatierungssymbole entfernt. Ferner kann dieses Element beliebig häufig auftreten -- `maxOccurs` ist daher auf unbounded gesetzt. Die Fixierung der minimalen Auftrittshäufigkeit auf 1 (`minOccurs`) entspricht der Vorgabebelegung.
Für das Element `artikelNummer` ist als Typ `NCName` ausgewählt, was beliebigen Zeichenketten -- die keinen Doppelpunkt enthalten -- entspricht. Darüberhinaus ist das Attribut `form` mit dem Wert `qualified` versehen. Dies führt dazu, daß das Namensraumkürzel für dieses Element zwingend im Instanzdokument anzugeben ist.

Zur Umsetzung des freien Inhaltsmodells, das beliebige Inhalte aus den definierten Elementen und freien Texten zuläßt, wird ebenfalls auf das Typsystem zurückgegriffen.
Wird das `type` Attribut nicht belegt, so wird gemäß Vorgabe der Typ anyType angenommen. Elemente dieses Typs können beliebige wohlgeformte Inhalte beherbergen.
Die beiden nachfolgenden Angaben sind daher äquivalent.

```
<element
   name="elementName"
   type="xsd:anyType"/>
<element name="elementName"/>
```

XSD prägt den bereits im Kontext der DTD genutzten Typbegriff (dort beschränkt er sich lediglich auf verschiedene Darstellungsformen uninterpretierter Zeichenketten) strenger. Dies zeigt sich deutlich in der Existenz des XSD-Elements `complexType`. Es führt die Möglichkeit einer expliziten, d.h. von der Verwendung losgelösten Typbildung, ein. Syntaktisch kann die complexType-Definition sowohl innerhalb einer Elementdefinition, als auch separat erfolgen.
Den einfachsten Anwendungsfall bildet die eingebettete leere complexType-Definition zur Darstellung des **leeren Inhaltsmodells**.
Die Syntax hierfür lautet (der XSD-Namensraum sei an das Präfix `xsd` gebunden):

```
<xsd:element
```

```
    name="elementName">
    <xsd:complexType/>
</xsd:element>
```

Ein XML-Schema-validierender Parser verhält sich in diesem Falle identisch zu einem (DTD-)validierenden Parser. Daher werden für die obige Festlegung ausschließlich die beiden Darstellungsformen zur Angabe eines leeren Elements (`<elementName/>` bzw. `<elementName></elementName>`) akzeptiert.

Die Befüllung des `complexType`-Elements leitet direkt zum wichtigsten Inhaltsmodell über, dem explizit angegebener Kindelemente.
Zur Festlegung der Elementreihenfolge definiert XML-Schema das Element `sequence`, welches die Angabe der Kindelemente in genau der im Schema angegebenen Reihenfolge erzwingt.
Das Auswahlinhaltsmodell (auch: Selektionsmodell) --- welches alternativ das Auftreten beliebiger Elemente definiert --- wird entsprechend durch das XSD-Element `choice` ausgedrückt.
Eine besondere Variante des Selektionsmodells stellt die `all`-Gruppe dar. Es erlaubt die Angabe der Kindelemente in beliebiger Reihenfolge.
Die drei Ausgangsvarianten können im Rahmen einer Elementdefinition beliebig geschachtelt und auf diesem Wege kombiniert werden.
Am Beispiel der Elementdefinitionen der Projektverwaltung:

Beispiel 23: Einige Elementdefinitionen

```
(1)<?xml version = "1.0" encoding = "UTF-8"?>
(2)<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
(3)     <xsd:element name = "ProjektVerwaltung">
(4)             <xsd:complexType>
(5)                     <xsd:sequence>
(6)                             <xsd:element ref = "Person" maxOccurs = "unbounded"/>
(7)                             <xsd:element ref = "Projekt" maxOccurs = "unbounded"/>
(8)                     </xsd:sequence>
(9)             </xsd:complexType>
(10)    </xsd:element>
(11)    <xsd:element name = "Person">
(12)            <xsd:complexType>
(13)                    <xsd:sequence>
(14)                            <xsd:element name = "Vorname" type = "xsd:token" maxOccurs
= "unbounded"/>
(15)                            <xsd:element name = "Nachname" type = "xsd:token"/>
(16)                            <xsd:element ref = "Qualifikationsprofil" minOccurs = "0"/>
(17)                    </xsd:sequence>
(18)            </xsd:complexType>
(19)    </xsd:element>
(20)    <xsd:element name = "Projekt">
(21)            <xsd:complexType/>
(22)    </xsd:element>
(23)    <xsd:element name = "Qualifikationsprofil">
(24)            <xsd:complexType mixed = "true">
(25)                    <xsd:sequence>
(26)                            <xsd:element name = "Qualifikation" type = "xsd:string"
minOccurs = "0" maxOccurs = "unbounded"/>
(27)                            <xsd:element name = "Leistungsstufe" type = "xsd:string"
minOccurs = "0" maxOccurs = "unbounded"/>
(28)                    </xsd:sequence>
(29)            </xsd:complexType>
(30)    </xsd:element>
(31)</xsd:schema>
```

Download des Beispiels

Das Schema enthält alle Elementdefinitionen für die Projektverwaltung. Innerhalb jedes `element`-Elements sind die entsprechenden Kindelemente in `sequence`-Strukturen eingebettet. Die Elemente müssen daher in der Reihenfolge ihres Auftretens im Schema auch im Instanzdokument wiedergegeben werden.
Von besonderem Interesse ist die Definition des *Qualifikationsprofil*s. Es handelt sich dabei um ein **mixed content model**, ausgedrückt durch das Boole'sche Attribut `mixed` (in Spezifikation nachschlagen).
Darüberhinaus enthält das Beispiel neben *lokalen Elementdeklarationen*, die sich vollständig im Elternelement finden (wie `Vorname`, `Nachname` und `Qualifikation`), auch *globale Elementdeklarationen*, die zunächst deklariert und in einem zweiten Schritt durch Referenzierung als Kindelemente verwendet werden (wie `Person` und `Projekt` innerhalb `Projektverwaltung`, oder `Qualifikationsprofil` innerhalb des Elements `Person`). Hierdurch können vollständige Elemente an verschiedenen Stellen im Schema referenziert und so verwendet werden. Die Definition ist der lokalen ebenbürtig und wird im Instanzdokument identisch behandelt. Zusammenfassend läßt sich festhalten: Mit dem Referenzierungsmechanismus für Elemente kann eine einfache Form der Wiederverwendung umgesetzt werden.
Den Zeichenketten-artigen Elementtypen wurde durchgehend der XSD-Typ `string` zugewiesen.

Durch die Referenzierungsmöglichkeit existiert eine erste Möglichkeit zur Wiederverwendung bereits im Schema definierter Elemente. Jedoch werden Elemente hierbei zwingend in ihrer vollständigen Definition, d.h. Name, Typ und Inhaltsmodell, eingebunden.
XML-Schema bietet die Möglichkeit, strukturierte Typen, die ausschließlich durch ihr Inhaltsmodell definiert werden, festzulegen. In der Konsequenz verändert sich der durch die DTD formulierte Typbegriff hin zu einer eher an den Programmiersprachen orientierten Sichtweise, da die Benennung des Typs von der Namensgebung der typisierten Instanz separiert wird.
Syntaktisch erfolgt die Typbildung durch die Benennung des `complexType`-Elements durch ein Attribut `name`. Um die mehrfache Verwendung eines solchen Typen zu ermöglichen, muß seine Definition zwingend auf einer Baumstufe erfolgen, die für alle nutzenden Elemente erreichbar ist. Üblicherweise werden daher diese Definitionen auf der ersten Stufe, direkt unterhalb des Wurzelknotens, plaziert.
Zur Unterscheidung dieser *benannten komplexen Typen* werden die bisher genutzten -- namenlosen Typen -- als *anonyme komplexe Typen* bezeichnet.
Das nachfolgende Beispiel zeigt die Definition eines benannten komplexen Typen am Beispiel des Elements `Person`:

Beispiel 24: Nutzung benannter komplexer Typen

```
(1)<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
(2)   <xsd:complexType name="PersonType">
(3)        <xsd:sequence>
(4)        <xsd:element name = "Vorname" type = "xsd:string"
(5)                        maxOccurs = "unbounded"/>
(6)        <xsd:element name = "Nachname" type = "xsd:string"/>
(7)        <xsd:element ref = "Qualifikationsprofil" minOccurs = "0"/>
(8)   </xsd:sequence>
(9)   </xsd:complexType>
(10)
(11)   <xsd:element name = "ProjektVerwaltung">
(12)      <xsd:complexType>
(13)         <xsd:sequence>
(14)            <xsd:element name="Person" type="PersonType" maxOccurs = "unbounded"/>
(15)            <xsd:element ref = "Projekt" maxOccurs = "unbounded"/>
(16)         </xsd:sequence>
(17)      </xsd:complexType>
(18)   </xsd:element>
(19)
(20)   <xsd:element name = "Projekt">
(21)      <xsd:complexType/>
(22)   </xsd:element>
(23)
(24)   <xsd:element name = "Qualifikationsprofil">
(25)      <xsd:complexType mixed = "true">
(26)         <xsd:sequence>
(27)            <xsd:element name = "Qualifikation" type = "xsd:string"
(28)                           minOccurs = "0" maxOccurs = "unbounded"/>
(29)            <xsd:element name = "Leistungsstufe" type = "xsd:string"
(30)                           minOccurs = "0" maxOccurs = "unbounded"/>
(31)         </xsd:sequence>
(32)      </xsd:complexType>
(33)   </xsd:element>
(34)</xsd:schema>
```

Download des Beispiels

Das Schema zeigt die Definition des komplexen Typen `PersonType`. Dieser Typ wird zur Festlegung des Inhaltsmodells des Elements `Person` verwendet.

Definition eigener Datentypen durch Vererbung:
Zur Unterstützung von Wiederverwendung und Erhöhung der Strukturierung des Entwurfs definiert XSD ein Vererbungskonstrukt zur Bildung neuer komplexer Typen auf der Basis bereits bestehender.
Zwei verschiedene Ableitungssemantiken werden angeboten:

- Ableitung durch Einschränkung (*derivation by restriction*)
  Der erbende Subtyp gibt eine engere Definition des Supertypen
- Ableitung durch Erweiterung (*derivation by extension*)
  Der erbende Subtyp erweitert die Definition des Supertypen

Das nachfolgende Beispiel zeigt die Anwendung der einschränkenden Ableitung.
Hierbei erbt der benannte komplexe Typ `childType` von `parentType`. Innerhalb des -- aus syntaktischen Gründen notwendigen -- Elements `complexContent` findet sich die Definition der Vererbung im Element `restriction`, das `base`-Attribut verweist auf den benannten Elterntypen.
Der Inhalt des `restriction`-Elements gleicht der Inhaltsmodelldefinition des komplexen Typen: Auch hier werden Elemente und ihre Auftrittsstruktur (im betrachteten Beispiel `sequence`) angegeben. Die

Elementdefinition des Elements elementA in childType schränkt die gleichnamige Elementdefinition innerhalb des Elterntypen ein. Nachvollziehbar wird diese Einschränkungsbeziehung zwischen short und int bei Betrachtung der Datentyphierarchie und der Typdefinition der verwendeten Primitivtypen. So bildet short per definitionem eine eingeschränkte Untermenge von int an. (Die entsprechende XSD-Definition findet sich im *Schema für Schema*).
Die beiden Elementdefinitionen usage1 und usage2 zeigen die Verwendung der anwenderdefinierten Typen.

Beispiel 25: Einschränkende Typableitung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(3)<xsd:complexType name="parentType">
(4)     <xsd:sequence>
(5)             <xsd:element name="elementA" type="xsd:int"/>
(6)     </xsd:sequence>
(7)</xsd:complexType>
(8)
(9)<xsd:complexType name="childType">
(10)<xsd:complexContent>
(11)    <xsd:restriction base="parentType">
(12)            <xsd:sequence>
(13)                    <xsd:element name="elementA" type="xsd:short"/>
(14)            </xsd:sequence>
(15)    </xsd:restriction>
(16)</xsd:complexContent>
(17)</xsd:complexType>
(18)
(19)<xsd:element name="usage1" type="parentType"/>
(20)<xsd:element name="usage2" type="childType"/>
(21)
(22)</xsd:schema>
```

Download des Beispiels

Durch das strukturierte Inhaltsmodell ergeben sich über die reine Typisierung hinausgehende Möglichkeiten zur Einschränkung der Inhalte. Die nachfolgende Tabelle stellt einige Varianten zusammen.

Tabelle 7: Beispiele für zulässige Restriktionen

Basistyp	Restriktion	Bemerkung
	default	Zusätzliche Belegung eines Elements mit einem Vorgabewert
	fixed	Beschränkung eines zunächst frei wählbaren Elements auf konstanten Inhalt
	type	Definition eines Typen für ein zunächst untypisiertes Element. (Auch hierbei handelt es sich um eine einschränkende Redefinition, da allen Elementen ohne Typdefinition standardmäßig der Typ anyType zugeordnet wird.)
minOccurs=n_1, maxOccurs=m_1	minOccurs=n_2, maxOccurs=m_2	Restriktion der Auftrittshäufigkeit auf eine geringere Anzahl. Daher gilt: $n_1 <= n_2$ und $m_1 >= m_2$

Die direkte Umkehrung der einschränkenden Spezialisierung bildet die *erweiternde Spezialisierung*. Sie greift nicht verändernd auf die Elemente des Supertyps zu, sondern definiert zusätzliche neue. Untenstehendes XSD-Schema zeigt dies am Beispiel des Supertyps parentElement, der durch das abgeleitete Kindelement childElement erweitert wird. Hierzu definiert childElement ein zusätzliches elementB.

Beispiel 26: Erweiternde Typableitung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(3)     <xsd:complexType name="parentElement">
(4)             <xsd:sequence>
(5)                     <xsd:element name="elementA"/>
(6)             </xsd:sequence>
(7)     </xsd:complexType>
(8)
(9)     <xsd:complexType name="childElement">
(10)            <xsd:complexContent>
(11)                    <xsd:extension base="parentElement">
(12)                            <xsd:sequence>
(13)                                    <xsd:element name="elementB"/>
(14)                            </xsd:sequence>
(15)                    </xsd:extension>
(16)            </xsd:complexContent>
(17)    </xsd:complexType>
(18)</xsd:schema>
```

Download des Beispiels

Zusätzlich sieht XML Schema die Möglichkeit vor, komplexe Typen von simplen abzuleiten. Dies mag auf den ersten Blick ungewöhnlich erscheinen, eröffnet es doch scheinbar einen Weg, unstrukturierte Typen in strukturierte zu überführen.
Bei näherer Betrachtung offenbart sich jedoch, daß hier lediglich der Ableitungsbegriff überladen wurde, um einen einfachen Weg zur Verknüpfung der beiden Inhaltsmodelle strukturierter „XML-artiger“ Inhalt -- wie er durch `complexTypes` repräsentiert wird -- auf der einen, und unstrukturierter Inhalt -- wie er durch die einfachen Datentypen repräsentiert wird -- auf der anderen Seite, zu erhalten.

Beispiel 27: Ableitung eines komplexen Typen von einem Simplen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xs:schema
(3)     xmlns:xs="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)     <xs:element name="Vorname">
(7)             <xs:complexType>
(8)                     <xs:simpleContent>
(9)                             <xs:extension base="xs:string">
(10)                                    <xs:attribute
(11)                                            name="rufname"
(12)                                            type="xs:boolean"/>
(13)                            </xs:extension>
(14)                    </xs:simpleContent>
(15)            </xs:complexType>
(16)    </xs:element>
(17)</xs:schema>
```

Download des Beispiels

Durch die im Beispiel dargestellte Syntax wird es ermöglicht unstrukturiert-getypten Elementen Attribute zuzuordnen, obwohl diese eigentlich Bestandteil der Definition komplex-getyper Elemente sind.

So wird im Beispiel dem Element `Vorname` sowohl der simple Typ `string`, als auch durch den Ableitungsmechanismus das Attribut `rufname` -- im Rahmen eines `complexType`, zugeordnet.
Die Typisierung des Elements erfolgt hierbei nicht durch das `type`-Attribut innerhalb der Elementdeklaration, sondern innerhalb der `simpleContent`-Festlegung.

Neben der anwenderdefinierten Bildung komplexer Typen steht es dem XSD-Modellierer auch offen, eigene (primitive) Datentypen festzulegen oder eigene Typen von bestehenden abzuleiten.
Hierfür definiert XML-Schema Part1 das Element simpleType. Für einfache Typen ist jedoch nur die einschränkende Vererbung (*restriction*) zugelassen. Dies liegt in der praktischen Beherrschbarkeit des Typsystems begründet. Durch die strikte Restriktionssemantik ergibt sich die Möglichkeit kontravarianter Substitution, wie sie bei objektorientierten Typsystemen und Vererbungsstrukturen anzutreffen ist. Dies bedeutet, daß an jeder Stelle, an der eine Ausprägung eines Supertyps erwartet wird, auch -- unter Erhalt der Typrestriktion -- eine Ausprägung eines Subtypen auftreten darf. Beispielhaft: Wird an einer Stelle des Instanzdokumentes durch das Schema das Auftreten einer Ausprägung von `integer` verlangt, so kann der Anwender auch Ausprägungen der Subtypen `int`, `short` oder `byte` angeben ohne die Gültigkeit des XML-Dokuments zu beeinträchtigen.

Vereinigungstypen werden aus einer nichtleeren Menge von Ausgangstypen gebildet.

Das Beispiel zeigt die Definition eines Typen `termin`, der den vorgegebenen Primitivtypen `date` und eine Liste `NamenDerWochentage` (deren Definition nicht dargestellt ist) vereinigt. Insbesondere zeigt der Ausschnitt die Möglichkeit der Vereinigungsbildung auch über aggregierte Typen.

```
(1)<xs:simpleType name="termin">
(2)     <xs:union memberTypes="xs:date NamenDerWochentage"/>
(3)</xs:simpleType>
```

Das XSD-Beispiel zeigt, als Fragment der XML-Schemaspezifikation, die Definition des vorgegebenen Typs `short`, einer einschränkenden Spezialisierung des Typs `int`.
Am Beispiel gut nachvollziehbar sind die beiden Schritte zur Bildung eines eigenen Typen:

1. Auswahl eines Ausgangstypen (später Elementtyp (bei aggregierten Typen) oder Basistyp (bei abgeleiteten Typen))
2. Typdefinition durch Anwendung der entsprechenden Typkonstruktion und evtl. Einschränkung verschiedener Charakteristika

Im Beispiel wird der kleinste und größte gültige Wert (`minInclusive` bzw. `maxInclusive`) des neuen Typen `short` gegenüber dem Basistypen beschränkt.

Beispiel 28: Einschränkende Spezialisierung eines simplen Typen

```
(1)<xsd:simpleType name="short" id="short">
(2)     <xsd:restriction base="xsd:int">
(3)             <xsd:minInclusive value="-32768"
(4)                     id="short.minInclusive"/
>
(5)             <xsd:maxInclusive value="32767"
(6)                     id="short.maxInclusive"/>
(7)     </xsd:restriction>
(8)</xsd:simpleType>
```

Die Bildung aggregierter Typen folgt demselben Muster. Jedoch tritt an die Stelle der Ableitung die Spezifikation des Aggregationstyps (im Beispiel *Liste*) und Angabe des Inhaltstyps (im Beispiel `string`).

Beispiel 29: Bildung eines Aggregationstypen

```
(1)<xsd:simpleType name="WarenkorbElemente">
(2)     <xsd:list itemType="xsd:string"/>
(3)</xsd:simpleType>
```

Nachfolgend sind die verschiedenen Beschränkungsmöglichkeiten zusammengefaßt:

- `length`
  Längenbeschränkung bei atomaren Typen bzw. Beschränkung der Elementanzahl bei aggregierten Typen.
  Längenbeschränkung eines simplen Typs:

```
(1)<xs:simpleType name="Postleitzahl">
(2)     <xs:restriction base="xs:string">
(3)             <xs:length value="5"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

  Beschränkung der Elementanzahl einer Liste:

```
(1)<xs:simpleType name="VornamenList">
(2)     <xs:list itemType="xs:token"/>
(3)</xs:simpleType>
(4)<xs:simpleType name="VornamenRestrictedList">
(5)     <xs:restriction base="VornamenList">
(6)             <xs:length value="5"/>
(7)     </xs:restriction>
(8)</xs:simpleType>
```

- `minLength`
  Minimale Länge (bei atomaren Typen bzw. minimale Elementanzahl bei aggregierten Typen.
  Beispiel (der aggregierte Typ `VornamenRestrictedList` muß mindestens einen Eintrag enthalten):

```
(1)<xs:simpleType name="VornamenList">
(2)     <xs:list itemType="xs:token"/>
(3)</xs:simpleType>
(4)<xs:simpleType name="VornamenRestrictedList">
```

```
(5)     <xs:restriction base="VornamenList">
(6)             <xs:minLength value="1"/>
(7)     </xs:restriction>
(8)</xs:simpleType>
```

Beispiel (der spezialisierte atomare Typ Titel muß aus mindestens fünf Zeichen bestehen):

```
(1)<xs:simpleType name="Titel">
(2)     <xs:restriction base="xs:string">
(3)             <xs:minLength value="5"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- maxLength
  Maximale Länge bei atomaren Typen bzw. maximale Elementzahl bei aggregierten Typen.
  Beispiel (der aggregierte Type VornamenRestrictedList muß mindestens einen, jedoch höchstens drei, Einträge enthalten):

```
(1)<xs:simpleType name="VornamenList">
(2)     <xs:list itemType="xs:token"/>
(3)</xs:simpleType>
(4)<xs:simpleType name="VornamenRestrictedList">
(5)     <xs:restriction base="VornamenList">
(6)             <xs:minLength value="1"/>
(7)             <xs:maxLength value="3"/>
(8)     </xs:restriction>
(9)</xs:simpleType>
```

  Beispiel (der spezialisierte atomare Typ Titel muß aus mindestens fünf, darf jedoch aus höchstens 80 Zeichen bestehen):

```
(1)<xs:simpleType name="Titel">
(2)     <xs:restriction base="xs:string">
(3)             <xs:minLength value="5"/>
(4)             <xs:maxLength value="80"/>
(5)     </xs:restriction>
(6)</xs:simpleType>
```

- pattern
  Erlaubt die Inhaltsdefinition durch Muster (reguläre Ausdrücke).
  XML-Strukturen sind nicht durch reguläre Ausdrücke darstellbar, hierfür können lediglich die vorgestellten Strukturierungsprimitive eingesetzt werden; daher sind Muster auf atomare Typen beschränkt.
  Die XML-Schemaspezifikation definiert einzelne Symbole und Symbolsequenzen, sowie einige abkürzende Schreibweisen zum Aufbau beliebiger Ausdrücke. Alle anderen Zeichen können direkt in die Musterspezifikation übernommen werden.
  Die aus der Verarbeitung regulärer Ausdrücke mit anderen Programmiersprachen oder Werkzeugen bekannten Quantifier stehen in der bekannten Semantik zur Verfügung.
  Sei S eine Zeichenkette aus einer beliebigen Anzahl von Zeichen (d.h. sie kann auch leer sein!), dann gilt:

Tabelle 8: Übersicht der Quantifier

Quantifiziertes Mustersymbol	Bedeutung
S	Alle Zeichenketten, die S genau entsprechen.
S?	Alle Zeichenketten, die S genau entsprechen oder die leere Zeichenkette.
S*	Alle Reihungen von S; insbesondere entspricht S* auch der leere Zeichenkette.
S+	Alle Reihungen von S, die S mindestens einmal enthalten. Entspricht der Formulierung: S S*
S{n,m}	Alle Zeichenketten, die aus mindestens *n*, jedoch höchstens *m* Auftreten von S bestehen. Durch *n=0* lassen sich somit optionale, nach oben begrenzte, Auftrittsanzahlen realisieren, sowie durch *n=m=0* die leere Zeichenkette ausdrücken
S{n}	Alle Zeichenketten, die aus genau *n* Auftreten von S bestehen. Entspricht der Formulierung: S{n,n}

S{n,}	Alle Zeichenketten, die aus mindestens *n* Auftreten von S bestehen. Entspricht der Formulierung: S{n} S*

Zur Darstellung nicht-druckbarer Zeichen oder von Metasymbolen werden folgende Fluchtsymbole angeboten:

Tabelle 9: Übersicht der Escape-Symbole

Mustersymbol (*escape character*)	**ausgedrücktes Zeichen**
\n	Zeilenumbruch (#xA)
\r	Zeilenvorschub (#xD)
\t	Tabulator (#x9)
\\	\
\\|	|
\.	.
\-	-
\^	^
\?	?
*	*
\+	+
\{	{
\}	}
\(	(
\)	)
\[	[
\]	]

Ferner sind noch eine Reihe häufig benötigter Zeichenfamilien definiert und in den Kategorien Buchstaben (Letter), Marken (Marks), Zahlen (Numbers), Interpunktion (Punctuation), Trennsymbole (Separators) sowie sonstige (Others) zusammengefaßt. Die Zeichenfamilien innerhalb dieser Kategorien entsprechen den in Unicode v3.1 definierten general categories in Namen und Aufbau.

Tabelle 10: Buchstaben

symbolische Darstellung	**Bedeutung**
L	(All Letters) Alle Buchstaben
Lu	(Uppercase) Alle Großbuchstaben
Ll	(Lowercase) Alle Kleinbuchstaben
Lt	(Titlecase) Sprachabhängige (potentielle) Großschreibung des ersten Buchstabens eines Wortes. (vgl. UNICODE technical report #21 sowie Derived General Category Lt).
Lm	(Modifiers) Zusammenfassung der verschiedensten Ton- und Betonungszeichen
Lo	(Others) Zusammenfassung von Zeichen, die in keine der sonstigen L-Zeichenfamilien fallen

Tabelle 11: Markierungssymbole

symbolische Darstellung	**Bedeutung**
M	(All Marks) Alle Markierungssymbole
Mn	(Non-Spacing) Markierungssymbole, ausschließlich Leerzeichen
Mc	(Space combining) Markierungssymbole mit Leerzeichen kombiniert
Me	(Enclosing) Markierungssymbole, die andere Zeichen umschließen

Tabelle 12: Zahlen

symbolische Darstellung	Bedeutung
N	(All Numbers) Beliebige Ziffer; daher nicht notwendigerweise arabisch. Unicode stellt eingekreiste (#x2460-#x2473), geklammerte (#x2474-#x2487) sowie Ordinalzahlen (mit abschließendem Punkt) (#x2488-#x249b) zur Verfügung.
Nd	(Decimal Digit) eine arabische Ziffer
Nl	(Letter) auf Buchstaben beruhende Zifferndarstellungen, wie römische Ziffernsymbole (#x20dd-#x217f)
No	(Other) Alle sonstigen Ziffernsymbole

Tabelle 13: Interpunktionszeichen

symbolische Darstellung	Bedeutung
P	(Punctuation) Alle Interpunktionssymbole
Pc	(Connector) Verbindende Interpunktionssymbole (z.B. Unterstrich (#x5f)
Pd	(Dash) Verschiedene Verbindungsstriche
Ps	(Open) Öffnende Bereichssymbole wie die verschiedenen Klammertypen
Pe	(Close) Schließende Bereichssymbole wie die verschiedenen Klammertypen
Pi	(Initial Quote) Öffnende Anführungszeichen. In einigen Fällen Verhalten identisch zu Ps oder Pe
Pf	(Final Quote) Schließende Anführungszeichen. In einigen Fällen Verhalten identisch zu Ps oder Pe
Po	(Other) Alle anderen Interpunktionssymbole

Tabelle 14: Separatoren

symbolische Darstellung	Bedeutung
Z	Alle Separatoren
Zs	(Space) Trennende Leerzeichen
Zl	(Line) Zeilentrenner (#x2028)
Zp	(Paragraph) Absatztrenner (#x2029)

Tabelle 15: Symbole

symbolische Darstellung	Bedeutung
S	Alle Symbole
Sm	(Math) Verschiedene mathematische Symbole (Operatoren, Pfeile, etc.)
Sc	(Currency) Währungssymbole (Dollarzeichen: #x24, Eurozeichen: #x20ac)
Sk	(Modifier) Ton- und Betonungssymbole (ähnlich zu Lm)
So	(Other) Alle anderen Symbole

Tabelle 16: Sonstige Zeichen

symbolische Darstellung	Bedeutung
C	Alle sonstigen
Cc	(Control) Nicht-druckbare Kontrollzeichen
Cf	(Format) Formatierungszeichen (z.B. syrisches Abkürzungssymbol)
Co	(Private Use) ungefähr 137500 Zeichen zur freien anwenderdefinierten Belegung
Cn	(Not Assigned) Zeichen, denen innerhalb Unicode explizit keine Belegung zugewiesen wurde

Reguläre Ausdrücke können innerhalb des pattern-Elements direkt angegeben werden. Die Zeichenkettenfamilien werden durch ihre symbolische Darstellung, eingeleitet durch \p und durch geschweifte Klammern umschlossen, dargestellt.
Zusätzlich ist durch \P das Komplement zu jeder der aufgeführten Zeichenkettenfamilien definiert.
Ergänzend kann die Definition der zulässigen Zeichengruppen vollständig wahlfrei erfolgen. Hierzu wird das Negationssymbol ^ angeboten, welches eine Aufzählung von Zeichen von der Verwendung ausschließt.
Darüberhinaus können auf der Basis simpler Mengendifferenzoperationen eigene Zeichenklassen komfortabel definiert werden.
Für die am häufigsten benötigten Zeichenklassen sind durch XML-Schema bereits vorgegebene abkürzende Schreibweisen definiert. Hierzu werden bereits die bisher vorgestellten Syntaxmechanismen angewendet.

Tabelle 17: Zeichensequenzen

abkürzende Schreibweise	entsprechende Langform
.	[^\n\r]
\s	[#x20\t\n\r]
\S	[^\s]
\i	Die initialen Zeichen eines gültigen XML-Namens; entspricht dem ersten Teil der Syntaxproduktion 5
\I	[^\i]
\c	Diejenigen Zeichen, die der XML-Syntaxproduktion 4 entsprechen
\C	[^\c]
\d	\p{Nd}
\D	[^\d]
\w	[#x0000-#x10FFFF]-[\p{P}\p{S}\p{C}] Alle Zeichen, außer der Klassen für Interpunktions- und Separatorenzeichen, sowie der Klasse der sonstigen Zeichen.
\W	[^\w]

Beispiel (eine vereinfachte, d.h. unter Nicht-Berücksichtigung von Behörden- und Diplomatennummern) deutsche Autonummer im bekannten Format: Einführende Bezeichnung des Landkreises durch mindestens einen, jedoch höchstens drei Großbuchstaben, gefolgt von einem trennenden Bindestrich, an den sich ein oder zwei weitere Großbuchstaben anschließen. Auf ein Leerzeichen folgen eine, jedoch höchstens vier Ziffern:

```
(1)<xs:simpleType name="gerAutoNummer">
(2)     <xs:restriction base="xs:string">
(3)             <xs:pattern value="\p{Lu}{1,3}-\p{Lu}{1,2} \p
{Nd}{1,4}"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

Weitere Informationen: Anhang F -- Reguläre Ausdrücke -- der XML-Spezifikation

- enumeration
  Festlegung von zulässigen Werten eines Typs durch vollständige Aufzählung.
  Beispiel (die Ampelfarben: rot, gelb, grün):

```
(1)<xs:simpleType name="ampelfarben">
(2)     <xs:restriction base="xs:string">
(3)             <xs:enumeration value="rot"/>
```

```
(4)             <xs:enumeration value="gelb"/>
(5)             <xs:enumeration value="grün"/>
(6)     </xs:restriction>
(7)</xs:simpleType>
```

- whitespace
  Behandlung von white spaces innerhalb eines Elements des definierten Typs. Erlaubte Belegungen und ihre Bedeutung:
  preserve: Keine Veränderung des Inhaltes durch Normalisierung; evtl. enthaltene white spaces bleiben erhalten
  replace: Alle Vorkommen von Tabulatoren, Zeilenvorschub und Wagenrücklauf werden durch Leerzeichen (#x20) ersetzt
  collapse: Nach der Substitution gemäß *replace* werden mehrfach auftretende Leerzeichen zu einem einzigen kompaktifiziert, sowie führende und abschließende Leerzeichen entfernt.
  (in Spezifikation nachschlagen)
- maxInclusive
  Höchster zulässiger numerischer Wert. Im mathematischen Sinne sind daher alle Werte kleiner oder gleich dem im value-Attribut angegebenen zugelassen.
  Beispiel (Zahlen <= 100):

```
(1)<xs:simpleType name="uHu">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:maxInclusive value="100"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

  Anmerkung: In vielen Fällen kann eine maxInclusive-Festlegung ohne Informationsverlust in eine äquivalente maxExclusive-Definition überführt werden.
- maxExclusive
  Wert „unterhalb“ (im mathematischen Sinne „kleiner“) dem alle numerischen Belegungen zugelassen sind.
  Beispiel (Zahlen < 100):

```
(1)<xs:simpleType name="uHu">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:maxExclusive value="100"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

  Als Belegungen des Typs uHu sind alle Zahlen kleiner als 100 zugelassen. Durch die Verwendung des XSD-Typen decimal als Basistyp kann auch keine verlustfreie Überführung in eine maxInclusive-Festlegung überführt werden, da hierfür die größte zugelassene Zahl fixiert werden müßte, was jedoch die Typdefinition von decimal explizit offen läßt.
- minInclusive
  Kleinster zugelassener Wert.
  Beispiel (Zahlen >= 25):

```
(1)<xs:simpleType name="uFz">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:minInclusive value="25"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- minExclusive
  Kleinster Wert, der nicht mehr zugelassen ist. Im mathematischen Sinne sind alle Zahlen, die größer sind, gültige Belegungen.
  Beispiel (Zahlen > 25):

```
(1)<xs:simpleType name="uFz">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:minExclusive value="25"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- totalDigits
  Gesamtstellen einer Zahl, gebildet aus der Summe der Vorkomma- und der Nachkommastellen.
  Beispiel (Zahlen mit höchstens sieben Stellen):

```
(1)<xs:simpleType name="myNumber">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:totalDigits value="7"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- fractionDigits
  Anzahl der Nachkommastellen eines Dezimalbruches.
  Beispiel (Zahl mit höchstens neuen Stellen, davon zwei Nachkommastellen):

```
(1)<xs:simpleType name="myNumber">
(2)     <xs:restriction base="decimal">
(3)             <xs:totalDigits value="9"/>
(4)             <xs:fractionDigits value="2"/>
(5)     </xs:restriction>
(6)</xs:simpleType>
```

Definition von Attributen:
Die Attributdeklaration erfolgt durch das XSD-Element attribute. Die Mächtigkeit entspricht auch hier, wie bereits für die Elemente verwirklicht, einer Obermenge der DTD. So können neben optionalen, zwingenden und konstanten Attributen auch Aufzählungsattribute und Mengen realisiert werden. Hierbei wurde auf die Orthogonalität zum durch simpleType geschaffenen Typmechanismus geachtet.
Die Charakteristika (ausgedrückt in Attributen des XSD-Elements attribute) einer Attributdeklaration umfassen:

- **name**: Ein Doppelpunkt-freier Namen (NCName) gemäß Namensraumproduktion 7.
- **id**: erlaubt die eineindeutige Kennzeichnung eines Attributs durch eine Schema-weit eindeutige Zeichenkette.
- **default**: Belegung mit Vorgabewert.
- **fixed**: Konstante Belegung.
- **type**: Typ des Attributes, definiert durch einen simpleType.
- **form**: Legt fest, ob der Attributname im XML-Instanzdokument durch ein Namensraumpräfix eingeleitet wird (Belegung: qualified, andernfalls unqualified).
- **ref**: Verweis auf eine globale Attributdefinition.
- **use**: Verwendung des Attributes, Wert entspricht optional, required oder prohibited. Vorgabegemäß wird optional angenommen und das Attribut damit nicht zwingend im XML-Dokument erwartet. Den Gegensatz hierzu bildet required, wodurch das Attribut als zwingend anzugeben definiert wird. prohibited verbietet die Nutzung des Attributes im XML-Dokument.

Anmerkung: Einen Anwendungsfall der Belegung prohibited für use bilden Attribute, die innerhalb des Schemas bereits definiert sind, jedoch noch nicht zur allgemeinen Nutzung freigegeben wurden.

Beispiel 30: Einige Attributdefinitionen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
(3)     <xsd:attribute name="myAtt1"/>
(4)
(5)     <xsd:attribute name="myAtt2" type="xsd:decimal"/>
(6)
(7)     <xsd:attribute name="myAtt3">
(8)             <xsd:simpleType>
(9)                     <xsd:restriction base="xsd:int">
(10)                            <xsd:minInclusive value="10"/>
(11)                            <xsd:maxInclusive value="20"/>
(12)                    </xsd:restriction>
(13)            </xsd:simpleType>
(14)    </xsd:attribute>
(15)
(16)    <xsd:simpleType name="myType1">
(17)            <xsd:restriction base="xsd:string">
(18)                    <xsd:maxLength value="5"/>
(19)            </xsd:restriction>
(20)    </xsd:simpleType>
(21)    <xsd:attribute name="myAtt4" type="myType1"/>
(22)
(23)
(24)
(25)    <xsd:element name="foo">
(26)            <xsd:complexType>
(27)                    <xsd:attribute ref="myAtt1" use="optional"/>
(28)                    <xsd:attribute ref="myAtt2" use="required"/>
(29)                    <xsd:attribute ref="myAtt3" use="prohibited"/>
(30)                    <xsd:attribute ref="myAtt4"/>
(31)                    <xsd:attribute name="myAtt5" type="xsd:date" id="myDate"/>
(32)                    <xsd:attribute name="myAtt6">
(33)                            <xsd:simpleType>
(34)                                    <xsd:restriction base="xsd:float">
(35)                                            <xsd:totalDigits value="5"/>
(36)                                    </xsd:restriction>
```

```
(37)                            </xsd:simpleType>
(38)                    </xsd:attribute>
(39)            </xsd:complexType>
(40)    </xsd:element>
(41)
(42)</xsd:schema>
```

Download des Beispiels

Das Beispiel zeigt einige Varianten der Attributdeklaration. So definieren `myAtt1` mit `myAtt4` globale Attribute, die innerhalb verschiedener Elemente verwendet werden können. Hierdurch wird die bereits für Elemente verwirklichte Mimik der einmaligen Deklaration und anschließenden beliebigen Verwendung auch auf Attribute ausgedehnt. Die Nutzung der so deklarierten Attribute geschieht durch das `ref`-Attribut innerhalb des `Attribute`-Elements des beherbergenden Elements.
`myAtt1` definiert ein typenloses Attribut, dem vorgabegemäß der allgemeinste Typ `anyType` zugeordnet wird. Die Angabe dieses Attributes ist optional (`use="optional"`), was der Vorgabe entspricht.
Der XSD-Standardtyp `decimal` findet zur Definition des Attributs `myAtt2` Verwendung. Die zwingend anzugebenden (`use="required"`) Inhalte dieses Attributs werden durch einen XML-Schema-Parser auf Typkonformität geprüft.
`myAtt3` veranschaulicht die Bildung eines anonymen (inneren) atomaren Typen zur Definition eines Attributs. Der durch Restriktion gebildete neue Datentyp steht ausschließlich innerhalb des Attributs `myAtt3` zur Verfügung. Die Syntax der Datentypspezialisierung entspricht der im vorhergehenden Abschnitt diskutierten. Zudem ist die Verwendung des Attributes innerhalb eines XML-Dokumentes untersagt; ausgedrückt durch die Belegung `use="prohibited"`
Analog der Typisierung eines Elementinhaltes durch einen anwenderdefinierten Typen gestaltet sich das Vorgehen für Attribute. Veranschaulicht wird dies durch die Definition von `myAtt4`. Sie greift auf den eigen-definierten Typen `myType1` zurück.
Dem Attribut `myAtt5` ist zusätzlich zur Benennung, die innerhalb des verwendenden Elementes eindeutig sein sollte, ein Dokument-weiter Schlüssel (`id`) zugeordnet.
Innerhalb des Elements `foo` werden die fünf zuvor definierten Attribute verwendet. Trotz der Reihenfolge der Definitionen im `complexType`-Element verfügen die Attribute im XML-Instanzdokument -- auch bei der Verwendung von XML-Schema -- über keinerlei Reihenfolge (vgl. XML-Spezifikation).
Zusätzlich enthält die Elementdefintion für `foo` mit `myAtt6` ein „lokales“ Attribut. Diese Definitionsvariante entspricht am ehesten der der Document Type Definition, da sie eine Wiederverwendung außerhalb des definierenden Elements ausschließt.

Beispiel 31: Vollständiges XML-Schema der Projektverwaltung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(3)     <xsd:element name="Nachname" type="xsd:string"/>
(4)     <xsd:complexType name="PersonType">
(5)             <xsd:sequence>
(6)                     <xsd:element ref="Vorname" maxOccurs="unbounded"/>
(7)                     <xsd:element ref="Nachname" maxOccurs="unbounded"/>
(8)                     <xsd:element name="Qualifikationsprofil"
type="QualifikationsprofilType" minOccurs="0"/>
(9)             </xsd:sequence>
(10)            <xsd:attribute name="PersID" type="xsd:ID" use="required"/>
(11)            <xsd:attribute name="Gehaltsgruppe" default="1a">
(12)                    <xsd:simpleType>
(13)                            <xsd:restriction base="xsd:NMTOKEN">
(14)                                    <xsd:enumeration value="1"/>
(15)                                    <xsd:enumeration value="1a"/>
(16)                                    <xsd:enumeration value="2"/>
(17)                            </xsd:restriction>
(18)                    </xsd:simpleType>
(19)            </xsd:attribute>
(20)            <xsd:attribute name="mitarbeitInProjekt" type="xsd:IDREFS" use="required"/>
(21)    </xsd:complexType>
(22)    <xsd:complexType name="ProjektType">
(23)            <xsd:attribute name="ID" type="xsd:ID" use="required"/>
(24)            <xsd:attribute name="date" type="xsd:date"/>
(25)            <xsd:attribute name="budget" default="10000.00">
(26)                    <xsd:simpleType>
(27)                            <xsd:restriction base="xsd:double">
(28)                                    <xsd:fractionDigits value="2"/>
(29)                            </xsd:restriction>
(30)                    </xsd:simpleType>
(31)            </xsd:attribute>
(32)            <xsd:attribute name="Projektleiter" type="xsd:IDREF" use="required"/>
(33)            <xsd:attribute name="Mitarbeiter" type="xsd:IDREFS" use="required"/>
(34)    </xsd:complexType>
(35)    <xsd:element name="ProjektVerwaltung">
```

```
(36)            <xsd:complexType>
(37)                    <xsd:sequence>
(38)                            <xsd:element name="Person" type="PersonType"
maxOccurs="unbounded"/>
(39)                            <xsd:element name="Projekt" type="ProjektType"
maxOccurs="unbounded"/>
(40)                    </xsd:sequence>
(41)                    <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(42)            </xsd:complexType>
(43)    </xsd:element>
(44)    <xsd:complexType name="QualifikationsprofilType" mixed="true">
(45)            <xsd:choice minOccurs="0" maxOccurs="unbounded">
(46)                    <xsd:element ref="Qualifikation"/>
(47)                    <xsd:element ref="Leistungsstufe"/>
(48)                    <xsd:any namespace="http://www.w3.org/1999/xhtml"/>
(49)            </xsd:choice>
(50)    </xsd:complexType>
(51)    <xsd:element name="Qualifikation" type="xsd:string"/>
(52)    <xsd:element name="Leistungsstufe" type="xsd:string"/>
(53)    <xsd:element name="Vorname" type="xsd:string"/>
(54)</xsd:schema>
```

Download des Beispiels

Abschließend eine gültige (sowohl *valid* als auch *schema valid*) Dokumentinstanz der Projektverwaltungsstruktur.

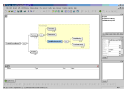
Beispiel 32: Gültiges Projektverwaltungsdokument

```
(1)<?xml version="1.0" encoding="ISO-8859-1"?>
(2)<ProjektVerwaltung
(3)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)     xsi:noNamespaceSchemaLocation="http://www.jeckle.de/vorlesung/xml/examples/
projektverwaltung.xsd">
(5)     <Person PersID="Pers01" mitarbeitInProjekt="Prj01">
(6)             <Vorname>Hans</Vorname>
(7)             <Nachname>Hinterhuber</Nachname>
(8)     </Person>
(9)     <Person PersID="Pers02" mitarbeitInProjekt="Prj02">
(10)            <Vorname>Franz</Vorname>
(11)            <Vorname>Xaver</Vorname>
(12)            <Nachname>Obermüller</Nachname>
(13)            <Qualifikationsprofil>
(14)     IT-Kompetenz verschiedene Betriebssysteme und <Leistungsstufe>professionelle</
Leistungsstufe>
(15)                    <Qualifikation>Programmierung</Qualifikation> verschiedener
Programmiersprachen
(16)     <Qualifikation>Entwickler</Qualifikation> von 1988-1990
(17)     <Qualifikation>Projektleiterfunktion</Qualifikation> von 1990-93 im X42-Projekt
in Abteilung AB&amp;amp;C
(18)   </Qualifikationsprofil>
(19)    </Person>
(20)    <Person PersID="Pers03" mitarbeitInProjekt="Prj02">
(21)            <Vorname>Fritz</Vorname>
(22)            <Nachname>Meier</Nachname>
(23)    </Person>
(24)    <Projekt ID="Prj01" Projektleiter="Pers01" Mitarbeiter="Pers01"/>
(25)    <Projekt ID="Prj02" Projektleiter="Pers02" Mitarbeiter="Pers03"/>
(26)</ProjektVerwaltung>
```

Download des Beispiels

Werkzeuge:
Zwar existiert -- wie für alle XML-Dokumente -- die Möglichkeit, Dokument Typ Definitionen und XML-Schemata „per Hand“ mit einem Texteditor zu erstellen, jedoch ist dieses Vorgehen, insbesondere für umfangreiche XML-Vokabulare, zeitaufwendig und fehlerträchtig. Zusätzlich läßt die rein textuelle Formulierung die entstehenden Schemadokumente schnell unübersichtlich werden.
Inzwischen existieren einige gute DTD- und Schemaeditoren, die zumeist neben visueller Syntaxhervorhebung auch die kontextsensitive Editierung erlauben und so eine wesentliche Erleichterung der Schemaerzeugung bilden. Gleichzeitig bieten die meisten verfügbaren Werkzeuge dieser Klasse auch Möglichkeiten zur Validierung des erzeugten Schemas an.
Ergänzend wird vielfach auch eine graphische Repräsentation der DTD- oder XSD-Struktur angeboten.
Die Abbildungen zeigen Ansichten der Werkzeuge XML Authority bzw. XML Spy





Web-Referenzen 7: Weiterführende Links und Werkzeuge

- XML Schema Part 0: Primer
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes
- XML Schema @ Cover-Pages
- Parsing the Atom -- Diskussion über die Vor- und Nachteile inhärent komplexer atomarer Typen
- Schema-Informationen @ jeckle.de
- XML-Authority (DTD- und XSD-Editor)
- XML Spy (DTD- und XSD-Editor)

2.4 XPath

Zur Extraktion beliebiger Teile eines wohl-geformten XML-Dokuments verabschiedete das W3C 1999 die Sprache *XPath*. Sie bildet eine pfadorientierte *Lokatorsprache*, die das Auffinden von Dokumentteilen (einzelnen Elementen, Attributen, etc.) durch Pfadausdrücke, die sich an der Struktur des XML-Dokuments orientieren, gestattet.
Die Grenze zwischen Lokatorsprache und „echter" Anfragesprache wie SQL sind fließend. Zwei Unterscheidungsmerkmale sollen jedoch hervorgehoben werden: XPath wird im üblichen Anwendungsfall nicht interaktiv oder in eine Programmiersprache als Wirtssprache eingebettet verwendet, sondern wurde (zunächst) nur für die Nutzung in Kombination mit der Transformationssprache *XSLT* und den erweiterten Verweisen der Sprache *XPointer* konzipiert. Zum zweiten fehlt XPath die üblicherweise mit dem reinen Anfrageteil verwobene Manipulationssprache zur Änderung bereits bestehender Daten; XPath ist allein für den lesenden Zugriff auf XML-Dokumente ausgelegt.
Hinweis: XPath unterscheidet XML-üblich zwischen Groß- und Kleinschreibung. Daher sind Element- und Attributnamen unbedingt in der im Dokument gewählten Schreibweise anzugeben.

Lokalisierungspfade:
Lokalisierungspfade dienen der abstrakten Beschreibung einer Menge von Informationsknoten innerhalb eines Dokuments.
Die einfachste Form eines Lokalisierungspfades beschreibt der *Wurzellokalisierungpfad* (*root location path*), ausgedrückt durch „/". Er liefert für jedes XML-Dokument den Wurzelknoten. Dieser ist nicht identisch mit dem Wurzelelement eines XML-Dokuments! Der (unbenannte) Wurzelknoten entspricht dem Document Information Item des Information Sets, während das erste benannte Element des Dokuments durch ein Element Information Item dargestellt wird.

Die Navigation zu den einzelnen Elementknoten, oder Knotenmengen, wird durch einen Pfadausdruck realisiert. Die explizite Variante erlaubt die Angabe aller zu traversierenden Knoten bis hin zu den zu extrahierenden. Hierzu werden die Knoten, von der Wurzel absteigend durch „/"-Symbole separiert, notiert. Wegen der Korrespondenz der voneinander abgetrennten Knotennamen und den Baumstufen, werden diese auch als *Lokalisierungsschritte* bezeichnet. Als weitere sprachliche Analoge spiegelt der XPath-Ausdruck, von links nach rechts gelesen, auch die Schritte -- ausgehend vom Wurzelelement des Dokuments -- zur Lokalisierung der gesuchten Knotenmenge wieder.
Das Beispiel zeigt eine solche Definition am Beispiel der Projektverwaltung.
Anmerkung: Das Resultat ist in XML-Notation dargestellt, obwohl genaugenommen eine Knotenmenge des Information Sets als Resultat zurückgeliefert wird. Die gewählte XML-Darstellung ist hierbei nur eine der möglichen Varianten zur Ergebnispräsentation.

Beispiel 33: XPath-Ausdruck zur Lokalisierung aller Vornamen

XPath-Ausdruck: `/ProjektVerwaltung/Person/Vorname`
Ergebnis:
```
<Vorname>Hans</Vorname>,
<Vorname>Franz</Vorname>,
<Vorname>Xaver</Vorname>,
<Vorname>Fritz</Vorname>
```

Die Einzelknoten werden entsprechend ihrer Auftrittsreihenfolge im Quelldokument (sog. *document order*) zurückgegeben.

Die expliziten Pfadausdrücke lassen sich in beliebiger Länge fortsetzen, jedoch zeigen sie fundamentale Schwächen in Puncto Flexibilität. Wie im Beispiel der XHTML-Verwendung innerhalb eines eigenen XML-Dokuments gesehen, kann Information desselben Typs (d.h. umschlossen durch denselben Tag) verschiedene Elternknoten besitzen. So im Beispiel, dort ist die `Qualifikation` auf derselben Baumstufe sowohl unterhalb des Elternelements `em` als auch `u` anzutreffen.
Als Lösung erlaubt XPath die Nutzung von Platzhaltern statt der expliziten Elementnamen innerhalb eines Lokalisierungsschrittes. In der Folge entstehen freie Lokalisierungsschritte, die alle Kindknoten einer im direkt vorhergehenden Lokalisierungsschritt selektierten Knotenmenge adressieren.

Der nachfolgende XPath-Ausdruck zeigt dies am Beispiel des `Qualifikationsprofils`.

Beispiel 34: Platzhalter in Lokalisierungsschritten

XPath-Ausdruck: `/ProjektVerwaltung/Person/Qualifikationsprofil/*/Qualifikation`
Ergebnis:
```
<Qualifikation>Programmierung</Qualifikation>
<Qualifikation>Projektleiterfunktion</Qualifikation>
```

Der Pfadausdruck liefert die beiden Kindelemente `Qualifikation` -- unabhängig von der Benennung des Elternknotens -- die direkt unterhalb des Knotens `Qualifikationsprofil` angeordnet sind.
Allerdings enthält die Ausgabe nicht alle Knoten des Typs `Qualifikation`. Der gegebene Pfadausdruck gestattet lediglich das Überspringen einer Hierarchieebene. Daher wird der hierarchisch tieferstehende `Qualifikations`-Knoten mit Inhalt `Entwickler` nicht lokalisiert. Die (zunächst naheliegende) Lösung den Pfadausdruck zu `/ProjektVerwaltung/Person/Qualifikationsprofil/*/*/Qualifikation` zu erweitern liefert nicht das gewünschte Resultat aller `Qualifikations`-Knoten, sondern ausschließlich den zuvor nicht lokalisierbaren, da der modifizierte Ausdruck nun zwingend zwei freie Lokalisierungsschritte vorsieht.
Zur Variierung der Tiefe der freien Schritte sieht XPath die Schreibweise „//“ vor. Sie erlaubt die Lokalisierung der Kindknoten auf einer beliebigen Hierarchiestufe.

Definition 11: Lokalisierungsschritt

Ein Lokalisierungsschritt setzt sich aus dem Namen der Achse gefolgt von zwei Doppelpunkten und einem *Knotentest*, optional ergänzt um ein auszuwertendes Prädikat, zusammen.
Wird keine Achse spezifiziert, so gilt vorgabegemäß die Achse `child`.
Ein *Knotentest* ist syntaktisch ein QName, der genau dann erfüllt ist, wenn der Knotenname mit dem Namen des Knotentests übereinstimmt.
Das *Prädikat* filtert die Ergebnismenge hinsichtlich verschiedener Charakteristika wie Existenz von Kindknoten oder Attributen, Position in der Ergebnismenge, etc.

Das Beispiel zeigt die korrekte XPath-Formulierung zur Lokation aller `Qualifikations`-Knoten:

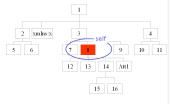
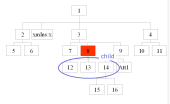
Beispiel 35: Hierarchieunabhängige Knoten-Lokalisierung

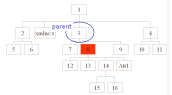
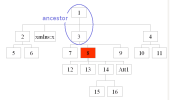
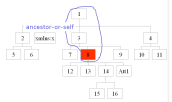
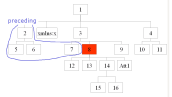
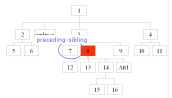
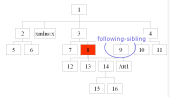
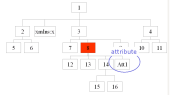
XPath-Ausdruck: `/ProjektVerwaltung/Person/Qualifikationsprofil//Qualifikation`
Ergebnis:
```
<Qualifikation>Programmierung</Qualifikation>

<Qualifikation>Entwickler</Qualifikation>

<Qualifikation>Projektleiterfunktion</Qualifikation>
```

Durch die abkürzende Schreibweise „//“ entsteht ein Muster zur Selektion aller nachfolgenden Knoten. In Verallgemeinerung dieses Konzepts bietet XPath sog. *Achsen* an, um relativ zum aktuellen Knoten beliebige Teilbäume zu lokalisieren.
Die Abbildung zeigt die verschiedenen durch Achsen zugänglichen Knotenmengen relativ zum rot hervorgehobenen aktuellen Knoten.

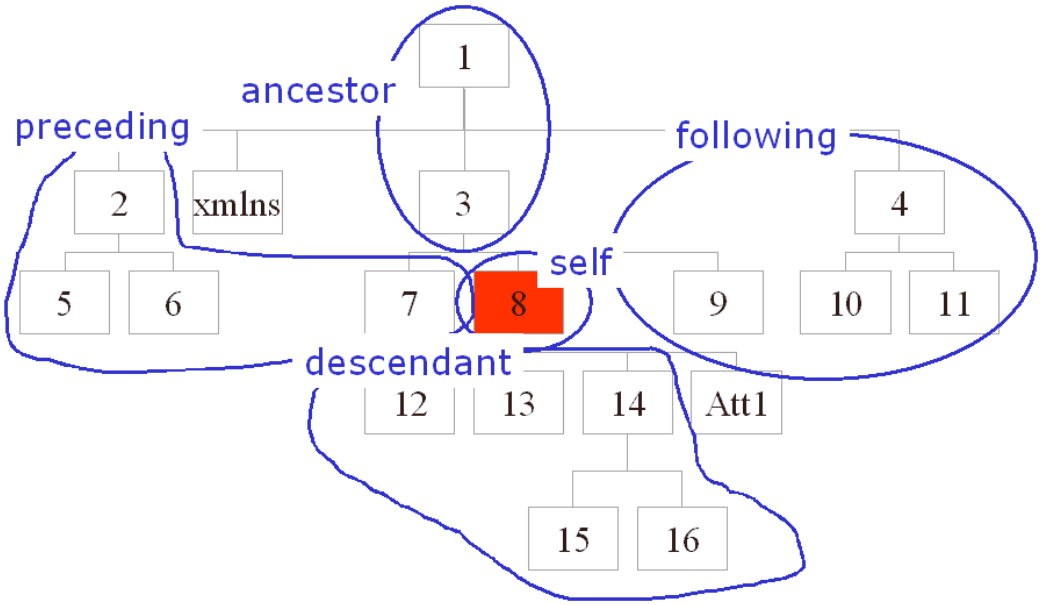
Download der XML-Datei mit dem Beispiel der Graphik

Tabelle 18: XPath-Achsen und ihre Bedeutung

Achse	Semantik	Im Beispiel selektierte Knoten	Graphik
`self`	Lokalisiert den aktuellen Knoten Als abkürzende Schreibweise kann der Punkt „.“ verwendet werden.	**XPath-Ausdruck:** `/node1/node3/node8/self::node8` **Ergebnisknotenmenge:** {8}	
`child`	Lokalisiert die (direkten) Kindknoten des aktuellen Knotens	**XPath-Ausdruck:** `/node1/node3/node8/child::*` **Ergebnisknotenmenge:** {12, 13, 14}	
`descendant`	Lokalisiert transitiv alle Kindknoten des aktuellen Knotens, außer Attribut- und Namensraumknoten	**XPath-Ausdruck:** `/node1/node3/node8/descendant::*` **Ergebnisknotenmenge:** {12, 13, 14, 15, 16}	
`descendant-or-self`	Lokalisiert transitiv alle Kindknoten des aktuellen Knotens (außer Attribut- und Namensraumknoten), sowie den Knoten selbst	**XPath-Ausdruck:** `/node1/node3/node8/descendant-or-self::*` **Ergebnisknotenmenge:** {8, 12, 13, 14, 15, 16}	

parent	Lokalisiert den Elternknoten des aktuellen Knotes, falls existent	**XPath-Ausdruck:** /node1/node3/node8/ parent::* **Ergebnisknotenmenge:** {3}	
ancestor	Lokalisiert transitiv alle Elternknoten des aktuellen Knotes. Die ancestor-Achse enthält daher immer den Wurzelknoten, außer der aktuelle Knoten ist es selbst; in diesem Falle liefert die Achse die leere Menge	**XPath-Ausdruck:** /node1/node3/node8/ ancestor::* **Ergebnisknotenmenge:** {1, 3}	
ancestor-or-self	Lokalisiert transitiv alle Elternknoten des aktuellen Knotes, sowie den aktuellen Knoten. Diese Achse enthält immer den Wurzelknoten des Dokuments.	**XPath-Ausdruck:** /node1/node3/node8/ ancestor-or-self::* **Ergebnisknotenmenge:** {1, 3, 8}	
preceding	Lokalisiert alle dem aktuellen Knoten vorausgehenden Knoten, ohne seine Vorfahren sowie Attribut- und Namensraumknoten	**XPath-Ausdruck:** /node1/node3/node8/ preceding::* **Ergebnisknotenmenge:** {2, 5, 6, 7}	
preceding-sibling	Lokalisiert die im Dokument vor dem aktuellen Knoten auftretenden Geschwisterknoten	**XPath-Ausdruck:** /node1/node3/node8/ preceding-sibling::* **Ergebnisknotenmenge:** {7}	
following	Lokalisiert alle dem aktuellen Knoten nachfolgenden Knoten ohne dessen Kind-, Attribut und Namensraumknoten	**XPath-Ausdruck:** /node1/node3/node8/ following::* **Ergebnisknotenmenge:** {9, 4, 10, 11}	
following-sibling	Lokalisiert alle „Geschwister“ des aktuellen Knotens, d. h. Knoten auf derselben Hierarchieebene.	**XPath-Ausdruck:** /node1/node3/node8/ following-sibling::* **Ergebnisknotenmenge:** {9}	
attribute	Lokalisiert Attribut(e) eines Knotens	**XPath-Ausdruck:** /node1/node3/node8/ attribute::* **Ergebnisknotenmenge:** {Att1}	
namespace	Lokalisiert Namensraum-Attribut eines Knotens	**XPath-Ausdruck:** /node1/node3/node8/ namespace::* **Ergebnisknotenmenge:** {xmlns:xml="http:// www.w3.org/XML/1998/ namespace", xmlns:x="namespace: www.jeckle.de/ vorlesung/xml"}	

Anmerkung:
Die Achsen ancestor, descendant, following, preceding und self partitionieren ein Dokument (unter Auslassung der Attribut- und Namensraumknoten): sie überschneiden sich nicht und enthalten alle Elementknoten des Dokuments.



Filterung durch Prädikate:
Ein -- durch eckige Klammern abgegrenztes -- Prädikat kann innerhalb jedes Lokalisierungsschrittes eines XPath-Ausdrucks angegeben werden. Fehlt es, wird die bisher ermittelte Knotenmenge nicht modifiziert. Das Prädikat kann selbst ein gültiger XPath-Ausdruck sein.
Das prinzipielle Vorgehen kann folgendermaßen beschrieben werden:
Beginnend von links nach rechts für jeden Lokalisierungsschritt: (1) Ermittlung der zur Anfrage passenden Knotenmenge
(2) Reduzierung der Ergebnismenge um diejenigen Knoten, für die das Prädikat `false` liefert.
Befinden sich rechts vom aktuell bearbeiteten Lokalisierungsschritt weitere Ausdrücke, so wird die Resultatmenge als Eingabe eines weiteren Schritts (1) übergeben.

Beispiel 36: Selektion unter Anwendung eines Prädikats

XPath-Ausdruck: `//Person[Qualifikationsprofil]/Nachname`
Ergebnis:
`<Nachname>Obermüller</Nachname>`

Der Ausdruck selektiert an beliebiger Stelle des Dokuments („//“) alle Knoten des Typs `Person`. Die Knotenmenge wird um diejenigen `Personen` vermindert, zu denen kein `Qualifikationsprofil` angelegt ist. D.h. Es werden nur diejenigen Knoten selektiert, die über einen Kindknoten des Typs `Qualifikationsprofil` verfügen. Von dieser Knotenmenge (des Typs `Person`!) werden anschließend im zweiten Lokalisierungsschritt die Kindknoten des Typs `Nachname` selektiert.
Mithin liefert der XPath-Ausdruck alle `Nachnamen` von `Personen`, zu denen ein `Qualifikationsprofil` abgelegt ist.
Anmerkung: Das Beispiel nutzt im Prädikat die abkürzende Schreibweise zur Angabe der Vorgabeachse `child`. Die ausführliche Schreibweise -- mit unveränderter Semantik -- des XPath-Ausdruckes lautet daher: `//Person[child::Qualifikationsprofil]/Nachname`

Durch die zusätzliche Definition eines Prädikats für den zweiten Lokalisierungsschritt kann eine weitere Filterung der Ergebnismenge realisiert werden. Zusätzlich können innerhalb eines Prädikats neben XPath-Ausdrücken auch einige vordefinierte Funktionen verwendet werden.
Das Beispiel zeigt die Selektion der `Vornamen` als Kind eines `Personen`-Knotens (Test der Elternschaft durch erstes Prädikat), wenn dieser mit „O“ beginnt (Test durch *starts-with*-Funktion innerhalb des zweiten Prädikats). Die Struktur der Eingabedatei zwingt zusätzlich zur Anwendung der `following`-Achse, da Knoten des Typs `Nachname` in der Dokumentreihenfolge nach Knoten des Types `Vornamen` auftreten.

Beispiel 37: Schrittweise Berechnung einer Selektion unter Verwendung mehrerer Prädikate

XPath-Ausdruck: `//Person[parent::ProjektVerwaltung]/Vorname [starts-with(following::Nachname,'O')]`

Ausgewerteter XPath:`//Person`
Ergebnis:
```
<Person PersID="Pers01" mitarbeitInProjekt="Prj01"> ... </Person>
<Person PersID="Pers02" mitarbeitInProjekt="Prj02"> ... </Person>
<Person PersID="Pers03" mitarbeitInProjekt="Prj02"> ... </Person>
```

Ausgewerteter XPath:`//Person[parent::ProjektVerwaltung]`
Ergebnis:
```
<Person PersID="Pers01" mitarbeitInProjekt="Prj01"> ... </Person>
<Person PersID="Pers02" mitarbeitInProjekt="Prj02"> ... </Person>
<Person PersID="Pers03" mitarbeitInProjekt="Prj02"> ... </Person>
```

Ausgewerteter XPath:`//Person[parent::ProjektVerwaltung]/Vorname`
Ergebnis:
```
<Vorname>Hans</Vorname>
<Vorname>Franz</Vorname>
<Vorname>Xaver</Vorname>
<Vorname>Fritz</Vorname>
```

Ausgewerteter XPath:`//Person[parent::ProjektVerwaltung]/Vorname[following::Nachname]`
Ergebnis:
```
<Vorname>Hans</Vorname>
<Vorname>Franz</Vorname>
<Vorname>Xaver</Vorname>
<Vorname>Fritz</Vorname>
```

Ausgewerteter XPath:
`//Person[parent::ProjektVerwaltung]/Vorname[starts-with (following::Nachname,'O')]`
Ergebnis:
```
<Vorname>Franz</Vorname>
<Vorname>Xaver</Vorname>
```

Die durch die XPath-Spezifikation vordefinierten Funktionen lauten in der Übersicht:

Tabelle 19: XPath-Funktionen für Knotenmengen (node-sets)

Funktionsprototyp	Funktionalität
number last()	Liefert die Größe der aktuellen Knotenmenge; damit den Index des letzten Elements
number position()	Liefert die Position des aktuellen Knotens innerhalb der Knotenmenge. Die erste Knoten trägt die Positionsnummer 1.
number count(*node-set*)	Liefert Elementzahl der übergebenen Knotenmenge
node-set id(*object*)	Liefert denjenigen Knoten, dessen ID-typisiertes Attribut den Argumentwert aufweist. *Anmerkung*: Zur Nutzung dieser Funktion muß zwingend eine Dokument-Grammatik (DTD oder Schema) zum Eingangsdokument vorliegen.
string local-name (*node-set*?)	Liefert den local name (oder die Menge der Namen) der übergebenen Knotenmenge. Wird keine Knotenmenge übergeben, dann wird der aktuelle Knoten als Argument genutzt.
string namespace-uri (*node-set*?)	Liefert die Namensraum-URI der übergebenen Knotenmenge. Wird keine Knotenmenge übergeben, dann wird der aktuelle Knoten als Argument genutzt. *Anmerkung*: Handelt es sich nicht um einen Element- oder Attributknoten, so ist die retournierte Zeichenkette leer.
string name (*node-set*?)	Liefert die QName(n) (=qualifizierte(r) Name(n) aus Namensraumkürzel und local name) der übergebenen Knotenmenge, oder des aktuellen Knotens bei leerer Knotenmenge. *Anmerkung*: Nur für Element- und Attributknoten liefert name andere Resultate als local-name.

Tabelle 20: XPath-Funktionen für Zeichenketten

Funktionsprototyp	Funktionalität
string string (*object*)?	Liefert Zeichenkettenrepräsentation einer Knotenmenge. Dabei wird der Zeichenkettenwert des ersten Knotens in der Dokumentreihenfolge zurückgegeben, andernfalls die leere Zeichenkette.
string concat (*string*, *string*, *string**)	Verkettet mindestens zwei Zeichenketten.
boolean starts-with (*string1*, *string2*)	Liefert true falls string1 das zweite Argument string2 als Präfix enthält; andernfalls false
boolean contains (*string1*, *string2*)	Liefert true falls string1 die Zeichenkette aus string2 enthält; andernfalls false.
string substring-before (*string1*, *string2*)	Liefert denjenigen Teil der Zeichenkette *string1*, der sich vor dem ersten Auftreten der Zeichenkette *string2* befindet.

string substring-after (*string1*, *string2*)	Liefert denjenigen Teil der Zeichenkette *string1*, der sich nach dem ersten Auftreten der Zeichenkette *string2* befindet.
string substring (*string*, *number1*, *number2*?)	Liefert eine Zeichenkette der Länge *number2* aus *string*, beginnend mit der Position *number1*. Fehlt das dritte Argument, so wird der Teilstring bis zum Ende der Zeichenkette *string* zurückgegeben. *Anmerkung*: Das erste Zeichen trägt die Indexnummer 1, nicht 0 wie in Java und C üblich.
number string-length(*string*?)	Liefert die Länge der übergebenen Zeichenkette. Wird kein Argument übergeben, so wird die Länge des zuvor in eine Zeichenkette konvertierten aktuellen Knotens zurückgegeben.
string normalize-space (*string*?)	Liefert die übergebene Zeichenkette unter Entfernung führender, schließender und mehrfacher Leerzeichen zurück. Ferner werden noch evtl. in der Argumentzeichenkette enthaltenen Entitätsreferenzen aufgelöst. *Anmerkung*: Der Normalisierungsvorgang entspricht damit der Attributwertenormalisierung nach Abschnitt 3.3.3 der XML-Spezifikation.
string translate (*string1*, *string2*, *string3*)	Liefert die Zeichenkette *string1* wobei jedes Zeichen aus *string2* durch das Zeichen an derselben Position aus *string3* ersetzt wurde.

Tabelle 21: Boole'sche XPath-Funktionen

Funktionsprototyp	Funktionalität
boolean boolean (*object*)	Liefert die Boole'sche Repräsentation des übergebenen Arguments. Hierbei gilt: •Eine Zahl wird genau dann nach true konvertiert, wenn sie weder Null (unbeachtlich ihres Vorzeichens) noch eine nicht darstellbare Zahl (NaN) ist. •Eine Knotenmenge ergibt true, wenn sie nicht leer ist. •Eine Zeichenkette ergibt true, wenn sie nicht leer (d.h. Länge größer Null) ist. •Die Konvertierung anderer Typen ist typabhängig, und nicht durch den Standard festgelegt
boolean not (*boolean*)	Negiert das übergebene Argument
boolean true()	Liefert statisch den Wert true
boolean false()	Liefert statisch den Wert false
boolean lang (*string*)	Liefert true wenn der aktuelle Knoten ein xml:lang-Attribut gemäß der als Argument übergebenen Sprache besitzt

Tabelle 22: Zahlenorientierte XPath-Funktionen

Funktionsprototyp	Funktionalität
number number (*object*?)	Konvertiert ein Objekt in eine Zahl gemäß folgender Regeln: •Eine Zeichenkette wird in eine Fließkommazahl gemäß IEEE 754 konvertiert, wenn sie aus einem optionalen Leerzeichen, gefolgt durch ein optionales Minuszeichen, gefolgt von einem optionalen Leerzeichen und einer Ziffernfolge besteht. •Der Boole'sche Wert true wird zu 1, der Wert false zu 0 konvertiert. •Eine Knotenmenge wird zunächst in eine Zeichenkette übersetzt, und dann gemäß der oben definierten Regeln umgesetzt. •Die Konvertierung anderer Typen erfolgt typabhängig, und ist nicht durch den Standard geregelt. Wird kein Argument übergeben, so wird stattdessen der aktuelle Knoten als einziges Element einer Knotenmenge interpretiert.
number sum (*node-set*)	Liefert die Summe aller Elemente der übergebenen Knotenmenge, die zuvor in eine Zahl konvertiert werden.
number floor (*number*)	Liefert die größte ganze Zahl, die nicht größer als das Argument ist. *Anmerkung*: Entspricht dem Abschneiden beliebiger Nachkommastellen
number ceiling (*number1*)	Liefert die kleinste ganze Zahl, die nicht kleiner als das Argument ist. *Anmerkung*: Entspricht floor(number1+0.999...)
number round (*number*)	Liefert das Argument auf die nächste ganze Zahl gerundet. Gibt es zwei solche -- wie bei Nachkommastelle gleich 0.5 immer der Fall -- so wird die größere zurückgeliefert.

Für mathematische Berechnungen auf zahlenartigen Knoten stehen folgende Operatoren zur Verfügung.

Tabelle 23: Mathematische Operatoren

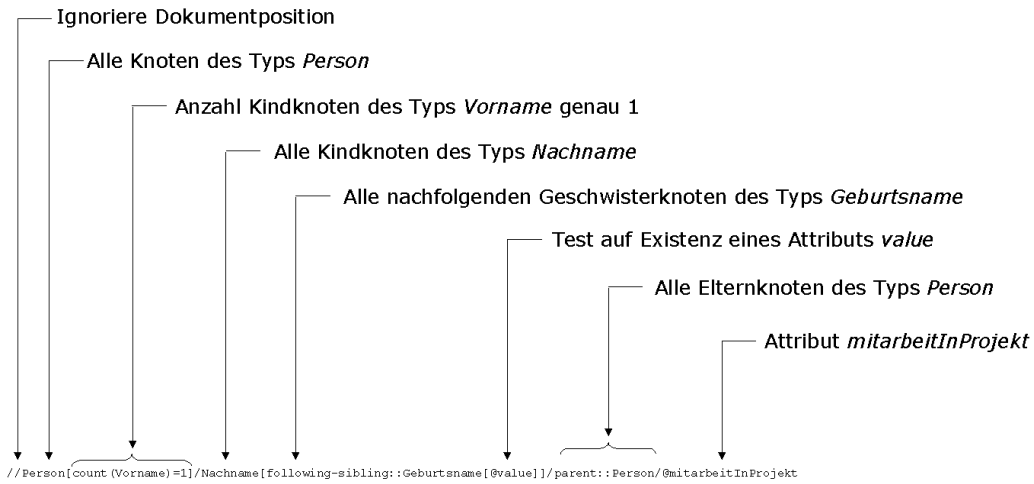
Operator	Funktionalität
+	Addition
-	Subtraktion als zweistelliger Operator. Der einstellige Operator - ist nicht spezifiziert, er liefert üblicherweise die negative Zahlendarstellung.
*	Multiplikation. Außer wenn innerhalb von XPath-Ausdrücken als Knotentest eingesetzt.
div	Division Achtung: Das Symbol / dient ausschließlich als Trennzeichen zur Separierung von Lokalisierungspfaden!
mod	Rest einer ganzzahligen Division

Ein umfangreiches Beispiel: Für das nachfolgende Beispiel wird das Projektverwaltungsdokument erweitert zu:

Beispiel 38: Erweiterte Projektverwaltung

```
(1)<?xml version="1.0" encoding="ISO-8859-15"?>
(2)<ProjektVerwaltung xmlns:xhtml="http://www.w3.org/1999/xhtml" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="L:\vorlesung\xml\examples
\projektverwaltung3.xsd">
(3)     <Person PersID="Pers01" mitarbeitInProjekt="Prj01">
(4)             <Vorname>Hans</Vorname>
(5)             <Nachname>Hinterhuber</Nachname>
(6)     </Person>
(7)     <Person PersID="Pers02" mitarbeitInProjekt="Prj02">
(8)             <Vorname>Franz</Vorname>
(9)             <Vorname>Xaver</Vorname>
(10)            <Nachname>Obermüller</Nachname>
(11)            <Qualifikationsprofil>
(12)                    <xhtml:u>IT-Kompetenz</xhtml:u>
(13)                    <xhtml:em>verschiedene</xhtml:em> Betriebssysteme und
(14)        <Leistungsstufe>professionelle</Leistungsstufe>
(15)                    <xhtml:em>
(16)                            <Qualifikation>Programmierung</Qualifikation>
(17)                    </xhtml:em>
(18)        verschiedener Programmiersprachen
(19)        <xhtml:em>
(20)                            <xhtml:u>
(21)                                    <Qualifikation>Entwickler</Qualifikation>
(22)                            </xhtml:u>
(23)                    </xhtml:em> von 1988-1990
(24)        <xhtml:u>
(25)                            <Qualifikation>Projektleiterfunktion</Qualifikation>
(26)                    </xhtml:u>
(27)        von <xhtml:b>1990-93</xhtml:b> im X42-Projekt in Abteilung AB&amp;amp;C
(28)      </Qualifikationsprofil>
(29)    </Person>
(30)    <Person PersID="Pers03" mitarbeitInProjekt="Prj02">
(31)            <Vorname>Fritz</Vorname>
(32)            <Nachname>Meier</Nachname>
(33)            <Geburtsname value="Huber"/>
(34)    </Person>
(35)    <Projekt ID="Prj01" Projektleiter="Pers01" Mitarbeiter="Pers01"/>
(36)    <Projekt ID="Prj02" Projektleiter="Pers02" Mitarbeiter="Pers03"/>
(37)</ProjektVerwaltung>
```

Download des Beispiels



Der XPath-Ausdruck der Abbildung 20 lokalisiert den Attributknoten des Inhalts `Prj02`.

Übung 2: Einige Übungen

Welches Ergebnis liefern folgende XPath-Ausdrücke?
(a) `//Person[//child::Qualifikationsprofil]/Nachname`
(b) `//Person[parent::ProjektVerwaltung]/Vorname[following-sibling::Vorname]`
(c) `/ProjektVerwaltung/Person[attribute::PersID='Pers01']//Nachname`

Wie muß ein XPath-Ausdruck lauten, um folgendes zu selektieren?
(d) Selektion aller Personen mit Nachnamen „Obermüller“.
(e) Selektion aller Nachnamen von Personen die über mehr als eine Qualifikation verfügen.
(f) Selektion der Nachnamen aller Projektleiter.

Anwendungsbeispiel: Integritätsbedingungen in XML-Schema

Über die Möglichkeiten der Datentypen hinausgehend bietet XML-Schema das Element `unique` zur Definition eindeutiger Wertbelegungen an. Hierbei wird auf die Lokatorsprache XPath zurückgegriffen um die abzuprüfenden Knoten innerhalb des Dokuments zu bezeichnen.

Die Syntax verwendet XPath-Ausdrücke eingeschränkter Mächtigkeit sowohl zur Festlegung des der Knotenmenge, auf die sich die Einschränkung bezieht (`selector`), als auch zur Angabe der eingeschränkten Knoten (`field`) selbst.

```
(1)<xsd:unique name="aName">
(2)     <xsd:selector xpath="aValidXPath"/>
(3)     <xsd:field xpath="aFieldStatement"/>
(4)     ...
(5)</xsd:unique>
```

Die Mächtigkeit der XPath-Ausdrücke ist dahingehend eingeschränkt, daß für das `selector`-Element ausschließlich Ausdrücke erlaubt sind, die Kindelemente des Knotens liefern, in dessen Kontext die durch `unique` formulierte Einschränkung angegeben wird. Als Konsequenz ist die Nutzung der verfügbaren XPath-Achsen auf diejenigen beschränkt, die Element-Knotenmengen zurückliefern.
Die Lokationsausdrücke in den -- möglicherweise mehrfach auftretenden -- `field`-Elementen werden relativ zum Pfad des `selector`-Knotens interpretiert. Hintereinandergesetzt muß der Pfad eines `selector`-Elements, gefolgt von einem Pfad eines `field`-Elements, einen gültigen Lokationsausdruck ergeben, der genau einen Knoten oder genau ein Attribut in der Ergebnismenge liefert. Sind mehrere `field`-Elemente zu einem `selector`-Element gegeben, so werden diese als durch logisches *und* verknüpft interpretiert. Mithin entspricht diese Semantik einem *concatenated primary key* aus den relationalen Datenbanken.

Das Beispiel zeigt die Nutzung des `unique`-Konstrukts zur Angabe der Eindeutigkeitsbedingung für das Attribut `PersID` des Elements `Person`.
Zunächst selektiert der Pfad `/Person` alle Knoten des gleichnamigen Typs; durch das `field`-Element wird die Eindeutigkeitsbedingung auf alle Attribut-Kindnoten des Typs `PersID` der Knoten in der selektierten Knotenmenge angewendet.
Die Semantik ist damit zur bisherigen `ID`-Typisierung identisch.

Beispiel 39: Unique-Einschränkung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema
(3)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)<xsd:element name="ProjektVerwaltung">
(7)     <xsd:complexType>
(8)             <xsd:sequence>
(9)                     <xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/
>
(10)                    <xsd:element name="Projekt" type="ProjektType"
maxOccurs="unbounded"/>
(11)            </xsd:sequence>
(12)            <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(13)    </xsd:complexType>
(14)    <xsd:unique name="uniquenessPersID">
(15)            <xsd:selector xpath="Person"/>
(16)            <xsd:field xpath="@PersID"/>
(17)    </xsd:unique>
(18)</xsd:element>
(19)
(20)<xsd:complexType name="PersonType">
(21)    <xsd:attribute name="PersID" type="xsd:token"/>
(22)</xsd:complexType>
(23)
(24)<xsd:complexType name="ProjektType"/>
(25)
(26)</xsd:schema>
```

Download des Beispiels

Das nächste Beispiel zeigt die Verwendung mehrerer `field`-Elemente zur Realisierung zusammengesetzter Schlüssel.
Hierzu wird die Kombination aus dem Inhalt des `Nachnamen`- und des `Vornamen`-Elements zusammen als eindeutig deklariert.
Überdies zeigt das Beispiel die Anwendung des Schlüsselmechanismus auf Elemente ohne Änderung der Basissyntax, abgesehen von der geänderten XPath-Achse.

Beispiel 40: Zusammengesetzter Schlüssel innerhalb eines unique-Elements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema
(3)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)<xsd:element name="ProjektVerwaltung">
(7)     <xsd:complexType>
(8)             <xsd:sequence>
(9)                     <xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/
>
(10)                    <xsd:element name="Projekt" type="ProjektType"
maxOccurs="unbounded"/>
(11)            </xsd:sequence>
(12)            <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(13)    </xsd:complexType>
(14)    <xsd:unique name="uniquenessPersID">
(15)            <xsd:selector xpath="Person"/>
(16)            <xsd:field xpath="Vorname"/>
(17)            <xsd:field xpath="Nachname"/>
(18)    </xsd:unique>
(19)</xsd:element>
(20)
(21)<xsd:complexType name="PersonType">
(22)    <xsd:sequence>
(23)            <xsd:element name="Vorname" type="xsd:token" minOccurs="1"
maxOccurs="unbounded"/>
(24)            <xsd:element name="Nachname" type="xsd:token" maxOccurs="1"/>
(25)    </xsd:sequence>
(26)</xsd:complexType>
(27)
(28)<xsd:complexType name="ProjektType"/>
(29)
(30)</xsd:schema>
```

Download des Beispiels

Zur Realisierung von wertdefinierenden Schlüsselbeziehungen bietet XML-Schema die Elemente key und keyref an. Sie werden verwendet um sicherzustellen, daß ein Element oder Attribut nur einen Wert annehmen darf, der bereits an anderer Stelle im Instanzdokument auftritt.
Hierzu lokalisiert key auf der Basis eines XPath-Ausdruckes eine Referenzmenge, während keyref diejenige Knotenmenge lokalisiert, in der ausschließlich Elemente der Referenzmenge enthalten sein dürfen.
Das Beispiel zeigt die Anwendung auf das Element ProjektVerwaltung. Der mit *projectKey* benannte Schlüssel definiert die Referenzmenge als das Ergebnis der Anfrage Projekt/@ID, worauf die *projectReference* Bezug nimmt.

Beispiel 41: Schlüsselbasierte Referenzierung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema
(3)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)     <xsd:element name="ProjektVerwaltung">
(7)             <xsd:complexType>
(8)                     <xsd:sequence>
(9)                             <xsd:element name="Person" type="PersonType"
maxOccurs="unbounded"/>
(10)                            <xsd:element name="Projekt" type="ProjektType"
maxOccurs="unbounded"/>
(11)                    </xsd:sequence>
(12)                    <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(13)            </xsd:complexType>
(14)
(15)            <xsd:key name="projectKey">
(16)                    <xsd:selector xpath="Projekt"/>
(17)                    <xsd:field xpath="@ID"/>
(18)            </xsd:key>
(19)            <xsd:keyref name="projectReference" refer="projectKey">
(20)                    <xsd:selector xpath="Person"/>
(21)                    <xsd:field xpath="@mitarbeitInProjekt"/>
(22)            </xsd:keyref>
(23)    </xsd:element>
(24)
(25)    <xsd:complexType name="PersonType">
(26)            <xsd:attribute name="mitarbeitInProjekt" type="xsd:token"/>
(27)    </xsd:complexType>
(28)    <xsd:complexType name="ProjektType">
(29)            <xsd:attribute name="ID" type="xsd:token"/>
(30)    </xsd:complexType>
(31)</xsd:schema>
```

Download des Beispiels

Web-Referenzen 8: Weiterführende Links

- XPath Spezifikation
- Deutsche Übersetzung der XPath-Spezifikation
- XPath Visualisierer (Java-basiert)
- Visual XPath (.NET Windows-Applikation)

Originalbezugsquelle

- XPath Explorer (Java-basiert)

Originalbezugsquelle

- Online Experimentieren mit XPath

3 Datenbankzugriff

3.1 Java Database Connectivity (JDBC)

Motivation

Häufig besteht der Wunsch oder die Notwendigkeit, auf bereits vorliegende Datenbestände, die durch ein

Datenbankmanagementsystem (DBMS) verwaltet werden, in einer Applikationsprogrammiersprache zuzugreifen. Dabei soll die Anbindung der benötigten Datenquelle nicht problemspezifisch wieder und wieder neu entwickelt werden, sondern sollte sich auf ähnliche Datenanbindungsprobleme übertragen lassen.
Vor diesem Hintergrund liegt es nahe, sich an den Typen der verfügbaren und kommerziell bedeutsamen DBMS zu orientieren und herstellerspezifische Entwicklungen außer Acht zu lassen. Gleichzeitig offenbaren sich hierbei Standardisierungsbemühungen wie die Sprache *SQL* zum Zugriff auf relationale DBMS als lohnenswerter Ansatz der Etablierung einer generischen und übertragbaren Schnittstelle.

Die Idee zur Schaffung einer solchen generischen Schnittstelle für den Zugriff auf relationale DBMS geht zurück auf eine Initiative der *SQL Access Group*, welche später in der Vereinigung mit der *X/Open* Group aufging, die zwischenzeitlich in *Open Group* umbenannt wurde. Das dort konzipierte programmiersprachenunabhängige *SQL Call Level Interface* (SQL/CLI) konnte sich dank der Umsetzung unter dem Namen Open Database Connectivity (ODBC) durch die Firma Microsoft und die parallel erfolgte internationale Normierung unter dem Titel *SQL/CLI* breit am Markt etablieren.

Die für die Programmiersprache Java adaptierte Variante des Zugriffs auf relationale DBMS wird durch SUN Microsystems unter dem Namen *Java Database Connectivity* (JDBC) propagiert und stellt eine auf ODBC konzeptionell aufbauende und auf die spezifischen Bedürfnisse dieser Applikationsprogrammiersprache optimierte Untermenge des SQL/CLI-Standards dar.

Konzept und Grundidee

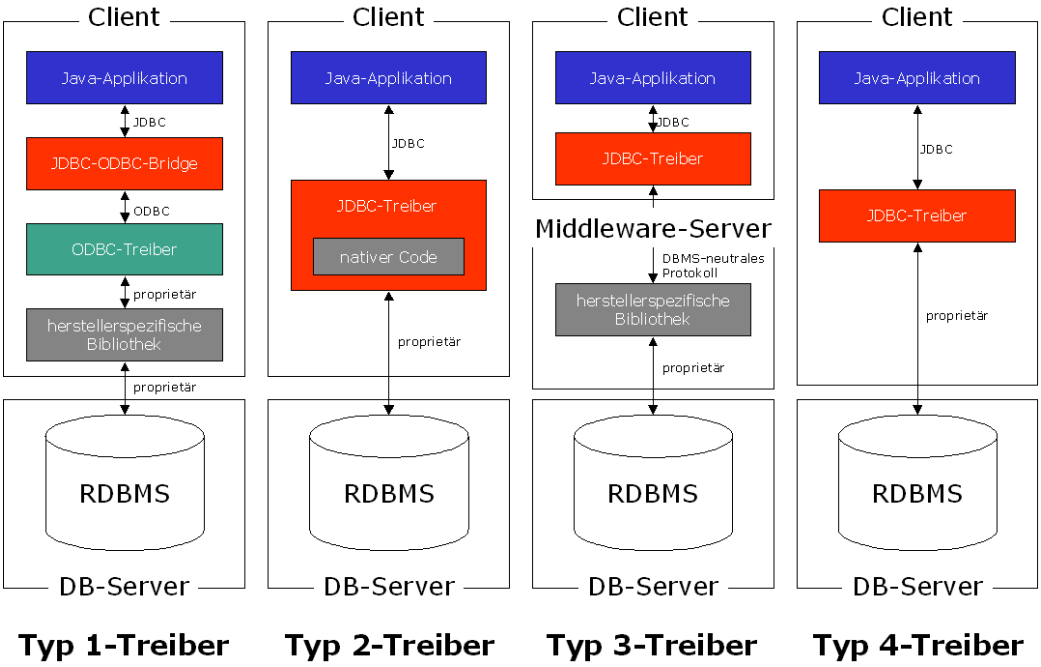
Von den Vorgängeransätzen übernommene Grundidee der Schnittstelle ist es den physischen Zugriff auf das Datenbankmanagementsystem durch eine von der Applikation spearierte wiederverwendbare Softwarekomponente, den sog. *JDBC-Treiber*, abzuwickeln.

Dieser Treiber vermittelt zwischen der Javaapplikation und dem verwendeten DBMS. Hierbei muß für jedes DBMS ein auf es abgestimmter JDBC-Treiber verwendet werden, da lediglich die Schnittstelle zur Applikation, nicht jedoch die zum DBMS, standardisiert ist.

Diesem Treiber obliegt die Abwicklung der gesamten Kommunikationsvorgänge mit dem DBMS. Er setzt jedoch selbst keine datenbankspezifischen Funktionalitäten, wie Syntax- oder Plausibilitätsprüfungen der übermittelten Kommandos um. Etwaige Fehlerprüfungen können, ebenso wie Anfrageoptimierungen, daher erst seitens des DBMS vorgenommen werden.
Der Vorteil dieses Vorgehens liegt in der Generizität des JDBC-Treibers. Er kann ohne aufwendige Logikanteile als reine uninterpretierende Vermittlungsschicht zwischen Applikation und DBMS umgesetzt werden, wodurch schlanke Implementierungen ermöglicht werden.

Die JDBC-Spezifikation detailliert den Treiberbegriff zusätzlich hinsichtlich der gewählten technischen Umsetzung aus. So werden die vier in Abbildung 3 dargestellten Treibertypen gemäß ihrer Charakteristika beschrieben und unterschieden.

Abbildung 3: *JDBC-Treibertypen*



(click on image to enlarge!)

Die historisch älteste Variante bildet der **Typ 1 Treiber**. Strenggenommen verkörpert er selbst keinen Datenbanktreiber, sondern lediglich eine Umsetzungsschicht die einem existierenden ODBC-Treiber vorgeschaltet wird.
Die Abbildung belegt diesen Treibertyp daher mit dem Begriff *JDBC-ODBC-Bridge*, da er lediglich den

Brückenschlag zwischen den beiden Standards vornimmt und sich in der konkreten Anwendung auf die Umsetzung zwischen den beiden Protokollen beschränkt, ohne realen Zugriff auf die Datenbank zu erhalten.
Dieser ist dem ODBC-Treiber vorbehalten, der im allgemeinen Falle mit einer weiteren Umsetzungsstufe kommuniziert, welche die generischen ODBC-Aufrufe in konkrete DBMS-spezifische wandelt.
Während sowohl der JDBC-ODBC-Brückentreiber als auch der ODBC-Treiber selbst für verschiedene DBMS verwendet werden können, muß für jedes konkrete DBMS eine herstellerspezifische, d.h. an das verwendete DBMS angepaßte, Bibliothek vorliegen.

Für den Fall eines **Typ 2 Treiber**s entfällt diese durch ODBC geschaffene zusätzliche Indirektionsstufe zugunsten der Adaption der Konversionskomponente, welcher die Wandlung der Aufrufe in das DBMS-native Protokoll obliegt, an das JDBC-Protokoll und ihrer Integration in den JDBC-Treiber selbst.
Die Natur der Kommunikation des Java-Anteils des Treibers mit den Nativen ist im Rahmen der durch die JDBC-Spezifikation gegebenen Definition nicht festgelegt.
Durch die integration der DBMS-nativen Treiberanteile in den JDBC-Treiber muß dieser für jedes anzusprechende DBMS neu erstellt werden. Eine Wiederverwendung der JDBC-spezifischen Anteile, die für die Clientkommunikation eingesetzt werden, kann hierbei nicht erfolgen.

Der Fall der (partiellen) Konkretisierung dieser Kommunikationsbeziehung zu einem beliebigen *DBMS-neutralen Protokoll* wird durch einen **Typ 3 Treiber** aufgegriffen.
Hier wird die DBMS-spezifische Komponente (in der Abbildung grau dargestellt) als vom JDBC-Treiber separiertes Modul aufgefaßt, daß mit diesem mittels eines festgelegten neutralen Protokolls kommuniziert.
Durch diese Separierung, die auch durch Installation auf physisch getrennten Maschinen --- der DBMS-spezifische Anteil könnte beispielsweise auf einem Middleware-Server untergebracht werden --- fundiert werden kann, gelingt die Wiederverwendung des JDBC-Treiberanteils, der mit verschiedenen DBMS-spezifischen Bibliotheken über das gewählte Protokoll kommunizieren kann.

Der **Typ 4 Treiber** stellt die letzte durch die JDBC-Spezifikation vorgesehene Ausprägung dar. Er konzipiert eine vollständig in Java implementierte Zugriffsschicht, die in sich geschlossen ist. Sie besitzt daher lediglich die notwendige JDBC-Schnittstelle zur Kommunikation mit der Java-Applikation und eine DBMS-Spezifische zum Zugriff auf die Datenquelle.
Die Vorteile dieser Architekturvariante liegen in ihrer Portabilität und den geringen Installations und Wartungsaufwänden, die aus der Reduktion der Kommunikationsbeziehungen resultieren. So kann ein solcher Treiber durch einfache Integration in die Java-Applikation verwendet werden und bedarf keiner Installationen oder Modifikationen an der verwendeten Ausführungsumgebung.
Gleichzeitig offenbart sich diese Lösung jedoch als technisch aufwendig in der Umsetzung, sobald DBMS verschiedener Hersteller angesprochen werden sollen, da die JDBC-Anteile des Treibers nicht separat wiederverwendet werden können.

Hinsichtlich des Laufzeitverhaltens zeigt sich deutlich die Schwäche der Typ 1 Treiber, welche in der inhärent notwendigen Doppelkonversion (JDBC zu ODBC und ODBC zu nativem Aufruf) begründet liegt.
Daher sind Treiber dieses Typs als Übergangserscheinung hin zu „echten“ JDBC-Treibern, d.h. Treibern der restlichen Typen, anzusehen und sollten in Produktivumgebungen nicht eingesetzt werden.
Die Vorteile der Typ 2 und 3 Treiber seitens der Ausführungsgeschwindigkeit liegen in den nativen Codeanteilen begründet, welche für das jeweilige verwendete DBMS optimiert werden können.
Zwar spricht der leichte Installations- und Adminstrationsaufwand eindeutig für Typ 4 Treiber, jedoch fallen diese in ihrer Leistungsfähigkeit durch die ausschließliche Verwendung der Programmiersprache Java teilweise deutlich hinter Treiber des Typs 2 und 3, mit unter sogar hinter solche des Typs 1, zurück.
Sie verkörpern jedoch den aus konzeptioneller Sicht zu bevorzugenden Ansatz hinsichtlich Portabilität und Vergleichbarkeit der erzielten quantitativen Ergebnisse.
Typischerweise kommen im produktiven Einsatz jedoch Treiber der Typen 2 und 4 zum Einsatz, die entweder durch den Hersteller des DBMS mitgeliefert werden (Typ 2) oder auf der Basis publizierter Schnittstellen plattformunabhängig für genau ein spezifisches DBMS entwickelt wurden (Typ 4).

Generell formuliert das JDBC-Konzept auf dieser Ebene noch keine Einschränkung hinsichtlich der unterstützten DBMS-Typen und ist generell auf verschiedenste Datenquellen anwendbar. Durch die Struktur des API und die verfügbaren Treiber kristallisieren sich jedoch relationale DBMS als Hauptanwendungsgebiet dieser Zugriffsschnittstelle heraus.

Im folgenden wird die Verwendung des Typ 4 Treibers Connector/J im Zusammenspiel mit dem RDBMS *MySQL* betrachtet.

Die Beispiele basieren auf einer Demodatenbank, deren Struktur und Inhalte nachfolgend angegeben sind.

Die Tabelle EMPL0YEE

```
+----------+-------+---------+-----------+------------+--------------------------+------
+----------+-----------+------+
| FNAME    | MINIT | LNAME   | SSN       | BDATE      | ADDRESS                  | SEX  |
SALARY   | SUPERSSN  | DNO  |
+----------+-------+---------+-----------+------------+--------------------------+------
+----------+-----------+------+
```

```
| John     | B     | Smith   | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX | M    |
30000.00 | 333445555 |    5 |
| Franklin | T     | Wong    | 333445555 | 1955-12-08 | 638 Voss, Houston, TX    | M    |
40000.00 | 888665555 |    5 |
| Joyce    | A     | English | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX   | F    |
25000.00 | 333445555 |    5 |
| Ramesh   | K     | Narayan | 666884444 | 1962-09-15 | 975 Fire Oak, Humble, TX | M    |
38000.00 | 333445555 |    5 |
| James    | E     | Borg    | 888665555 | 1937-11-10 | 450 Stone, Houston, TX   | M    |
55000.00 |      NULL |    1 |
| Jennifer | S     | Wallace | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX  | F    |
43000.00 | 888665555 |    4 |
| Ahmad    | V     | Jabbar  | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX  | M    |
25000.00 | 987654321 |    4 |
| Alicia   | J     | Zelaya  | 999887777 | 1968-07-19 | 3321 Castle, Spring, TX  | F    |
25000.00 | 987654321 |    4 |
+----------+-------+---------+-----------+------------+--------------------------+------
+----------+-----------+------+
```

Umsetzung in der Java-API

Das Klassendiagramm der Abbildung 4 zeigt die zentralen Klassen des Paketes `java.sql`.
Auffallend ist, daß alle Elemente des dargestellten Pakets -- abgesehen von den definierten Exceptionklassen -- als Schnittstellen ausgelegt sind. Durch diese Mimik wird die Organisation der JDBC-Schnittstelle deutlich. Die API legt lediglich das Verhalten hinsichtlich seiner Semantik und die Einzeloperationen durch Definition ihrer Parameter fest, die konkrete DBMS-spezifische Implementierung dieser Operationen wird durch den JDBC-Treiber bereitgestellt.

Zentrale Klasse der JDBC-API ist die Schnittstelle `Connection`. Sie bildet die Kommunikationsverbindungen zum DBMS ab und bietet notwendige Verwaltungsoperationen. Hierunter fallen insbesondere auch die Aufrufe zur Transaktionssteuerung.

Die Schnittstelle `Statement` realisiert genau eine aus Javasicht atomare Datenbankaktion. Diese muß hierbei aus minimal einem Aufruf an das DBMS bestehen, kann aber eine Reihe separater Aufrufe zu einem *Batch* bündeln.
Als Sonderform sieht die API die Spezialisierung `PreparedStatement` vor, die es gestattet, parametrisierte Anfragen zwischenzuspeichern, die nach Belegung der Parameterfelder an das DBMS übergeben werden. Hierdurch wird ein einfacher Mechanismus zur Wiederverwendung von DBMS-Aufrufen etabliert.

Liefert eine DBMS-Anfrage Ergebnistupel, so werden diese konform zur Schnittstelle `ResultSet` verwaltet. Diese Schnittstelle erlaubt die lesende Traversierung der vom DBMS gelieferten Tupel ebenso wie ihre Aktualisierung im Hauptspeicher und das anschließende Zurückschreiben in die Datenbank.
Die in der Abbildung nur durch *getXXX* und *updateXXX* angedeuteten Operationen existieren in Ausprägungen für alle unterstützten Datentypen, wobei *XXX* den Namen des Typs bezeichnet.

Ferner definiert die API mit `SQLWarning` eine Ausnahme zur Behandlung auftretender Fehlersituationen sowie eine Reihe weiterer, in der Abbildung 4 nicht dargestellter Klassen wie beispielsweise verschiedene Datentypen.

Die Klasse `SQLException` bietet durch ihre Methoden `getErrorCode` und `getSQLState` Möglichkeiten an um die nähere Ursache eines datenbankseitigen Fehlers zu ermitteln.
Zusätzlich gestatten Objekte dieses Ausnahmetyps die Verschachtelung von Ausnahmen, d.h. die rekursive Einbettung eines Ausnahmeereignisobjekts in ein bestehendes. Auf diesem Wege können aufgetretene Fehler durch mehrere Ausnahmeobjekte näher spezifiziert werden.
Beispiel 42 zeigt die Abfrage von Details der empfangenen und aller eingebetteten Ausnahmeereignisobjekte mittels der durch die JDBC-API vorgesehenen Methoden.

Beispiel 42: Ermittlung von Fehlerdetails

```
(1)try {
(2)     // JDBC code
(3)} catch (SQLException e) {
(4)     while (e != null) {
(5)             System.err.println("SQLState: " + e.getSQLState());
(6)      System.err.println("Message:  " + e.getMessage());
(7)      System.err.println("Vendor:   " + e.getErrorCode());
(8)      System.err.println("-----");
(9)      e = e.getNextException();
(10)    }
(11)}
```

Download des Beispiels

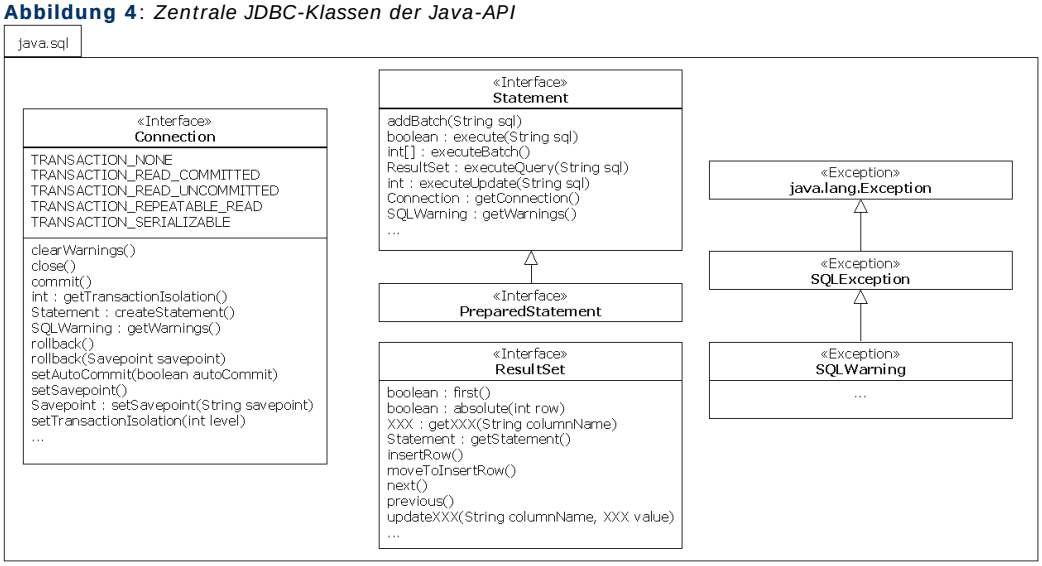
Mit der Version 1.4 der Java-Standard-Edition wurde die zuvor nur in der JDBC-API zur Verfügung stehende Möglichkeit zur Schachtelung von Ausnahmeereignissen auch für beliebige Ausnahmeereignisobjekte des Typs `Throwable` definiert.
Anders als die JDBC-API sieht die generische Lösung jedoch die Nutzung der Methode `getCause` zur Extraktion der eingebetteten Ausnahmeereignisobjekte vor.
Der Code des Beispiels 43 spiegelt daher die Standard-API-konforme Realisierung wieder. Zusätzlich wendet die Lösung die Standard-Methode `getMessage` zur Ermittlung der deskriptiven Fehlerbeschreibung an.

Beispiel 43: Standard-API-konforme Ermittlung von Fehlerdetails

```
(1)try {
(2)     // Normal code
(3)} catch (SQLException e) {
(4)     Throwable t = e;
(5)     while (t != null) {
(6)     System.err.println("Type: " + t.getClass().getName());
(7)      System.err.println("Message: " + t.getMessage());
(8)      System.err.println("-----");
(9)      t = t.getCause();
(10)    }
(11)}
```

Download des Beispiels

Abbildung 4: *Zentrale JDBC-Klassen der Java-API*



(click on image to enlarge!)

Zugriff auf die Datenbank

Beispiel 44 zeigt den Ablauf zur Aufnahme einer Verbindung mit der Datenbank `jdbctest` auf dem lokalen Rechner (`localhost`).

Zunächst muß die Klasse des gewählten JDBC-Treibers (im Beispiel `com.mysql.jdbc.Driver` vor ihrer Verwendung geladen werden. Dies geschieht durch den Aufruf der statischen Methode `forName` auf der Klasse `Class`.
Der zu ladende Treiber muß hierbei die JDBC-Schnittstellenklasse `Driver` implementieren um später durch die JDBC-API verwendet werden zu können.
Gleichzeitig mit dem dynamischen Ladevorgang erfolgt die Registrierung des Treibers beim JDBC-`DriverManager`, der die Verwaltung der geladenen DB-Treiber übernimmt.

Nach dem erfolgreichen Laden des Treibers wird durch den Aufruf von `getConnection` (Zeile 16) die Verbindung zur Datenbank hergestellt. Die anzusprechende Datenbank wird hierbei durch eine URI der Form `jdbc:mysql://DB-Server/DB-Name` repräsentiert (Zeile 17). Zusätzlich können ein zur Anmeldung am DB-System benötiger Benutzer (Zeile 18) und sein Paßwort (Zeile 19) übergeben werden.

Beispiel 44: Aufbau einer Datenbankverbindung

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)
(5)public class JDBCConnect {
(6)     public static void main(String[] args) {
(7)             try {
(8)                     Class.forName("com.mysql.jdbc.Driver");
(9)             } catch (ClassNotFoundException e) {
(10)                    System.err.println("Driver class not found");
(11)                    e.printStackTrace();
(12)            }
(13)            Connection con = null;
(14)            try {
(15)                    con =
(16)                            (Connection) DriverManager.getConnection(
(17)                                    "jdbc:mysql://localhost/jdbctest/",
(18)                                    "mario",
(19)                                    "thePassword");
(20)            } catch (SQLException e1) {
(21)                    System.err.println("Error establishing database connection");
(22)                    Throwable t = e1;
(23)                    while (t != null) {
(24)                    System.err.println("Type: " + t.getClass().getName());
(25)                    System.err.println("Message: " + t.getMessage());
(26)                    System.err.println("-----");
(27)                    t = t.getCause();
(28)                    }
(29)            }
(30)    }
(31)}
```

Download des Beispiels

Zusätzlich stellen die Klassen `Driver` und `DriverManager` die Möglichkeit der Abfrage von verbindungsunabhängigen Verwaltungsinformationen zur Verfügung.

Beispiel 45: Ermittlung von Informationen über Treiber und Treibermanager

```
(1)import java.sql.Driver;
(2)import java.sql.DriverManager;
(3)import java.util.Enumeration;
(4)
(5)public class JDBCDriver {
(6)
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)
(15)            System.out.println(
(16)                    "DriverManager:\nlogin timeout=" + DriverManager.getLoginTimeout
()));
(17)
(18)            Enumeration e = DriverManager.getDrivers();
(19)            while (e.hasMoreElements()) {
(20)                    Driver drv = (Driver) e.nextElement();
(21)
(22)                    System.out.println(
(23)                            "Driver="
(24)                                    + drv.getClass().getName()
(25)                                    + "\nmajor version="
(26)                                    + drv.getMajorVersion()
(27)                                    + "\nminor version="
(28)                                    + drv.getMinorVersion()
(29)                                    + "\nJDBC compliant="
(30)                                    + drv.jdbcCompliant());
(31)            }
(32)
(33)    }
(34)}
```

Download des Beispiels
Download der Ergebnisdatei

Beispiel 45 zeigt die Ermittlung des durch den `DriverManager` für alle durch ihn verwalteten Treiber global definierten Login Timouts, der angibt wie lange beim Anmeldevorgang an der Datenbank auf eine Rückmeldung gewartet wird.
Zusätzlich werden für alle verwalteten Treiber der Klassenname sowie Daten zur Version und zum Stand der JDBC-Unterstützung ermittelt und ausgegeben.
Der JDBC-Unterstützungsstand gibt an, ob ein gegebener Treiber die Konformitätstests der Firma SUN bestanden hat. Voraussetzung hierfür ist u.a. die vollständige Unterstützung des SQL 92-Standards (entry level).
Diese Interpreatation von Spezifikationskonformität verwundert etwas, da alle JDBC-Treiber mit Ausnahme der inhärent DB-neutralen Typ 1 Treiber DBMS-spezifisch realisiert sind. Aus diesem Grunde bewertet der Konformitätstest vielmehr den Umsetzungsgrad des SQL-Standards in dem via JDBC genutzten DBMS als die Güte des JDBC-Treibers selbst.

Seit der JDBC-Schnittstellenversion 2 ist neben der „klassischen" Zugriffsvariante auch eine auf dem Java Naming and Directory Interface (JNDI) basierende Zugriffsmethodik definiert, deren Verwendung --- abgesehen von der geänderten Mimik im Aufbau der DB-Verbindung --- identisch gestaltet ist.

Jedoch ist, wie in JNDI üblich, vor dem Zugriff ein benanntes Objekt beim JNDI-Dienst zu registrieren.
Im Falle von JDBC ist dies ein Objekt welches die Schnittstelle `DataSource` implementiert.

Der Code des Beispiels 46 zeigt die notwendigen Schritte zur Registrierung eines `MysqlDataSource`-Objekts, der durch den MySQL-JDBC-Treiber gelieferten Implementierung der Schnittstelle `DataSource`.

Beispiel 46: Ablage von Verbindungsinformation in einem JNDI-Verzeichnis

```
(1)import java.util.Hashtable;
(2)import javax.naming.Context;
(3)import javax.naming.InitialContext;
(4)import javax.naming.NamingException;
(5)import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;
(6)
(7)public class JDBCConnect2Server {
(8)
(9)     public static void main(String[] args) {
(10)            Hashtable env = new Hashtable();
(11)            env.put(
(12)                    Context.INITIAL_CONTEXT_FACTORY,
(13)                    "com.sun.jndi.fscontext.RefFSContextFactory");
(14)            env.put(Context.PROVIDER_URL, "file:/tmp/registry");
(15)
(16)            MysqlDataSource ds = new MysqlDataSource();
(17)            ds.setDatabaseName("jdbctest");
(18)            Context ctx = null;
(19)            try {
(20)                    ctx = new InitialContext(env);
(21)            } catch (NamingException ne) {
(22)                    ne.printStackTrace();
(23)            }
(24)
(25)            try {
(26)                    ctx.rebind("jdbc/mySrc", ds);
(27)            } catch (NamingException ne) {
(28)                    ne.printStackTrace();
(29)            }
(30)    }
(31)}
```

Download des Beispiels

Entsprechend der modifizierten Ablage der Verwaltungsinformation ändert sich die Erzeugung der Datenbankverbindung beim Zugriff. Hier wird nun zunächst über einen Zugriff auf den JNDI-Verzeichnisdienst das benannte `DataSource`-Objekt (es trägt den Namen `jdbc/mySrc` ermittelt.
Anschließend wird durch das dem Verzeichnisdienst entnommene `DataSource`-Objekt die Datenbankverbindung (d.h. das `Connection`-Objekt) erzeugt.
Alle weiteren Schritte zur Interaktion mit der Datenbank verlaufen dann identisch zur im Beispiel 44 gezeigten Verbindungsaufnahme.
Der Code des Beispiels 47 zeigt die notwendigen Schritte zur Ermittlung der Referenz auf das Objekt des Typs `DataSource` aus dem JNDI-Verzeichnis, sowie die Erzeugung des `Connection`-Objekts.

Beispiel 47: Verbindungsaufbau unter Nutzung von JNDI

```
(1)import java.sql.Connection;
(2)import java.sql.SQLException;
(3)import java.util.Hashtable;
(4)import javax.naming.Context;
(5)import javax.naming.InitialContext;
(6)import javax.naming.NamingException;
(7)import javax.sql.DataSource;
(8)
(9)public class JDBCConnect2 {
(10)    public static void main(String[] args) {
(11)            Hashtable env = new Hashtable();
(12)            env.put(
(13)                    Context.INITIAL_CONTEXT_FACTORY,
(14)                    "com.sun.jndi.fscontext.RefFSContextFactory");
(15)            env.put(Context.PROVIDER_URL, "file:/tmp/registry");
(16)            Context ctx = null;
(17)            try {
(18)                    ctx = new InitialContext(env);
(19)            } catch (NamingException ne) {
(20)                    ne.printStackTrace();
(21)            }
(22)            DataSource ds = null;
(23)            try {
(24)                    ds = (DataSource) ctx.lookup("jdbc/mySrc");
(25)            } catch (NamingException ne) {
(26)                    ne.printStackTrace();
(27)            }
(28)            Connection con = null;
(29)            try {
(30)                    con = ds.getConnection("mario", "thePassword");
(31)            } catch (SQLException sqle) {
(32)                    Throwable t = sqle;
(33)                    while (t != null) {
(34)                    System.err.println("Type: " + t.getClass().getName());
(35)                    System.err.println("Message: " + t.getMessage());
(36)                    System.err.println("-----");
(37)                    t = t.getCause();
(38)                    }
(39)            }
(40)    }
(41)}
```

Download des Beispiels

Auffallend ist die Ablage des Datenbanknamens im Verzeichnisdienst mittels des Methodenaufrufs `setDatabaseName`. Diese Verschiebung der Information wird durch die geänderte Mimik der Erzeugung des `Connection`-Objekts impliziert. So sieht die Implementierung dieser Methode für die Klasse `DataSource` keine Möglichkeit zur gleichzeitigen Übergabe von Anmeldenamen, Paßwort und Datenbank vor. Vielmehrnoch ist es sogar möglich diese Daten allesamt innerhalb des JNDI-Verzeichnisdienstes abzulegen. (Für diesen Zweck stehen die Methoden `setUser` bzw. `setPassword` zur Verfügung.) Als Konsequenz hiervon kann der Verbinungswunsch durch Aufruf der Methode `getConnection` ohne weitere Parameter erfüllt werden.
Diese Umsetzungsweise ist vor ihrer Realisierung hinsichtlich des damit eintretenden Verlustes an Sicherheit zu prüfen, da in ihrer Folge eine Datenbankverbindung allein durch Kenntnis des JNDI-residenten Namens des `DataSource`-Objektes erfolgen kann.

Generell wählen JDBC-Umsetzungen den Weg, jede Ausprägung eines `Connection`-Objekts in eine physische Datenbankverbindung abzubilden. Dieses, durchaus der intuitiven Semantik der `Connection`-Klasse entsprechende Vorgehen kann jedoch in realen Applikationen, begründet in der Vielzahl der durch das DBMS zu verwaltenden Verbindungen, zu Zugriffsengpässen führen.
Aus diesem Grunde definiert die JDBC-Schnittstelle Operationen zur Zusammenfassung „gleichartiger“ Zugriffe. Hierzu zählen Zugriffe die unter derselben Nutzerkennung auf dieselbe Datenbank abgewickelt werden. Diese Zugriffsform tritt insbesondere bei Anwendungen auf, die über nur einen in der Datenbank eingetragenen Anwender verfügen und die gesamte Nutzerverwaltung datenbanktransparent applikationsseitig abwickeln.
Zur Optimierung von Zugriffen dieser Natur sieht die JDBC-Schnittstelle das sog. *Connection Pooling* vor, welches gleichartige Zugriffe bündelt.
Das Beispiel 48 zeigt eine Umsetzung:

Beispiel 48: Verbindungsaufbau unter Nutzung von Connection Pooling

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import javax.sql.PooledConnection;
(4)import com.mysql.jdbc.Connection;
(5)import com.mysql.jdbc.jdbc2.optional.MysqlPooledConnection;
(6)
(7)public class JDBCConnection3 {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.mysql.jdbc.Driver");
(11)            } catch (ClassNotFoundException cnfe) {
(12)                    System.err.println("Driver class not found");
(13)                    cnfe.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                    System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            PooledConnection pc = new MysqlPooledConnection(con);
(34)
(35)            java.sql.Connection con1 = null;
(36)            try {
(37)                    con1 = pc.getConnection();
(38)            } catch (SQLException sqle) {
(39)                    Throwable t = sqle;
(40)                    while (t != null) {
(41)                    System.err.println("Type: " + t.getClass().getName());
(42)                    System.err.println("Message: " + t.getMessage());
(43)                    System.err.println("-----");
(44)                    t = t.getCause();
(45)                    }
(46)            }
(47)    }
(48)}
```

Download des Beispiels

Statt für jede gewünschte Datenbankverbindung ein zusätzliches Objekt des Type `Connection` zu erzeugen, wird die erzeugte Verbindung zur Konstruktion eines Objektes, welches Konform zur Schnittstelle `PooledConnection` definiert ist, verwendet. Dieses sorgt für die Verwaltung der DB-Verbindung und stellt dieselbe physische Verbindung verschiedenen Anfragern zur Verfügung. Konsequenterweise wird daher eine neue Verbindung nicht mehr vom `DriverManager` angefordert, sondern durch die Methode `getConnection` der aus der Verwaltungsstruktur entnommenen `PooledConnection` beantragt.

Aufgrund der Unterstützung des SQL-Sprachumfanges, durch unveränderte textuelle Propagation an das DBMS sind durch JDBC im Allgemeinen alle Facetten der Datenbanksprache nutzbar, sofern sie durch das verwendete DBMS Unterstützung finden. Hierunter fallen:

- Data Definition Language.
  Zur Erzeugung eines Datenmodells.
- Data Manipulation Language.
  Zur Modifikation der verwalteten Daten.
- Data Retrieval Language.
  Zur Anfrage der in einer Datenbank gespeicherten Daten.
- Data Control Language.
  Zur Festlegung und Kontrolle von Zugriffsberechtigungen.

JDBC reflektiert jedoch nicht diese Sprach(-sub-)klassen selbst in der API, sondern sieht vielmehr ausschließlich zwei Formen des Zugriffs vor. Solche die tabellenwerte Resultate liefern und solche, deren

Ausführung lediglich primitivwertige Rückgabewerte liefert.

Primitivwertige Zugriffe

Primitivwertige Datenbankzugriffe liefern, abgesehen von Fehler- oder Warnmeldungen, lediglich die Anzahl der geänderten Tupel, falls zutreffend, oder 0 zurück.
Aus dieser Festlegung lassen sich diejenigen SQL-Anweisungstypen ableiten, welche als primitivwertiger Zugriff realisiert sind. Hierunter fallen alle Operationen der Datendefinition wie `CREATE` oder `ALTER TABLE` sowie alle Einfüge- (`INSERT`) Änderungs- (`UPDATE`) und Löschvorgänge (`DELETE`). Darüberhinaus alle Operationen zur Administration der Datenbank durch Rechtevergabe (`GRANT`, `REVOKE`).

Zugriffe dieser Art werden generell durch die Methode `executeUpdate`, oder einer Abart davon, realisiert.

Beispiel 49: Erstellung einer neuen Tabelle

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.Statement;
(5)
(6)public class JDBCCreateTable {
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)            Connection con = null;
(15)
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                    System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            Statement  stmt = null;
(34)            try {
(35)                    stmt = (Statement) con.createStatement();
(36)            } catch (SQLException e2) {
(37)                    System.err.println("Error creating SQL-Statement");
(38)                    Throwable t = e2;
(39)                    while (t != null) {
(40)                    System.err.println("Type: " + t.getClass().getName());
(41)                    System.err.println("Message: " + t.getMessage());
(42)                    System.err.println("-----");
(43)                    t = t.getCause();
(44)                    }
(45)            }
(46)            String createTab = new String("CREATE TABLE EMPLOYEE(" +
(47)                    "FNAME VARCHAR(10) NOT NULL," +
(48)                    "MINIT VARCHAR(1)," +
(49)                    "LNAME VARCHAR(10) NOT NULL," +
(50)                    "SSN INTEGER(9) NOT NULL," +
(51)                    "BDATE DATE," +
(52)                    "ADDRESS VARCHAR(30)," +
(53)                    "SEX ENUM('M','F')," +
(54)                    "SALARY REAL(7,2) UNSIGNED," +
(55)                    "SUPERSSN INTEGER(9)," +
(56)                    "DNO INTEGER(1));");
(57)            try {
(58)                    System.out.println("result="+stmt.executeUpdate(createTab));
(59)            } catch (SQLException e3) {
(60)                    System.err.println("Error creating table EMPLOYEE");
(61)                    Throwable t = e3;
```

```
(62)                    while (t != null) {
(63)                    System.err.println("Type: " + t.getClass().getName());
(64)                    System.err.println("Message: " + t.getMessage());
(65)                    System.err.println("-----");
(66)                    t = t.getCause();
(67)                    }
(68)            }
(69)    }
(70)}
```

Download des Beispiels
Download der Ergebnisdatei

Beispiel 49 zeigt die notwendigen Schritte zur Erstellung der Tabelle EMPLOYEE in der Datenbank.

Nach dem (üblichen) Verbindungsaufbau (Zeile 8-24) wird in Zeile 27 eine Variable des Typs Statement deklariert. Auch bei Statement handelt es sich um eine durch die JDBC-API vordefinierte Schnittstelle, die als Bestandteil des JDBC-Treibers von einer Klasse implementiert wird.
Ausgehend von der etablierten Datenbankverbindung wird durch Aufruf der Methode createStatement eine konkrete Ausprägung konform zur Statement-Schnittstelle erzeugt (Zeile 29).

Der Aufruf von executeUpdate übergibt das als Zeichenkette abgelegte SQL-Kommando an die Datenbank zur Ausführung.
Da durch CREATE TABLE keine Tupeländerungen vorgenommen werden ist das Resultat des Aufrufs der Rückgabewert 0.

Beispiel 50 zeigt mit dem ALTER TABLE-Kommando eine weitere Anwendung der executeUpdate-Methode. Auch in diesem Falle wird als Resultat 0 geliefert, da die Definition des Primärschlüssels keine Änderungen an den verwalteten Datensätzen vornimmt.

Beispiel 50: Modifikation der Tabellendefinition

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.Statement;
(5)
(6)public class JDBCAlterTable {
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)            Connection con = null;
(15)
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                            System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            Statement stmt = null;
(34)            try {
(35)                    stmt = (Statement) con.createStatement();
(36)            } catch (SQLException e2) {
(37)                    System.err.println("Error creating SQL-Statement");
(38)                    Throwable t = e2;
(39)                    while (t != null) {
(40)                    System.err.println("Type: " + t.getClass().getName());
(41)                    System.err.println("Message: " + t.getMessage());
(42)                    System.err.println("-----");
(43)                    t = t.getCause();
```

```
(44)                    }
(45)            }
(46)            String createTab =
(47)                    new String("ALTER TABLE EMPLOYEE ADD PRIMARY KEY (SSN);");
(48)            try {
(49)                    System.out.println("result=" + stmt.executeUpdate(createTab));
(50)            } catch (SQLException e3) {
(51)                    System.err.println("Error altering table EMPLOYEE");
(52)                    Throwable t = e3;
(53)                    while (t != null) {
(54)                    System.err.println("Type: " + t.getClass().getName());
(55)                    System.err.println("Message: " + t.getMessage());
(56)                    System.err.println("-----");
(57)                    t = t.getCause();
(58)                    }
(59)            }
(60)    }
(61)}
```

Download des Beispiels
Download der Ergebnisdatei

Beispiel 51: Einfügen von Werten

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.Statement;
(5)
(6)public class JDBCInsert1 {
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)            Connection con = null;
(15)
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                    System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            Statement stmt = null;
(34)            try {
(35)                    stmt = (Statement) con.createStatement();
(36)            } catch (SQLException e2) {
(37)                    System.err.println("Error creating SQL-Statement");
(38)                    Throwable t = e2;
(39)                    while (t != null) {
(40)                    System.err.println("Type: " + t.getClass().getName());
(41)                    System.err.println("Message: " + t.getMessage());
(42)                    System.err.println("-----");
(43)                    t = t.getCause();
(44)                    }
(45)            }
(46)
(47)            try {
(48)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('John', 'B', 'Smith', 123456789, '1965-01-09', '731 Fondren, Houston, TX',
'M', 30000, 333445555, 5);"));
(49)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
```

```
EMPLOYEE VALUES('Franklin', 'T', 'Wong', 333445555, '1955-12-08', '638 Voss, Houston, TX',
'M', 40000, 888665555, 5);"));
(50)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('Alicia', 'J', 'Zelaya', 999887777, '1968-07-19', '3321 Castle, Spring,
TX', 'F', 25000, 987654321, 4);"));
(51)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('Jennifer', 'S', 'Wallace', 987654321, '1941-06-20', '291 Berry, Bellaire,
TX', 'F', 43000, 888665555, 4);"));
(52)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('Ramesh', 'K', 'Narayan', 666884444, '1962-09-15', '975 Fire Oak, Humble,
TX', 'M', 38000, 333445555, 5);"));
(53)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('Joyce', 'A', 'English', 453453453, '1972-07-31', '5631 Rice, Houston,
TX', 'F', 25000, 333445555, 5);"));
(54)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('Ahmad', 'V', 'Jabbar', 987987987, '1969-03-29', '980 Dallas, Houston,
TX', 'M', 25000, 987654321, 4);"));
(55)                    System.out.println("result=" + stmt.executeUpdate("INSERT INTO
EMPLOYEE VALUES('James', 'E', 'Borg', 888665555, '1937-11-10', '450 Stone, Houston, TX',
'M', 55000, null, 1);"));
(56)            } catch (SQLException e3) {
(57)                    System.err.println("Error inserting values into table EMPLOYEE");
(58)                    Throwable t = e3;
(59)                    while (t != null) {
(60)                    System.err.println("Type: " + t.getClass().getName());
(61)                    System.err.println("Message: " + t.getMessage());
(62)                    System.err.println("-----");
(63)                    t = t.getCause();
(64)                    }
(65)            }
(66)    }
(67)}
```

Download des Beispiels
Download der Ergebnisdatei

Beispiel 51 zeigt den Einfügevorgang von acht Werten in die durch die vorangegangenen Beispiele erzeugte Tabelle `EMPLOYEE`.
Jeder der Einfügevorgänge der Zeilen 36-43 führt im Rahmen einer separaten Datenbankkommunikation sequentiell genau einen Einfügevorgang durch, was durch den Rückgabewert 1 dokumentiert wird.

Zwar ist dieses Verfahren praktikabel und erzielt die angestrebten Resultate, jedoch ist es unter Zeiteffizienzgesichtspunkten inadäquat, da sich Einfüge- und Kommunikationsvorgänge zahlenmäßig entsprechen.

Aus diesem Grunde bietet die Schnittstelle `Statement` die Möglichkeit zur Bündelung einzelner SQL-Aufrufe in einem sog. *Batch* an.

Beispiel 52 zeigt die entsprechende Umgestaltung des vorangegangenen Beispiels.

Beispiel 52: Einfügen von Werten mittels eines Batches

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.Statement;
(5)
(6)public class JDBCInsert2 {
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)            Connection con = null;
(15)
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
```

```
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                    System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            Statement stmt = null;
(34)            try {
(35)                    stmt = (Statement) con.createStatement();
(36)            } catch (SQLException e2) {
(37)                    System.err.println("Error creating SQL-Statement");
(38)                    Throwable t = e2;
(39)                    while (t != null) {
(40)                    System.err.println("Type: " + t.getClass().getName());
(41)                    System.err.println("Message: " + t.getMessage());
(42)                    System.err.println("-----");
(43)                    t = t.getCause();
(44)                    }
(45)            }
(46)
(47)            try {
(48)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('John', 'B', 'Smith',
123456789, '1965-01-09', '731 Fondren, Houston, TX', 'M', 30000, 333445555, 5);");
(49)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Franklin', 'T',
'Wong', 333445555, '1955-12-08', '638 Voss, Houston, TX', 'M', 40000, 888665555, 5);");
(50)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Alicia', 'J',
'Zelaya', 999887777, '1968-07-19', '3321 Castle, Spring, TX', 'F', 25000, 987654321, 4);");
(51)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Jennifer', 'S',
'Wallace', 987654321, '1941-06-20', '291 Berry, Bellaire, TX', 'F', 43000, 888665555,
4);");
(52)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Ramesh', 'K',
'Narayan', 666884444, '1962-09-15', '975 Fire Oak, Humble, TX', 'M', 38000, 333445555,
5);");
(53)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Joyce', 'A',
'English', 453453453, '1972-07-31', '5631 Rice, Houston, TX', 'F', 25000, 333445555, 5);");
(54)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Ahmad', 'V', 'Jabbar',
987987987, '1969-03-29', '980 Dallas, Houston, TX', 'M', 25000, 987654321, 4);");
(55)                    stmt.addBatch("INSERT INTO EMPLOYEE VALUES('James', 'E', 'Borg',
888665555, '1937-11-10', '450 Stone, Houston, TX', 'M', 55000, null, 1);");
(56)                    int[] insertCounts = stmt.executeBatch();
(57)            } catch (SQLException e3) {
(58)                    System.err.println("Error inserting values into table EMPLOYEE");
(59)                    Throwable t = e3;
(60)                    while (t != null) {
(61)                    System.err.println("Type: " + t.getClass().getName());
(62)                    System.err.println("Message: " + t.getMessage());
(63)                    System.err.println("-----");
(64)                    t = t.getCause();
(65)                    }
(66)            }
(67)    }
(68)}
```

Download des Beispiels

Statt der Einzelübergabe der SQL `INSERT`-Anweisungen werden diese nun (in Zeile 36-43) in in einem Batch gesammelt. Hierzu werden die SQL-Zeichenketten durch den Aufruf `addBatch` innerhalb des `Statement`-Objekts abgelegt und durch Aufruf der Methode `executeBatch` gesammelt an das DBMS übergeben.
Statt der Einzelresultate wird durch diese Aufrufvariante ein Array geliefert, das die Einzelrückgabewerte der als Batch übergebenen Aufrufe versammelt.

Dies verdeutlicht nochmals das nachfolgende Beispiel. In ihm wird zunächst mittels `ALTER TABLE` eine neue Tabellenspalte zur Aufnahme des Wochentages der Geburt erstellt und anschließend durch SQL `UPDATE`-Anweisungen die benötigten Daten aus dem vorhandenen Geburtsdatum ermittelt.
Auch dieses Beispiel bedient sich zur Performancebeschleunigung der Möglichkeiten des Batchaufrufes.

Beispiel 53: Aktualisieren von Tabellendefinitionen und Werten

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.Statement;
(5)
(6)public class JDBCUpdate1 {
(7)     public static void main(String[] args) {
(8)             try {
(9)                     Class.forName("com.mysql.jdbc.Driver");
(10)            } catch (ClassNotFoundException e) {
(11)                    System.err.println("Driver class not found");
(12)                    e.printStackTrace();
(13)            }
(14)            Connection con = null;
(15)
(16)            try {
(17)                    con =
(18)                            (Connection) DriverManager.getConnection(
(19)                                    "jdbc:mysql://localhost/jdbctest/",
(20)                                    "mario",
(21)                                    "thePassword");
(22)            } catch (SQLException e1) {
(23)                    System.err.println("Error establishing database connection");
(24)                    Throwable t = e1;
(25)                    while (t != null) {
(26)                    System.err.println("Type: " + t.getClass().getName());
(27)                    System.err.println("Message: " + t.getMessage());
(28)                    System.err.println("-----");
(29)                    t = t.getCause();
(30)                    }
(31)            }
(32)
(33)            Statement stmt = null;
(34)            try {
(35)                    stmt = (Statement) con.createStatement();
(36)            } catch (SQLException e2) {
(37)                    System.err.println("Error creating SQL-Statement");
(38)                    Throwable t = e2;
(39)                    while (t != null) {
(40)                    System.err.println("Type: " + t.getClass().getName());
(41)                    System.err.println("Message: " + t.getMessage());
(42)                    System.err.println("-----");
(43)                    t = t.getCause();
(44)                    }
(45)            }
(46)
(47)            try {
(48)                    stmt.addBatch("ALTER TABLE EMPLOYEE ADD BDAY VARCHAR(10);");
(49)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Sunday' WHERE DAYOFWEEK
(BDATE)=1;");
(50)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Monday' WHERE DAYOFWEEK
(BDATE)=2;");
(51)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Tuesday' WHERE DAYOFWEEK
(BDATE)=3;");
(52)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Wednesday' WHERE DAYOFWEEK
(BDATE)=4;");
(53)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Thursday' WHERE DAYOFWEEK
(BDATE)=5;");
(54)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Friday' WHERE DAYOFWEEK
(BDATE)=6;");
(55)                    stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Saturday' WHERE DAYOFWEEK
(BDATE)=7;");
(56)                    int[] result = stmt.executeBatch();
(57)                    for (int i=0; i<result.length;i++){
(58)                            System.out.println("Statement No "+i+" changed "+result[i]
+" rows");
(59)                    }
(60)            } catch (SQLException e3) {
(61)                    System.err.println("Error inserting values into table EMPLOYEE");
(62)                    Throwable t = e3;
(63)                    while (t != null) {
(64)                    System.err.println("Type: " + t.getClass().getName());
(65)                    System.err.println("Message: " + t.getMessage());
(66)                    System.err.println("-----");
(67)                    t = t.getCause();
(68)                    }
```

```
(69)            }
(70)    }
(71)}
```

Download des Beispiels
Download der Ergebnisdatei

Die Ausführung liefert als Resultat:

```
Statement No 0 changed 8 rows
Statement No 1 changed 0 rows
Statement No 2 changed 1 rows
Statement No 3 changed 0 rows
Statement No 4 changed 1 rows
Statement No 5 changed 1 rows
Statement No 6 changed 2 rows
Statement No 7 changed 3 rows
```

So werden durch den ALTER TABLE-Aufruf (Indexnummer 0) alle acht Tupel der Tabelle modifiziert, während die nachfolgenden Aufrufe nur Teilmengen davon verändern.

Die nähere Betrachtung der Zeilen 37-43 des Quellcodes von Beispiel 53 zeigt sich, daß diese im Kern denselben Vorgang ausführen, nur jeweils mit variierenden Parametern.
Zur Behandlung von Fällen dieser Problemstellung definiert die JDBC-API die Schnittstelle `PreparedStatement` als Spezialisierung von `Statement`.

Diese Schnittstelle gestattet es, Anweisungen, die später an die Datenbank übermittelt werden sollen, mit Platzhaltern zu versehen und diese vor der Übermittlung mit Werten zu befüllen.
Beispiel 54 zeigt die entprechende Modifikation des vorangegangenen Beispiels.

Beispiel 54: Aktualisieren von Tabellendefinitionen und Werten

```
(1)import java.sql.DriverManager;
(2)import java.sql.SQLException;
(3)import com.mysql.jdbc.Connection;
(4)import com.mysql.jdbc.PreparedStatement;
(5)import com.mysql.jdbc.Statement;
(6)
(7)public class JDBCUpdate2 {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.mysql.jdbc.Driver");
(11)            } catch (ClassNotFoundException e) {
(12)                    System.err.println("Driver class not found");
(13)                    e.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)
(17)            try {
(18)                    con =
(19)                            (Connection) DriverManager.getConnection(
(20)                                    "jdbc:mysql://localhost/jdbctest/",
(21)                                    "mario",
(22)                                    "thePassword");
(23)            } catch (SQLException e1) {
(24)                    System.err.println("Error establishing database connection");
(25)                    Throwable t = e1;
(26)                    while (t != null) {
(27)                    System.err.println("Type: " + t.getClass().getName());
(28)                    System.err.println("Message: " + t.getMessage());
(29)                    System.err.println("-----");
(30)                    t = t.getCause();
(31)                    }
(32)            }
(33)
(34)            Statement stmt = null;
(35)            PreparedStatement pstmt = null;
(36)            try {
(37)                    stmt = (Statement) con.createStatement();
(38)                    pstmt = (PreparedStatement) con.prepareStatement("UPDATE EMPLOYEE
SET BDAY=? WHERE DAYOFWEEK(BDATE)=?;");
(39)
(40)            } catch (SQLException e2) {
(41)                    System.err.println("Error creating SQL-Statement");
(42)                    Throwable t = e2;
```

```
(43)                    while (t != null) {
(44)                    System.err.println("Type: " + t.getClass().getName());
(45)                    System.err.println("Message: " + t.getMessage());
(46)                    System.err.println("-----");
(47)                    t = t.getCause();
(48)                    }
(49)            }
(50)
(51)            try {
(52)                    String[] days=
{"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"} ;
(53)                    stmt.addBatch("ALTER TABLE EMPLOYEE ADD BDAY VARCHAR(10);");
(54)                    for (int i=1; i<8;i++){
(55)                            pstmt.setString(1,days[i-1]);
(56)                            pstmt.setInt(2,i);
(57)                            pstmt.addBatch();
(58)                    }
(59)                    int[] result = stmt.executeBatch();
(60)                    for (int i=0; i<result.length;i++){
(61)                            System.out.println("Statement No "+i+" changed "+result[i]
+" rows");
(62)                    }
(63)            } catch (SQLException e3) {
(64)                    System.err.println("Error inserting values into table EMPLOYEE");
(65)                    Throwable t = e3;
(66)                    while (t != null) {
(67)                    System.err.println("Type: " + t.getClass().getName());
(68)                    System.err.println("Message: " + t.getMessage());
(69)                    System.err.println("-----");
(70)                    t = t.getCause();
(71)                    }
(72)            }
(73)    }
(74)}
```

Download des Beispiels
Download der Ergebnisdatei

Im Beispiel wird neben dem Objekt des Typs `Statement` zusätzlich eines des Typs `PreparedStatement` erzeugt (Zeile 32).
Die dem Konstruktor übergebene Anweisung enthält als Sonderzeichen zur Markierung der Platzhalter das Fragezeichen (?).
Die Wochentage werde in Zeile 40, des vereinfachten Zugriffs wegen, als Array definiert.
In den Zeilen 42 mit 46 werden die benötigten SQL-`UPDATE`-Anweisungen dynamisch durch Einsetzen der geeigneten Werte in den vorpräparierten Änderungsausruck erzeugt und einem eigenen Batch zugeordnet.
Der Einsetzungsvorgang der benötigten Werte geschieht durch die Methoden `setString` für zeichenkettenartige bzw. `setInt` für den ganzzahlige Parameter. Den Methoden wird jeweils die Position des Parameters, gezählt ab 1 sowie die zu wählende Wertbelegung übermittelt.
Zur Ausführung müssen beide Batches getrennt angefordert werden.

Tabellenwertige Zugriffe

Die in der Praxis quantitativ bedeutendste Klasse von Datenbankzugriffen dürfte zweifellos auf die lesende Ermittlung von bestehenden Daten darstellen, kurzum alle Spielarten der SQL `SELECT`-Anweisung.

Für Anfragen an die Datenbank steht prinzipiell der gesamte durch das DBMS unterstützte SQL-Umfang zur Verfügung.

Anfragen werden im Gegensatz zu den bisher betrachteten lesenden Zugriffen nicht als primivwerte Methoden realisiert, sondern liefern als Resultat immer eine Tabelle zurück.
Diese wird durch den API-Typ `ResultSet` dargestellt.
Zusätzlich werden Anfragen durch die Methode executeQuery ausgeführt.

Das Beispiel 55 zeigt die generische Extraktion von DB-Daten und den Zugriff auf Metadaten.
Die aus der Datenbank gelesenen Ergebnistupel werden im durch `rs` benannten `ResultSet` abgelegt (Zeile 39). Die Resultatmenge wird mithilfe eines *Cursors* (Datensatzzeiger) traversiert. Hierzu wird der initial auf eine Ausgangsstellung vor dem ersten empfangenen Tupel positionierte Cursor durch Aufruf der Methode `next` solange weitergerückt, bis der letzte Datensatz verarbeitet wurde.

Der Aufruf der Methode`getMetaData` liefert deskriptive Metadaten wie Spaltenzahl sowie deren Bezeichner und Typen für die erstellte Resultattupelmenge.
In Zeile 43 werden diese Metadaten verwendet um die Spaltennamen der extrahierten Attribute

anzuzeigen.
Zeile 47-52 liest die einzelnen Werte jedes Tupels mittels getObject aus und stellt sie am Bildschirm dar.

Beispiel 55: Auslesen von Daten und Metadaten

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.ResultSetMetaData;
(4)import java.sql.SQLException;
(5)
(6)import com.mysql.jdbc.Connection;
(7)import com.mysql.jdbc.Statement;
(8)
(9)public class JDBCSelect1 {
(10)    public static void main(String[] args) {
(11)            try {
(12)                    Class.forName("com.mysql.jdbc.Driver");
(13)            } catch (ClassNotFoundException e) {
(14)                    System.err.println("Driver class not found");
(15)                    e.printStackTrace();
(16)            }
(17)            Connection con = null;
(18)
(19)            try {
(20)                    con =
(21)                            (Connection) DriverManager.getConnection(
(22)                                    "jdbc:mysql://localhost/jdbctest/",
(23)                                    "mario",
(24)                                    "thePassword");
(25)            } catch (SQLException e1) {
(26)                    System.err.println("Error establishing database connection");
(27)                    Throwable t = e1;
(28)                    while (t != null) {
(29)                    System.err.println("Type: " + t.getClass().getName());
(30)                    System.err.println("Message: " + t.getMessage());
(31)                    System.err.println("-----");
(32)                    t = t.getCause();
(33)                    }
(34)            }
(35)
(36)            Statement stmt = null;
(37)            try {
(38)                    stmt = (Statement) con.createStatement();
(39)            } catch (SQLException e2) {
(40)                    System.err.println("Error creating SQL-Statement");
(41)                    Throwable t = e2;
(42)                    while (t != null) {
(43)                    System.err.println("Type: " + t.getClass().getName());
(44)                    System.err.println("Message: " + t.getMessage());
(45)                    System.err.println("-----");
(46)                    t = t.getCause();
(47)                    }
(48)            }
(49)
(50)            try {
(51)                    ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(52)                    ResultSetMetaData rsmd = rs.getMetaData();
(53)                    int noColumns = rsmd.getColumnCount();
(54)                    for (int i = 1; i < noColumns; i++) {
(55)                            System.out.print(rsmd.getColumnLabel(i) + "\t");
(56)                    }
(57)                    System.out.println();
(58)
(59)                    while (rs.isLast() == false) {
(60)                            rs.next();
(61)                            for (int i = 1; i < noColumns; i++) {
(62)                                    System.out.print( rs.getObject(i)+"\t" );
(63)                            }
(64)                            System.out.println();
(65)                    }
(66)
(67)            } catch (SQLException e3) {
(68)                    System.err.println("Error selecting values from table EMPLOYEE");
(69)                    Throwable t = e3;
(70)                    while (t != null) {
(71)                    System.err.println("Type: " + t.getClass().getName());
(72)                    System.err.println("Message: " + t.getMessage());
```

```
(73)                    System.err.println("-----");
(74)                    t = t.getCause();
(75)                    }
(76)            }
(77)    }
(78)}
```

Download des Beispiels
Download der Ergebnisdatei

Neben im Beispiel 55 gezeigten Verarbeitung in exakter der Ablagereihenfolge der Datenbank kann auch durch Definition eines Cursors die Traversierung in inverser Ablagerichtung erreicht werden.
Das nachfolgende Beispiel illustriert das entsprechende Vorgehen durch anfängliche Positionierung des Cursors ans Ende der empfangenen Daten (d.h. nach dem letzten Datensatz) und anschließendes schrittweises Rückpositionieren durch Aufruf der Methode `previous`.

Beispiel 56: Auslesen von Daten in invertierter Reihenfolge

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import com.mysql.jdbc.Connection;
(5)import com.mysql.jdbc.Statement;
(6)
(7)public class JDBCSelect5 {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.mysql.jdbc.Driver");
(11)            } catch (ClassNotFoundException e) {
(12)                    System.err.println("Driver class not found");
(13)                    e.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)
(17)            try {
(18)                    con =
(19)                            (Connection) DriverManager.getConnection(
(20)                                    "jdbc:mysql://localhost/jdbctest/",
(21)                                    "mario",
(22)                                    "thePassword");
(23)            } catch (SQLException e1) {
(24)                    System.err.println("Error establishing database connection");
(25)                    Throwable t = e1;
(26)                    while (t != null) {
(27)                    System.err.println("Type: " + t.getClass().getName());
(28)                    System.err.println("Message: " + t.getMessage());
(29)                    System.err.println("-----");
(30)                    t = t.getCause();
(31)                    }
(32)            }
(33)
(34)            Statement stmt = null;
(35)            try {
(36)                    stmt =
(37)                            (Statement) con.createStatement();
(38)            } catch (SQLException e2) {
(39)                    System.err.println("Error creating SQL-Statement");
(40)                    Throwable t = e2;
(41)                    while (t != null) {
(42)                    System.err.println("Type: " + t.getClass().getName());
(43)                    System.err.println("Message: " + t.getMessage());
(44)                    System.err.println("-----");
(45)                    t = t.getCause();
(46)                    }
(47)            }
(48)
(49)            try {
(50)                    ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(51)                    rs.afterLast();
(52)                    while (rs.previous()){
(53)                            System.out.println(rs.getString("FNAME"));
(54)                    }
(55)            } catch (SQLException e3) {
(56)                    System.err.println("Error selecting values from table EMPLOYEE");
(57)                    Throwable t = e3;
(58)                    while (t != null) {
```

```
(59)                    System.err.println("Type: " + t.getClass().getName());
(60)                    System.err.println("Message: " + t.getMessage());
(61)                    System.err.println("-----");
(62)                    t = t.getCause();
(63)                    }
(64)            }
(65)    }
(66)}
```

Download des Beispiels
Download der Ergebnisdatei

Ferner kann der Cursor wahlfrei auf eine beliebige Position der Ergebnisrelation gesetzt werden. Das nachfolgende Beispiel zeigt dies. Ferner illustriert es das Vorgehen zur Größenermittlung des resultierenden `ResultSets` durch das Aufrufpaar `last` und `getRow`, welches zunächst den Cursor auf den letzten aus der Datenbank extrahierten Datensatz positioniert und anschließend dessen Nummer liefert.

Beispiel 57: Auslesen von Daten in wahlfreier Reihenfolge

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import com.mysql.jdbc.Connection;
(5)import com.mysql.jdbc.Statement;
(6)
(7)public class JDBCSelect6 {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.mysql.jdbc.Driver");
(11)            } catch (ClassNotFoundException e) {
(12)                    System.err.println("Driver class not found");
(13)                    e.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)
(17)            try {
(18)                    con =
(19)                            (Connection) DriverManager.getConnection(
(20)                                    "jdbc:mysql://localhost/jdbctest/",
(21)                                    "mario",
(22)                                    "thePassword");
(23)            } catch (SQLException e1) {
(24)                    System.err.println("Error establishing database connection");
(25)                    Throwable t = e1;
(26)                    while (t != null) {
(27)                    System.err.println("Type: " + t.getClass().getName());
(28)                    System.err.println("Message: " + t.getMessage());
(29)                    System.err.println("-----");
(30)                    t = t.getCause();
(31)                    }
(32)            }
(33)
(34)            Statement stmt = null;
(35)            try {
(36)                    stmt = (Statement) con.createStatement();
(37)            } catch (SQLException e2) {
(38)                    System.err.println("Error creating SQL-Statement");
(39)                    Throwable t = e2;
(40)                    while (t != null) {
(41)                    System.err.println("Type: " + t.getClass().getName());
(42)                    System.err.println("Message: " + t.getMessage());
(43)                    System.err.println("-----");
(44)                    t = t.getCause();
(45)                    }
(46)            }
(47)
(48)            try {
(49)                    int position = 0;
(50)                    ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(51)                    rs.last();
(52)                    int size = rs.getRow();
(53)                    for (int i = 0; i < size; i++) {
(54)                            position = (position + 3) % size;
(55)                            rs.absolute(position + 1);
(56)                            System.out.println(
```

```
(57)                                    "position=" + (position + 1) + ": " + rs.getString
("FNAME"));
(58)                    }
(59)            } catch (SQLException e3) {
(60)                    System.err.println("Error selecting values from table EMPLOYEE");
(61)                    Throwable t = e3;
(62)                    while (t != null) {
(63)                    System.err.println("Type: " + t.getClass().getName());
(64)                    System.err.println("Message: " + t.getMessage());
(65)                    System.err.println("-----");
(66)                    t = t.getCause();
(67)                    }
(68)            }
(69)    }
(70)}
```

Download des Beispiels
Download der Ergebnisdatei

Wird der benötigte `ResultSet` geeignet (d.h. mit den Parameter `CONCUR_UPDATABLE`) (siehe Zeile 49) initialisiert, so können Änderungen, die im Hauptspeicher durch die JDBC-API durchgeführt werden, in die Datenbank persistiert werden.
Beispiel 58 zeigt dies exemplarisch für den Einfügevorgang eines neuen Tupels.

Die Voraussetzungen für Einfüge- und Aktualisierungsvorgänge entstprechen denen von *updatable views*, d.h. die Daten dürfen nur aus genau einer Tabelle entnommen sein und müssen den Primärschlüssel enthalten.

Beispiel 58: Auslesen und Einfügen von Daten

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.ResultSetMetaData;
(4)import java.sql.SQLException;
(5)import java.sql.Statement;
(6)
(7)import com.mysql.jdbc.Connection;
(8)
(9)public class JDBCSelect2 {
(10)    private static void printResultSet(ResultSet rs) throws SQLException {
(11)            ResultSetMetaData rsmd = rs.getMetaData();
(12)            int noColumns = rsmd.getColumnCount();
(13)            for (int i = 1; i < noColumns; i++) {
(14)                    System.out.print(rsmd.getColumnLabel(i) + "\t");
(15)            }
(16)            System.out.println();
(17)
(18)            while (rs.isLast() == false) {
(19)                    rs.next();
(20)                    for (int i = 1; i < noColumns; i++) {
(21)                            System.out.print( rs.getObject(i)+"\t" );
(22)                    }
(23)                    System.out.println();
(24)            }
(25)
(26)    }
(27)    public static void main(String[] args) {
(28)            try {
(29)                    Class.forName("com.mysql.jdbc.Driver");
(30)            } catch (ClassNotFoundException e) {
(31)                    System.err.println("Driver class not found");
(32)                    e.printStackTrace();
(33)            }
(34)            Connection con = null;
(35)
(36)            try {
(37)                    con =
(38)                            (Connection) DriverManager.getConnection(
(39)                                    "jdbc:mysql://localhost/jdbctest/",
(40)                                    "mario",
(41)                                    "thePassword");
(42)            } catch (SQLException e1) {
(43)                    System.err.println("Error establishing database connection");
(44)                    Throwable t = e1;
(45)                    while (t != null) {
```

```
(46)                    System.err.println("Type: " + t.getClass().getName());
(47)                    System.err.println("Message: " + t.getMessage());
(48)                    System.err.println("-----");
(49)                    t = t.getCause();
(50)                    }
(51)            }
(52)
(53)            Statement stmt = null;
(54)            try {
(55)                    stmt = (Statement) con.createStatement(ResultSet.
TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
(56)            } catch (SQLException e2) {
(57)                    System.err.println("Error creating SQL-Statement");
(58)                    Throwable t = e2;
(59)                    while (t != null) {
(60)                    System.err.println("Type: " + t.getClass().getName());
(61)                    System.err.println("Message: " + t.getMessage());
(62)                    System.err.println("-----");
(63)                    t = t.getCause();
(64)                    }
(65)            }
(66)
(67)            try {
(68)                    ResultSet uprs = (ResultSet) stmt.executeQuery("SELECT * FROM
EMPLOYEE;");
(69)                    printResultSet(uprs);
(70)                    uprs.moveToInsertRow();
(71)                    uprs.updateString("FNAME","Mario");
(72)                    uprs.updateString("LNAME","Jeckle");
(73)                    uprs.updateInt("SSN",111111111);
(74)                    uprs.insertRow();
(75)                    uprs = (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(76)                    printResultSet(uprs);
(77)            } catch (SQLException e3) {
(78)                    System.err.println("Error selecting values from table EMPLOYEE");
(79)                    Throwable t = e3;
(80)                    while (t != null) {
(81)                    System.err.println("Type: " + t.getClass().getName());
(82)                    System.err.println("Message: " + t.getMessage());
(83)                    System.err.println("-----");
(84)                    t = t.getCause();
(85)                    }
(86)            }
(87)    }
(88)}
```

Download des Beispiels
Download der Ergebnisdatei

Auf dieselbe Weise können auch Tupel einer Relation verändert werden. Hierzu stehen eine Reihe von updateXXX-Methoden zur Verfügung, wobei XXX für den Typ des zu aktualisierenden Attributs steht.
Nach durchgeführter Modifikation der hauptspeicherresidenten Werte werden diese durch updateRow in die Datenbank rückgeschrieben.
Beispiel 59 zeigt dies:

Beispiel 59: Modifizieren von Daten

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import java.sql.Statement;
(5)
(6)import com.mysql.jdbc.Connection;
(7)
(8)public class JDBCSelect3 {
(9)     public static void main(String[] args) {
(10)            try {
(11)                    Class.forName("com.mysql.jdbc.Driver");
(12)            } catch (ClassNotFoundException e) {
(13)                    System.err.println("Driver class not found");
(14)                    e.printStackTrace();
(15)            }
(16)            Connection con = null;
(17)
(18)            try {
```

```
(19)                    con =
(20)                            (Connection) DriverManager.getConnection(
(21)                                    "jdbc:mysql://localhost/jdbctest/",
(22)                                    "mario",
(23)                                    "thePassword");
(24)            } catch (SQLException e1) {
(25)                    System.err.println("Error establishing database connection");
(26)                    Throwable t = e1;
(27)                    while (t != null) {
(28)                    System.err.println("Type: " + t.getClass().getName());
(29)                    System.err.println("Message: " + t.getMessage());
(30)                    System.err.println("-----");
(31)                    t = t.getCause();
(32)                    }
(33)            }
(34)
(35)            Statement stmt = null;
(36)            try {
(37)                    stmt =
(38)                            (Statement) con.createStatement(
(39)                                    ResultSet.TYPE_SCROLL_SENSITIVE,
(40)                                    ResultSet.CONCUR_UPDATABLE);
(41)            } catch (SQLException e2) {
(42)                    System.err.println("Error creating SQL-Statement");
(43)                    Throwable t = e2;
(44)                    while (t != null) {
(45)                    System.err.println("Type: " + t.getClass().getName());
(46)                    System.err.println("Message: " + t.getMessage());
(47)                    System.err.println("-----");
(48)                    t = t.getCause();
(49)                    }
(50)            }
(51)
(52)            try {
(53)                    ResultSet uprs =
(54)                            (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(55)                    int namePos = uprs.findColumn("LNAME");
(56)
(57)                    while (uprs.isLast() == false) {
(58)                            uprs.next();
(59)                            if (uprs.getString(namePos).compareTo("Wallace") == 0) {
(60)                                    uprs.updateString(namePos, "Doe");
(61)                                    uprs.updateRow();
(62)                            }
(63)                    }
(64)
(65)            } catch (SQLException e3) {
(66)                    System.err.println("Error selecting values from table EMPLOYEE");
(67)                    Throwable t = e3;
(68)                    while (t != null) {
(69)                    System.err.println("Type: " + t.getClass().getName());
(70)                    System.err.println("Message: " + t.getMessage());
(71)                    System.err.println("-----");
(72)                    t = t.getCause();
(73)                    }
(74)            }
(75)    }
(76)}
```

Download des Beispiels

Analog vollzieht sich der Löschvorgang mittels `deleteRow`:

Beispiel 60: Löschen von Daten

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import java.sql.Statement;
(5)
(6)import com.mysql.jdbc.Connection;
(7)
(8)public class JDBCSelect4 {
(9)     public static void main(String[] args) {
(10)            try {
(11)                    Class.forName("com.mysql.jdbc.Driver");
(12)            } catch (ClassNotFoundException e) {
(13)                    System.err.println("Driver class not found");
(14)                    e.printStackTrace();
(15)            }
(16)            Connection con = null;
(17)
(18)            try {
(19)                    con =
(20)                            (Connection) DriverManager.getConnection(
(21)                                    "jdbc:mysql://localhost/jdbctest/",
(22)                                    "mario",
(23)                                    "thePassword");
(24)            } catch (SQLException e1) {
(25)                    System.err.println("Error establishing database connection");
(26)                    Throwable t = e1;
(27)                    while (t != null) {
(28)                    System.err.println("Type: " + t.getClass().getName());
(29)                    System.err.println("Message: " + t.getMessage());
(30)                    System.err.println("-----");
(31)                    t = t.getCause();
(32)                    }
(33)            }
(34)
(35)            Statement stmt = null;
(36)            try {
(37)                    stmt =
(38)                            (Statement) con.createStatement(
(39)                                    ResultSet.TYPE_SCROLL_SENSITIVE,
(40)                                    ResultSet.CONCUR_UPDATABLE);
(41)            } catch (SQLException e2) {
(42)                    System.err.println("Error creating SQL-Statement");
(43)                    Throwable t = e2;
(44)                    while (t != null) {
(45)                    System.err.println("Type: " + t.getClass().getName());
(46)                    System.err.println("Message: " + t.getMessage());
(47)                    System.err.println("-----");
(48)                    t = t.getCause();
(49)                    }
(50)            }
(51)
(52)            try {
(53)                    ResultSet uprs =
(54)                            (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(55)                    int namePos = uprs.findColumn("LNAME");
(56)
(57)                    while (uprs.isLast() == false) {
(58)                            uprs.next();
(59)                            if (uprs.getString(namePos).compareTo("Smith") == 0) {
(60)                                    uprs.deleteRow();
(61)                            }
(62)                    }
(63)
(64)            } catch (SQLException e3) {
(65)                    System.err.println("Error selecting values from table EMPLOYEE");
(66)                    Throwable t = e3;
(67)                    while (t != null) {
(68)                    System.err.println("Type: " + t.getClass().getName());
(69)                    System.err.println("Message: " + t.getMessage());
(70)                    System.err.println("-----");
(71)                    t = t.getCause();
(72)                    }
(73)            }
(74)    }
(75)}
```

Download des Beispiels

Die bisher betrachteten Varianten extrahieren Daten aus der Datenbank im Stile einer Momentaufnahme (*snapshot*) zum Zeitpunkt der Anfrage. Die einmal angefragten Inhalte können sich jedoch noch zur Laufzeit der zugreifenden JDBC-Applikation datenbankseitig ändern, wenn sie durch eine andere Applikation neu geschrieben werden. Zur Gewährleistung der Konsistenz des extrahierten Snapshots mit den tatsächlichen Datenbankinhalten steht die Operation `rowUpdated` zur Verfügung. Sie ermittelt ob der im Hauptspeicher befindliche Wert mit dem aktuellen Datenbankinhalt übereinstimmt, d.h. ob der DB-Inhalt aktualisiert wurde.
Beispiel 61 zeigt ein Umsetzungsbeispiel.

Beispiel 61: Test auf geänderte Daten

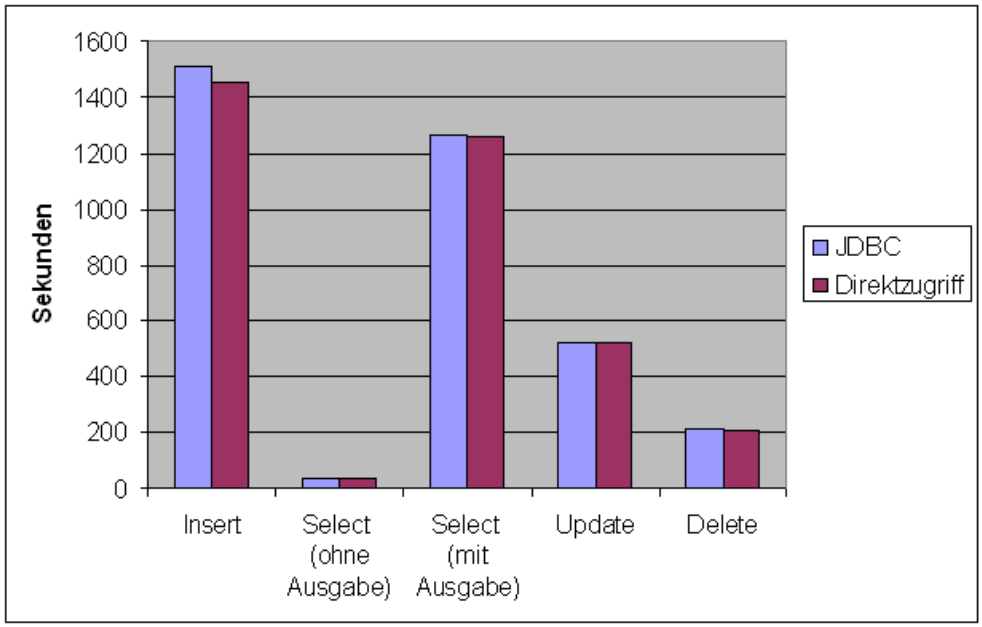
```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import com.mysql.jdbc.Connection;
(5)import com.mysql.jdbc.Statement;
(6)
(7)public class JDBCSelect7 {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.mysql.jdbc.Driver");
(11)            } catch (ClassNotFoundException cnfe) {
(12)                    System.err.println("Driver class not found");
(13)                    cnfe.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)
(17)            try {
(18)                    con =
(19)                            (Connection) DriverManager.getConnection(
(20)                                    "jdbc:mysql://localhost/jdbctest/",
(21)                                    "mario",
(22)                                    "thePassword");
(23)            } catch (SQLException e) {
(24)                    System.err.println("Error establishing database connection");
(25)                    Throwable t = e;
(26)                    while (t != null) {
(27)                    System.err.println("Type: " + t.getClass().getName());
(28)                    System.err.println("Message: " + t.getMessage());
(29)                    System.err.println("-----");
(30)                    t = t.getCause();
(31)                    }
(32)            }
(33)
(34)            Statement stmt = null;
(35)            try {
(36)                    stmt =
(37)                            (Statement) con.createStatement(
(38)                                    ResultSet.TYPE_SCROLL_SENSITIVE,
(39)                                    ResultSet.CONCUR_UPDATABLE);
(40)            } catch (SQLException e) {
(41)                    System.err.println("Error creating SQL-Statement");
(42)                    Throwable t = e;
(43)                    while (t != null) {
(44)                    System.err.println("Type: " + t.getClass().getName());
(45)                    System.err.println("Message: " + t.getMessage());
(46)                    System.err.println("-----");
(47)                    t = t.getCause();
(48)                    }
(49)            }
(50)
(51)            try {
(52)                    ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(53)                    rs.absolute(5);
(54)                    System.out.println(rs.getString("FNAME"));
(55)
(56)                    System.out.println("sleeping ...");
(57)                    Thread.sleep(6000);
(58)                    System.out.println("awake ...");
(59)
(60)                    if (rs.rowUpdated() == true) {
(61)                            rs.refreshRow();
(62)                            System.out.println(rs.getString("FNAME"));
```

```
(63)                    }
(64)
(65)            } catch (SQLException e) {
(66)                    System.err.println("Error selecting values from table EMPLOYEE");
(67)                    Throwable t = e;
(68)                    while (t != null) {
(69)                    System.err.println("Type: " + t.getClass().getName());
(70)                    System.err.println("Message: " + t.getMessage());
(71)                    System.err.println("-----");
(72)                    t = t.getCause();
(73)                    }
(74)            } catch (InterruptedException ie) {
(75)                    ie.printStackTrace();
(76)            }
(77)    }
(78)}
```

Download des Beispiels

Performancebetrachtungen

Abbildung 5: *JDBC-Geschwindigkeitsvergleich*



(click on image to enlarge!)

Die Abbildung zeigt die Ergebnisse einiger Geschwindigkeitsmessungen als Vergleich zwischen dem Zugriff auf eine MySQL-Datenbank unter Nutzung der Textschnittstelle und der Abwicklung derselben Zugriffe mittels JDBC.

Zur Messung wurde eine nicht-indexierte Datenbank mit 10^7 Einträgen verwendet die aus der Relation `tab` bestand. Deren Tupel wurden aus Paaren von 36-Byte großen UUIDs gemäß dem Spezifikationsentwurf der IETF gebildet.

Zur Zeitmessung wurden folgende Einzeloperationen betrachtet:

- **Insert**: `INSERT INTO tab VALUES(...)`
  Die Werte wurden unter Deaktivierung des Autocommit sequentiell eingefügt.
- **Select (ohne Ausgabe)**: `SELECT COUNT(*) FROM tab`
  Die ermittelte Gesamtzahl wurde nicht am Bildschirm ausgegeben.
- **Select (mit Ausgabe)**: `SELECT * FROM tab`
  Die Resultate wurden durch den Standarclient bzw. eine selbsterstellte (textmode) Java-Implementierung ausgegeben.
- **Update**: `UPDATE tab SET UUID1="X" WHERE UUID1<>"X"`
  Durch die Initialisierung der Werte mit UUID-Einträgen wird sichergestellt, daß alle Tupel aktualisiert werden, da sie in keinem Fall den Wert `X` enthalten.
- **Delete**: `DELETE FROM tab WHERE UUID2<>"X"`
  Durch die Initialisierung der Werte mit UUID-Einträgen wird sichergestellt, daß alle Tupel aktualisiert werden, da sie in keinem Fall den Wert `X` enthalten.

Insgesamt zeigt sich ein ausgewogenes Bild, in welchem der JDBC-Zugriff lediglich bei datenintensiven Zugriffen (große Mengen schreibender Zugriffe bei `INSERT` bzw. große Mengen lesender Operationen bei `SELECT`) im Bereich von fünf Prozent zurückliegt.

Diese enge Vergleichbarkeit der beiden Zugriffsmodi rührt von den Realisierung des eingesetzten JDBC-Treibers her; insbesondere von der Handhabung der physischen Datenbankverbindung auf Ebene des Netzwerkprotokolls.

SQL3-Datentypen

Die JDBC-API unterstützt mit Zugriffsmethoden auf die Datentypen `BLOB`, `CLOB`, `ARRAY`, `Object` und `Ref` bereits eine Untermenge des SQL:1999-Standards. So können, vorausgesetzt das durch JDBC angesprochene DBMS unterstützt dies, große unstrukturierte Binär- oder Textdaten sowie einfache verschachtelte Tabellen, mithin NF^2-Strukturen verwaltet werden.

Beispiel 62 zeigt den Zugriff auf ein als eingebettete Tabelle realisiertes mengenwertiges Attribut. Die Beispieldatenbank wurde hierfür wie folgt modifiziert:

```
alter table EMPLOYEE ADD CAR SET('53M91','521R4', 'LLO415', 'XNU457');
update EMPLOYEE set CAR='XNU457' where SSN=123456789;
update EMPLOYEE set CAR='XNU457,521R4'  where SSN="999887777";
```

Beispiel 62: Zugriff auf ein mengenwertiges Attribut

```
(1)import java.sql.Array;
(2)import java.sql.DriverManager;
(3)import java.sql.ResultSet;
(4)import java.sql.SQLException;
(5)import com.mysql.jdbc.Connection;
(6)import com.mysql.jdbc.Statement;
(7)
(8)public class JDBCSelect8 {
(9)     public static void main(String[] args) {
(10)            try {
(11)                    Class.forName("com.mysql.jdbc.Driver");
(12)            } catch (ClassNotFoundException cnfe) {
(13)                    System.err.println("Driver class not found");
(14)                    cnfe.printStackTrace();
(15)            }
(16)            Connection con = null;
(17)
(18)            try {
(19)                    con =
(20)                            (Connection) DriverManager.getConnection(
(21)                                    "jdbc:mysql://localhost/jdbctest/",
(22)                                    "mario",
(23)                                    "thePassword");
(24)            } catch (SQLException sqle) {
(25)                    System.err.println("Error establishing database connection");
(26)                    Throwable t = sqle;
(27)                    while (t != null) {
(28)                    System.err.println("Type: " + t.getClass().getName());
(29)                    System.err.println("Message: " + t.getMessage());
(30)                    System.err.println("-----");
(31)                    t = t.getCause();
(32)                    }
(33)            }
(34)
(35)            Statement stmt = null;
(36)            try {
(37)                    stmt = (Statement) con.createStatement();
(38)            } catch (SQLException sqle) {
(39)                    System.err.println("Error creating SQL-Statement");
(40)                    Throwable t = sqle;
(41)                    while (t != null) {
(42)                    System.err.println("Type: " + t.getClass().getName());
(43)                    System.err.println("Message: " + t.getMessage());
(44)                    System.err.println("-----");
(45)                    t = t.getCause();
(46)                    }
(47)            }
(48)
(49)            try {
(50)                    ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(51)                    while (!rs.isLast()) {
(52)                            rs.first();
(53)                            System.out.print(rs.getString("FNAME") + "\t");
(54)                            Array cars = rs.getArray("CAR");
```

```
(55)                            ResultSet carsRS = cars.getResultSet();
(56)                            System.out.print("(");
(57)                            while (!carsRS.isLast()) {
(58)                                    rs.first();
(59)                                    System.out.print(carsRS.getString("CAR"));
(60)                                    carsRS.next();
(61)                            }
(62)                            System.out.println(")");
(63)                            rs.next();
(64)                    }
(65)            } catch (SQLException sqle) {
(66)                    System.err.println("Error selecting values from table EMPLOYEE");
(67)                    Throwable t = sqle;
(68)                    while (t != null) {
(69)                    System.err.println("Type: " + t.getClass().getName());
(70)                    System.err.println("Message: " + t.getMessage());
(71)                    System.err.println("-----");
(72)                    t = t.getCause();
(73)                    }
(74)            }
(75)    }
(76)}
```

Download des Beispiels

Das Beispiel unterstreicht die Rolle der mengenwertigen Attribute als eingebettete Tabellen. So erfolgt der Zugriff auf die Einzelwerte des Attributs `CAR` identisch zur Ermittlung der Resultatmenge der SQL-Anfrage mittels `getResultSet`. Auch die Traversierung der einzelnen `CAR`-Elemente erfolgt äquivalent.

Die Aufnahme der *large objects* in ihrer Ausprägungsform als *Character Large Objects* (CLOB) oder *Binary Large Objects* (BLOB) stellen eine der zentralen Erweiterungen des SQL:1999-Standards gegenüber seinen Vorgängern dar.
Zwar ist die Ablage großer unstrukturierter Datenobjekte in relationalen Datenbanken konzeptionell durchaus diskussionswert, jedoch in der Praxis oftmals, trotz der teilweise erheblichen Geschwindigkeitseinbußen im Zugriff (so benötigt die Ausführung der Beispielapplikation mit einem 10^6 Byte großen Datenstrom 1,1 Sekunden, während dieselbe Operation dateisystembasiert in 0,1 Sekunde abläuft), gewünscht.
Beispiel 63 zeigt die notwendigen Schritte zur Ablage und erneuten Auslese eines aus einer Datei gewonnen Binärdatenstroms in der Datenbank.
Die Beispieldatenbank wurde hierfür um ein Attribut zur Aufnahme binärer Daten erweitert:
`ALTER TABLE EMPLOYEE ADD binData blob;`

Beispiel 63: Verarbeitung unstrukturierter Binärdaten

```
(1)import java.io.File;
(2)import java.io.FileInputStream;
(3)import java.io.FileOutputStream;
(4)import java.io.IOException;
(5)import java.sql.DriverManager;
(6)import java.sql.PreparedStatement;
(7)import java.sql.ResultSet;
(8)import java.sql.SQLException;
(9)
(10)import com.mysql.jdbc.Connection;
(11)import com.mysql.jdbc.Statement;
(12)
(13)public class JDBCSelect9 {
(14)    public static void main(String[] args) {
(15)            try {
(16)                    Class.forName("com.mysql.jdbc.Driver");
(17)            } catch (ClassNotFoundException cnfe) {
(18)                    System.err.println("Driver class not found");
(19)                    cnfe.printStackTrace();
(20)            }
(21)            Connection con = null;
(22)
(23)            try {
(24)                    con =
(25)                            (Connection) DriverManager.getConnection(
(26)                                    "jdbc:mysql://localhost/jdbctest/",
(27)                                    "mario",
(28)                                    "thePassword");
(29)            } catch (SQLException sqle) {
(30)                    System.err.println("Error establishing database connection");
(31)                    Throwable t = sqle;
```

```
(32)                    while (t != null) {
(33)                    System.err.println("Type: " + t.getClass().getName());
(34)                    System.err.println("Message: " + t.getMessage());
(35)                    System.err.println("-----");
(36)                    t = t.getCause();
(37)                    }
(38)            }
(39)
(40)            try {
(41)                    File file = new File(args[0]);
(42)                    FileInputStream fis = new FileInputStream(args[0]);
(43)                    PreparedStatement pstmt =
(44)                            con.prepareStatement(
(45)                                    "UPDATE EMPLOYEE SET binData =? WHERE
SSN=123456789");
(46)                    pstmt.setBinaryStream(1, fis, (int) file.length());
(47)                    pstmt.executeUpdate();
(48)                    fis.close();
(49)
(50)                    //read it back from the database
(51)                    Statement stmt = (Statement) con.createStatement();
(52)                    ResultSet rs =
(53)                            stmt.executeQuery(
(54)                                    "SELECT binData FROM EMPLOYEE WHERE
SSN='123456789';");
(55)
(56)                    FileOutputStream fos = new FileOutputStream(args[1]);
(57)                    if (rs.next())
(58)                            fos.write(rs.getBytes(1));
(59)                    fos.close();
(60)
(61)            } catch (SQLException sqle) {
(62)                    System.err.println("Error selecting values from table EMPLOYEE");
(63)                    Throwable t = sqle;
(64)                    while (t != null) {
(65)                    System.err.println("Type: " + t.getClass().getName());
(66)                    System.err.println("Message: " + t.getMessage());
(67)                    System.err.println("-----");
(68)                    t = t.getCause();
(69)                    }
(70)            } catch (IOException ioe) {
(71)                    ioe.printStackTrace();
(72)            }
(73)    }
(74)}
```

Download des Beispiels

Die Binärdaten können naturgemäß nicht direkt in die SQL-UPDATE-Anweisung eingebunden werden, sie werden daher einer mittels `prepareStatement` vorerzeugten Anweisung durch Aufruf der Methode `setBinaryStream` übergeben.

Transaktionssteuerung

Zur Steuerung des transaktionalen Verhaltens einer JDBC-Anfrage bietet die Klasse `Connection` verschiedene Methoden an:

- Abfrage der aktuellen Isolationsstufe: `getIsolationLevel`.
  Hierbei werden fünf Stufen unterschieden:
  - `TRANSACTION_NONE`: Keinerlei Transaktionsunterstützung
  - `TRANSACTION_READ_UNCOMMITTED`: Auch nicht durch `commit` freigegebene Daten werden gelesen. Es können daher *dirty reads*, Nicht-wiederholbare- und Phantomlesevorgänge auftreten.
  - `TRANSACTION_READ_COMMITTED`: Nur durch `commit` freigegebene Daten werden gelesen. nichtwiederholbare- und Phantomlesevorgänge können jedoch auftreten.
  - `TRANSACTION_REPEATABLE_READ`: Innerhalb einer Transaktion können die verarbeiteten Daten nicht durch eine andere Transaktion verändert werden. Das Auftreten von *dirty reads* und nichtwiederholbaren Lesevorgängen ist daher ausgeschlossen, Phantomlesevorgänge sind jedoch weiterhin möglich.
  - `TRANSACTION_SERIALIZABLE`: Strikte Isolation aller Transaktionen, auf dieser Stufe sind auch Phantomlesevorgänge ausgeschlossen.

  Jede Stufe geht zwar mit einer gesteigerten Qualität der durch eine Transaktion verarbeiteten Daten einher, jedoch senkt gleichzeitig eine strikterere Isolationsstufe die Anzahl der gleichzeitigen Zugriffe auf die Datenbank und damit die Gesamtsystemperformance.

- Setzen der Isolationsstufe: `setTransactionIsolation`
- (De-)Aktivierung der automatischen Freigabe: `setAutoCommit`. Die Übergabe von `true` bewirkt die Aktivierung des Modus bei dem jede Einzelanweisung sofort persistent übernommen und für andere Transaktionen sichtbar wird.
  Standardmäßig ist diese Option aktiviert. Ihr aktueller Zustand kann per `getAutoCommit` ermittelt werden.
- Freigabe von Änderungen: `commit`.
- Rücknahme von Änderungen: `rollback`.
- Rücknahme von Änderungen bis zu definiertem Sicherungspunkt: `rollback(Savepoint s)`.
- Setzen eines Sicherungspunktes: `setSavepoint`.

Beispiel 1 zeigt die Nutzung des Transaktionskonzepts.
Zunächst wird die aktuelle Isolationsstufe ermittelt und geprüft ob das angesprochene DBMS die höchste durch JDBC vorhergesehene Isolationsstufe unterstützt.
Nach Abschaltung der automatischen Änderungsübernahme (`setAutoCommit(false)`) werden zunächst zwei Tupel in die Tabelle `EMPLOYEE` eingefügt, die jedoch nur innerhalb der laufenden Transaktion sichtbar werden, für alle anderen Transaktionen innerhalb des DBMS bleiben die neuen Werte (zunächst) unsichtbar.
Eine angenommene Fehlersituation führt zum Rücksetzen der Transaktion durch (`rollback`).
Nach Abschluß des Programms wurden zwar die beiden ersten Werte lokal in die Datenbank übernommen, aber noch innerhalb der laufenden Transaktion wieder daraus entfernt, weshalb sie zu keinem Zeitpunkt für andere Datenbankbenutzer sichtbar waren.

Beispiel 64: Transaktionsverarbeitung

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)
(5)import com.mysql.jdbc.Connection;
(6)import com.mysql.jdbc.Statement;
(7)
(8)public class JDBCTransact1 {
(9)     private static void printContent(Statement stmt) throws SQLException {
(10)            ResultSet rs =
(11)                    stmt.executeQuery("SELECT FNAME,MINIT,LNAME FROM EMPLOYEE;");
(12)            while (!rs.isLast()) {
(13)                    rs.next();
(14)                    System.out.println(
(15)                            rs.getString("LNAME")
(16)                                    + "\t"
(17)                                    + rs.getString("MINIT")
(18)                                    + "\t"
(19)                                    + rs.getString("LNAME"));
(20)            }
(21)    }
(22)    public static void main(String[] args) {
(23)            try {
(24)                    Class.forName("com.mysql.jdbc.Driver");
(25)            } catch (ClassNotFoundException cnfe) {
(26)                    System.err.println("Driver class not found");
(27)                    cnfe.printStackTrace();
(28)            }
(29)            Connection con = null;
(30)
(31)            try {
(32)                    con =
(33)                            (Connection) DriverManager.getConnection(
(34)                                    "jdbc:mysql://localhost/jdbctest/",
(35)                                    "mario",
(36)                                    "thePassword");
(37)            } catch (SQLException sqle) {
(38)                    System.err.println("Error establishing database connection");
(39)                    Throwable t = sqle;
(40)                    while (t != null) {
(41)                    System.err.println("Type: " + t.getClass().getName());
(42)                    System.err.println("Message: " + t.getMessage());
(43)                    System.err.println("-----");
(44)                    t = t.getCause();
(45)                    }
(46)            }
(47)
(48)            Statement stmt = null;
(49)            try {
(50)                    stmt = (Statement) con.createStatement();
(51)            } catch (SQLException sqle) {
(52)                    System.err.println("Error creating SQL-Statement");
```

```
(53)                    Throwable t = sqle;
(54)                    while (t != null) {
(55)                    System.err.println("Type: " + t.getClass().getName());
(56)                    System.err.println("Message: " + t.getMessage());
(57)                    System.err.println("-----");
(58)                    t = t.getCause();
(59)                    }
(60)            }
(61)
(62)            try {
(63)                    int transactionIsolation = con.getTransactionIsolation();
(64)                    switch (transactionIsolation) {
(65)                            case Connection.TRANSACTION_NONE :
(66)                                    System.out.println("Transactions are not
supported");
(67)                                    break;
(68)                            case Connection.TRANSACTION_READ_UNCOMMITTED :
(69)                                    System.out.println(
(70)                                            "Dirty reads, non-repeatable reads and
phantom reads can occur");
(71)                                    break;
(72)                            case Connection.TRANSACTION_READ_COMMITTED :
(73)                                    System.out.println(
(74)                                            "Dirty reads are prevented; non-repeatable
reads and phantom reads can occur");
(75)                                    break;
(76)                            case Connection.TRANSACTION_REPEATABLE_READ :
(77)                                    System.out.println(
(78)                                            "Dirty reads and non-repeatable reads are
prevented; phantom reads can occur");
(79)                                    break;
(80)                            case Connection.TRANSACTION_SERIALIZABLE :
(81)                                    System.out.println(
(82)                                            "Dirty reads, non-repeatable reads and
phantom reads are prevented");
(83)                                    break;
(84)                    }
(85)                    if (transactionIsolation < Connection.TRANSACTION_SERIALIZABLE) {
(86)                            con.setTransactionIsolation(
(87)                                    Connection.TRANSACTION_SERIALIZABLE);
(88)                            if (con.getTransactionIsolation()
(89)                                    != Connection.TRANSACTION_SERIALIZABLE) {
(90)                                    System.out.println(
(91)                                            "cannot set Connection.
TRANSACTION_SERIALIZABLE");
(92)                            } else {
(93)                                    System.out.println(
(94)                                            "reached highest possible isolation
level");
(95)                            }
(96)                    }
(97)
(98)                    con.setAutoCommit(false);
(99)                    stmt.executeUpdate(
(100)                           "INSERT INTO EMPLOYEE VALUES
('Hans','X','Hinterhuber','111111111',NULL,NULL,NULL,NULL,NULL,NULL);");
(101)                   stmt.executeUpdate(
(102)                           "INSERT INTO EMPLOYEE VALUES
('Franz','X','Obermüller','222222222',NULL,NULL,NULL,NULL,NULL,NULL);");
(103)                   printContent(stmt);
(104)                   //suppose error happens here
(105)                   Thread.sleep(5000);
(106)                   boolean error = true;
(107)                   if (error) {
(108)                           con.rollback();
(109)                   } else {
(110)                           stmt.executeUpdate(
(111)                                   "INSERT INTO EMPLOYEE VALUES
('Fritz','X','Meier','333333333',NULL,NULL,NULL,NULL,NULL,NULL);");
(112)                   }
(113)                   printContent(stmt);
(114)           } catch (SQLException sqle) {
(115)                   Throwable t = sqle;
(116)                   while (t != null) {
(117)                           System.err.println("Type: " + t.getClass().getName());
(118)                   System.err.println("Message: " + t.getMessage());
```

```
(119)                   System.err.println("-----");
(120)                   t = t.getCause();
(121)                   }
(122)           } catch (InterruptedException ie) {
(123)                   ie.printStackTrace();
(124)           }
(125)   }
(126)}
```

Download des Beispiels
Download der Ergebnisdatei

Neben dem Zurücksetzen einer vollständigen Transaktion bietet die JDBC-API auch die Möglichkeit alle Schritte bis zu einem anwenderdefierten aus der Datenbank zu entfernen.
Beispiel 65 zeigt dies unter Verwendung der Methode `setSavepoint` zur Definition eines Sicherungspunktes und `rollback(sp)` zum Zurücksetzen bis zu diesem Sicherungspunkt.

Beispiel 65: Transaktionsverarbeitung mit Sicherungspunkten

```
(1)import java.sql.DriverManager;
(2)import java.sql.ResultSet;
(3)import java.sql.SQLException;
(4)import java.sql.Savepoint;
(5)
(6)import com.mysql.jdbc.Connection;
(7)import com.mysql.jdbc.Statement;
(8)
(9)public class JDBCTransact2 {
(10)    private static void printContent(Statement stmt) throws SQLException {
(11)            ResultSet rs =
(12)                    stmt.executeQuery("SELECT FNAME,MINIT,LNAME FROM EMPLOYEE;");
(13)            while (!rs.isLast()) {
(14)                    rs.next();
(15)                    System.out.println(
(16)                            rs.getString("LNAME")
(17)                                    + "\t"
(18)                                    + rs.getString("MINIT")
(19)                                    + "\t"
(20)                                    + rs.getString("LNAME"));
(21)            }
(22)    }
(23)    public static void main(String[] args) {
(24)            try {
(25)                    Class.forName("com.mysql.jdbc.Driver");
(26)            } catch (ClassNotFoundException cnfe) {
(27)                    System.err.println("Driver class not found");
(28)                    cnfe.printStackTrace();
(29)            }
(30)            Connection con = null;
(31)
(32)            try {
(33)                    con =
(34)                            (Connection) DriverManager.getConnection(
(35)                                    "jdbc:mysql://localhost/jdbctest/",
(36)                                    "mario",
(37)                                    "thePassword");
(38)            } catch (SQLException sqle) {
(39)                    System.err.println("Error establishing database connection");
(40)                    Throwable t = sqle;
(41)                    while (t != null) {
(42)                    System.err.println("Type: " + t.getClass().getName());
(43)                    System.err.println("Message: " + t.getMessage());
(44)                    System.err.println("-----");
(45)                    t = t.getCause();
(46)                    }
(47)            }
(48)
(49)            Statement stmt = null;
(50)            try {
(51)                    stmt = (Statement) con.createStatement();
(52)            } catch (SQLException sqle) {
(53)                    System.err.println("Error creating SQL-Statement");
(54)                    Throwable t = sqle;
(55)                    while (t != null) {
(56)                    System.err.println("Type: " + t.getClass().getName());
(57)                    System.err.println("Message: " + t.getMessage());
```

```
(58)                    System.err.println("-----");
(59)                    t = t.getCause();
(60)                    }
(61)            }
(62)
(63)            try {
(64)                    int transactionIsolation = con.getTransactionIsolation();
(65)                    switch (transactionIsolation) {
(66)                            case Connection.TRANSACTION_NONE :
(67)                                    System.out.println("Transactions are not
supported");
(68)                                    break;
(69)                            case Connection.TRANSACTION_READ_UNCOMMITTED :
(70)                                    System.out.println(
(71)                                            "Dirty reads, non-repeatable reads and
phantom reads can occur");
(72)                                    break;
(73)                            case Connection.TRANSACTION_READ_COMMITTED :
(74)                                    System.out.println(
(75)                                            "Dirty reads are prevented; non-repeatable
reads and phantom reads can occur");
(76)                                    break;
(77)                            case Connection.TRANSACTION_REPEATABLE_READ :
(78)                                    System.out.println(
(79)                                            "Dirty reads and non-repeatable reads are
prevented; phantom reads can occur");
(80)                                    break;
(81)                            case Connection.TRANSACTION_SERIALIZABLE :
(82)                                    System.out.println(
(83)                                            "Dirty reads, non-repeatable reads and
phantom reads are prevented");
(84)                                    break;
(85)                    }
(86)                    if (transactionIsolation < Connection.TRANSACTION_SERIALIZABLE) {
(87)                            con.setTransactionIsolation(
(88)                                    Connection.TRANSACTION_SERIALIZABLE);
(89)                            if (con.getTransactionIsolation()
(90)                                    != Connection.TRANSACTION_SERIALIZABLE) {
(91)                                    System.out.println(
(92)                                            "cannot set Connection.
TRANSACTION_SERIALIZABLE");
(93)                            } else {
(94)                                    System.out.println(
(95)                                            "reached highest possible isolation
level");
(96)                            }
(97)                    }
(98)
(99)                    con.setAutoCommit(false);
(100)                   stmt.executeUpdate(
(101)                           "INSERT INTO EMPLOYEE VALUES
('Hans','X','Hinterhuber','111111111',NULL,NULL,NULL,NULL,NULL,NULL);");
(102)                   Savepoint sp = con.setSavepoint();
(103)                   stmt.executeUpdate(
(104)                           "INSERT INTO EMPLOYEE VALUES
('Franz','X','Obermüller','222222222',NULL,NULL,NULL,NULL,NULL,NULL);");
(105)                   printContent(stmt);
(106)                   //suppose error happens here
(107)                   Thread.sleep(5000);
(108)                   boolean error = true;
(109)                   if (error) {
(110)                           con.rollback(sp);
(111)                   }
(112)                   stmt.executeUpdate(
(113)                           "INSERT INTO EMPLOYEE VALUES
('Fritz','X','Meier','333333333',NULL,NULL,NULL,NULL,NULL,NULL);");
(114)                   printContent(stmt);
(115)                   con.commit();
(116)           } catch (SQLException sqle) {
(117)                   Throwable t = sqle;
(118)                   while (t != null) {
(119)                           System.err.println("Type: " + t.getClass().getName());
(120)                   System.err.println("Message: " + t.getMessage());
(121)                   System.err.println("-----");
(122)                   t = t.getCause();
(123)                   }
```

```
(124)           } catch (InterruptedException ie) {
(125)                   ie.printStackTrace();
(126)           }
(127)   }
(128)}
```

Download des Beispiels

Netzwerkverkehr

Der Netzwerkmitschnitt zeigt den TCP-Kommunikationsverlauf der SQL-Anfrage `SELECT * FROM EMPLOYEE;`.
Die vom Anfrager an den MySQL-Server übermittelten Datenanteile sind rot hervorgehoben. Zusätzlich sind die für eine gleichwertige native Kommunikation anfallenden Daten berücksichtigt. Ihre Anteile entsprechen den zusätzlich durch Fettdruck hervorgehobenen.

```
0000 4  00 00 00 \n 4  .  0  .  1  2  -  s  t  a  n  d  a  r  d  -  l  o  g  00 '  00 00 00
N  4  V
0020 5  G  D  a  9  00 ,     \b 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 14 00 00 01 87
00 ÿ  ÿ
0040 ÿ  m  a  r  i  o  00 N  W  N  J  S  ]  L  K  00 03 00 00 02 00 00 00 \t 00 00 00 02 j
d  b  c
0060 t  e  s  t  03 00 00 01 00 00 00 0f 00 00 00 03 S  H  O  W     V  A  R  I  A  B  L  E
S  01 00
0080 00 01 02 19 00 00 02 00 \r V  a  r  i  a  b  l  e  _  n  a  m  e  03 1e 00 00 01 þ  03
01 00 1f
00a0 11 00 00 03 00 05 V  a  l  u  e  03 00 01 00 01 þ  03 01 00 1f 01 00 00 04 þ  \f 00 00
05 \b b
00c0 a  c  k  _  l  o  g  02 5  0  7  00 00 06 07 b  a  s  e  d  i  r  .  /  o  p  t  /  r
a  i  d
00e0 /  m  y  s  q  l  -  s  t  a  n  d  a  r  d  -  4  .  0  .  1  2  -  p  c  -  l  i  n
u  x  -
0100 i  6  8  6  /  18 00 00 07 11 b  i  n  l  o  g  _  c  a  c  h  e  _  s  i  z  e  05 3
2  7  6
0120 8     00 00 \b 17 b  u  l  k  _  i  n  s  e  r  t  _  b  u  f  f  e  r  _  s  i  z  e
07 8  3
0140 8  8  6  0  8  15 00 00 \t \r c  h  a  r  a  c  t  e  r  _  s  e  t  06 l  a  t  i  n
1  á  00
0160 00 \n 0e c  h  a  r  a  c  t  e  r  _  s  e  t  s  Ñ  l  a  t  i  n  1     b  i  g
5     c  z
0180 e  c  h     e  u  c  _  k  r     g  b  2  3  1  2     g  b  k     l  a  t  i  n  1  _
d  e
01a0 s  j  i  s     t  i  s  6  2  0     u  j  i  s     d  e  c  8     d  o  s     g  e  r
m  a  n
01c0 1     h  p  8     k  o  i  8  _  r  u     l  a  t  i  n  2     s  w  e  7     u  s  a
7     c
01e0 p  1  2  5  1     d  a  n  i  s  h     h  e  b  r  e  w     w  i  n  1  2  5  1     e
s  t  o
0200 n  i  a     h  u  n  g  a  r  i  a  n     k  o  i  8  _  u  k  r     w  i  n  1  2  5
1  u  k
0220 r     g  r  e  e  k     w  i  n  1  2  5  0     c  r  o  a  t     c  p  1  2  5  7
l  a  t
0240 i  n  5  15 00 00 0b 11 c  o  n  c  u  r  r  e  n  t  _  i  n  s  e  r  t  02 O  N  12
00 00 \f
0260 0f c  o  n  n  e  c  t  _  t  i  m  e  o  u  t  01 5  17 00 00 \r 15 c  o  n  v  e  r
t  _  c
0280 h  a  r  a  c  t  e  r  _  s  e  t  00 <  00 00 0e 07 d  a  t  a  d  i  r  3  /  o  p
t  /  r
02a0 a  i  d  /  m  y  s  q  l  -  s  t  a  n  d  a  r  d  -  4  .  0  .  1  2  -  p  c  -
l  i  n
02c0 u  x  -  i  6  8  6  /  d  a  t  a  /  13 00 00 0f 0f d  e  l  a  y  _  k  e  y  _  w
r  i  t
02e0 e  02 O  N  19 00 00 10 14 d  e  l  a  y  e  d  _  i  n  s  e  r  t  _  l  i  m  i  t
03 1  0
0300 0  1b 00 00 11 16 d  e  l  a  y  e  d  _  i  n  s  e  r  t  _  t  i  m  e  o  u  t  03
3  0  0
0320 18 00 00 12 12 d  e  l  a  y  e  d  _  q  u  e  u  e  _  s  i  z  e  04 1  0  0  0  \n
00 00 13
0340 05 f  l  u  s  h  03 O  F  F  \r 00 00 14 \n f  l  u  s  h  _  t  i  m  e  01 0  !  00
00 00 15 11
0360 f  t  _  b  o  o  l  e  a  n  _  s  y  n  t  a  x  0e +     -  >  <  (  )  ~  *  :  "
"  &  |
0380 12 00 00 16 0f f  t  _  m  i  n  _  w  o  r  d  _  l  e  n  01 4  14 00 00 17 0f f  t
```

```
_ m a
03a0 x _ w o r d _ l e n 03 2 5 4 1c 00 00 18 18 f t _ m a x _ w o r
d _ l
03c0 e n _ f o r _ s o r t 02 2 0 1c 00 00 19 10 f t _ s t o p w o r
d _ f
03e0 i l e \n ( b u i l t - i n ) \f 00 00 1a \b h a v e _ b d b 02 N
O 0f 00
0400 00 1b \n h a v e _ c r y p t 03 Y E S 10 00 00 1c 0b h a v e _ i n
n o d
0420 b 03 Y E S 0e 00 00 1d \t h a v e _ i s a m 03 Y E S \r 00 00 1e \t h
a v e
0440 _ r a i d 02 N O 16 00 00 1f \f h a v e _ s y m l i n k \b D I S
A B L
0460 E D 10 00 00 \f h a v e _ o p e n s s l 02 N O 15 00 00 ! 10 h a
v e _
0480 q u e r y _ c a c h e 03 Y E S 0b 00 00 " \t i n i t _ f i l e
00 ( 00
04a0 00 # 1f i n n o d b _ a d d i t i o n a l _ m e m _ p o o l
_ s i
04c0 z e 07 2 0 9 7 1 5 2 ! 00 00 $ 17 i n n o d b _ b u f f e r _
p o o
04e0 l _ s i z e \b 1 6 7 7 7 2 1 6 - 00 00 % 15 i n n o d b _ d a
t a _
0500 f i l e _ p a t h 16 i b d a t a 1 : 1 0 M : a u t o e x t
e n d
0520 16 00 00 & 14 i n n o d b _ d a t a _ h o m e _ d i r 00 19 00 00
' 16 i
0540 n n o d b _ f i l e _ i o _ t h r e a d s 01 4 18 00 00 ( 15 i
n n o
0560 d b _ f o r c e _ r e c o v e r y 01 0 1c 00 00 ) 19 i n n o d
b _ t
0580 h r e a d _ c o n c u r r e n c y 01 8 ! 00 00 * 1e i n n o d
b _ f
05a0 l u s h _ l o g _ a t _ t r x _ c o m m i t 01 1 18 00 00 + 14
i n n
05c0 o d b _ f a s t _ s h u t d o w n 02 O N 15 00 00 , 13 i n n o
d b _
05e0 f l u s h _ m e t h o d 00 1c 00 00 - 18 i n n o d b _ l o c k
_ w a
0600 i t _ t i m e o u t 02 5 0 17 00 00 . 13 i n n o d b _ l o g _
a r c
0620 h _ d i r 02 . / 17 00 00 / 12 i n n o d b _ l o g _ a r c h i
v e 03
0640 O F F 1f 00 00 0 16 i n n o d b _ l o g _ b u f f e r _ s i z
e 07 8
0660 3 8 8 6 0 8 1d 00 00 1 14 i n n o d b _ l o g _ f i l e _ s i
z e 07
0680 5 2 4 2 8 8 0 1c 00 00 2 19 i n n o d b _ l o g _ f i l e s _
i n _
06a0 g r o u p 01 2 1d 00 00 3 19 i n n o d b _ l o g _ g r o u p _
h o m
06c0 e _ d i r 02 . / 1d 00 00 4 1a i n n o d b _ m i r r o r e d _
l o g
06e0 _ g r o u p s 01 1 1a 00 00 5 13 i n t e r a c t i v e _ t i m
e o u
0700 t 05 2 8 8 0 0 18 00 00 6 10 j o i n _ b u f f e r _ s i z e 06
1 3 1
0720 0 7 2 19 00 00 7 0f k e y _ b u f f e r _ s i z e \b 1 6 7 7 7
2 1 6
0740 L 00 00 8 \b l a n g u a g e B / o p t / r a i d / m y s q l
- s t
0760 a n d a r d - 4 . 0 . 1 2 - p c - l i n u x - i 6 8 6 / s
h a r
0780 e / m y s q l / e n g l i s h / 17 00 00 9 13 l a r g e _ f i
l e s
07a0 _ s u p p o r t 02 O N 10 00 00 : \f l o c a l _ i n f i l e 02
O N 15
07c0 00 00 ; 10 l o c k e d _ i n _ m e m o r y 03 O F F \b 00 00 < 03
l o g
07e0 03 O F F 0f 00 00 = \n l o g _ u p d a t e 03 O F F 0b 00 00 > 07 l
o g _
0800 b i n 02 O N 16 00 00 ? 11 l o g _ s l a v e _ u p d a t e s 03
O F F
0820 15 00 00 @ 10 l o g _ s l o w _ q u e r i e s 03 O F F 11 00 00 A
\f l o
0840 g _ w a r n i n g s 03 O F F 13 00 00 B 0f l o n g _ q u e r y
```

```
_ t i
0860 m e 02 1 0 19 00 00 C 14 l o w _ p r i o r i t y _ u p d a t e
s 03 O
0880 F F 1b 00 00 D 16 l o w e r _ c a s e _ t a b l e _ n a m e s
03 O F
08a0 F 1b 00 00 E 12 m a x _ a l l o w e d _ p a c k e t 07 1 0 4 7
5 5 2
08c0 ! 00 00 F 15 m a x _ b i n l o g _ c a c h e _ s i z e \n 4 2
9 4 9
08e0 6 7 2 9 5 1b 00 00 G 0f m a x _ b i n l o g _ s i z e \n 1 0 7
3 7 4
0900 1 8 2 4 14 00 00 H 0f m a x _ c o n n e c t i o n s 03 1 0 0 16
00 00 I
0920 12 m a x _ c o n n e c t _ e r r o r s 02 1 0 17 00 00 J 13 m a
x _ d
0940 e l a y e d _ t h r e a d s 02 2 0 1d 00 00 K 13 m a x _ h e a
p _ t
0960 a b l e _ s i z e \b 1 6 7 7 7 2 1 6 19 00 00 L \r m a x _ j o
i n _
0980 s i z e \n 4 2 9 4 9 6 7 2 9 5 15 00 00 M 0f m a x _ s o r t _
l e n
09a0 g t h 04 1 0 2 4 17 00 00 N 14 m a x _ u s e r _ c o n n e c t
i o n
09c0 s 01 0 12 00 00 O 0e m a x _ t m p _ t a b l e s 02 3 2 00 00 P
14 m a
09e0 x _ w r i t e _ l o c k _ c o u n t \n 4 2 9 4 9 6 7 2 9 5
* 00 00
0a00 Q 1f m y i s a m _ m a x _ e x t r a _ s o r t _ f i l e _
s i z
0a20 e \t 2 6 8 4 3 5 4 5 6 % 00 00 R 19 m y i s a m _ m a x _ s o
r t _
0a40 f i l e _ s i z e \n 2 1 4 7 4 8 3 6 4 7 1b 00 00 S 16 m y i s
a m _
0a60 r e c o v e r _ o p t i o n s 03 O F F 00 00 T 17 m y i s a
m _ s
0a80 o r t _ b u f f e r _ s i z e 07 8 3 8 8 6 0 8 17 00 00 U 11 n
e t _
0aa0 b u f f e r _ l e n g t h 04 8 1 9 2 14 00 00 V 10 n e t _ r e
a d _
0ac0 t i m e o u t 02 3 0 13 00 00 W 0f n e t _ r e t r y _ c o u n
t 02 1
0ae0 0 15 00 00 X 11 n e t _ w r i t e _ t i m e o u t 02 6 0 \b 00 00
Y 03 n
0b00 e w 03 O F F 13 00 00 Z 10 o p e n _ f i l e s _ l i m i t 01 0
F 00 00
0b20 [ \b p i d _ f i l e < / o p t / r a i d / m y s q l - s t
a n d
0b40 a r d - 4 . 0 . 1 2 - p c - l i n u x - i 6 8 6 / d a t
a / l i
0b60 n u x . p i d 0b 00 00 \ \t l o g _ e r r o r 00 \n 00 00 ] 04 p o
r t 04
0b80 3 3 0 6 14 00 00 ^ 10 p r o t o c o l _ v e r s i o n 02 1 0 18
00 00 _
0ba0 10 r e a d _ b u f f e r _ s i z e 06 1 3 1 0 7 2 1c 00 00 ` 14
r e a
0bc0 d _ r n d _ b u f f e r _ s i z e 06 2 6 2 1 4 4 14 00 00 a 11
r p l
0be0 _ r e c o v e r y _ r a n k 01 0 1a 00 00 b 11 q u e r y _ c a
c h e
0c00 _ l i m i t 07 1 0 4 8 5 7 6 13 00 00 c 10 q u e r y _ c a c h
e _ s
0c20 i z e 01 0 14 00 00 d 10 q u e r y _ c a c h e _ t y p e 02 O N
\f 00 00
0c40 e \t s e r v e r _ i d 01 1 17 00 00 f 11 s l a v e _ n e t _ t
i m e
0c60 o u t 04 3 6 0 0 19 00 00 g 15 s k i p _ e x t e r n a l _ l o
c k i
0c80 n g 02 O N 14 00 00 h 0f s k i p _ n e t w o r k i n g 03 O F F
17 00 00
0ca0 i 12 s k i p _ s h o w _ d a t a b a s e 03 O F F 13 00 00 j 10
s l o
0cc0 w _ l a u n c h _ t i m e 01 2 17 00 00 k 06 s o c k e t 0f / t
m p /
0ce0 m y s q l . s o c k 18 00 00 l 10 s o r t _ b u f f e r _ s i
z e 06
0d00 5 2 4 2 8 0 0b 00 00 m \b s q l _ m o d e 01 0 0f 00 00 n 0b t a b
```

```
l e _
0d20 c a c h e 02 6 4 12 00 00 o \n t a b l e _ t y p e 06 I N N O D
B 14 00
0d40 00 p 11 t h r e a d _ c a c h e _ s i z e 01 0 14 00 00 q \f t h
r e a
0d60 d _ s t a c k 06 1 9 6 6 0 8 1d 00 00 r \f t x _ i s o l a t i
o n 0f
0d80 R E P E A T A B L E - R E A D 0e 00 00 s \b t i m e z o n e 04
C E S
0da0 T 18 00 00 t 0e t m p _ t a b l e _ s i z e \b 3 3 5 5 4 4 3 2
\r 00 00
0dc0 u 06 t m p d i r 05 / t m p / 1c 00 00 v 07 v e r s i o n 13 4 .
0 . 1
0de0 2 - s t a n d a r d - l o g 13 00 00 w \f w a i t _ t i m e o
u t 05
0e00 2 8 8 0 0 01 00 00 x þ 11 00 00 00 03 S E T   a u t o c o m m i t
= 1 03
0e20 00 00 01 00 00 00 18 00 00 00 03 S E L E C T   *   F R O M   E M P L
O Y E
0e40 E ; 01 00 00 01 \n 19 00 00 02 \b E M P L O Y E E 05 F N A M E 03 \n 00
00 01 ý
0e60 03 01 00 00 19 00 00 03 \b E M P L O Y E E 05 M I N I T 03 01 00 00 01 þ
03 00 00
0e80 00 19 00 00 04 \b E M P L O Y E E 05 L N A M E 03 \n 00 00 01 ý 03 01 00
00 17 00
0ea0 00 05 \b E M P L O Y E E 03 S S N 03 \t 00 00 01 03 03 01 00 00 19 00 00 06
\b E M
0ec0 P L O Y E E 05 B D A T E 03 \n 00 00 01 \n 03 00 00 00 1b 00 00 07 \b E M
P L O
0ee0 Y E E 07 A D D R E S S 03 1e 00 00 01 ý 03 00 00 00 17 00 00 \b \b E M P
L O Y
0f00 E E 03 S E X 03 01 00 00 01 þ 03 00 01 00 1a 00 00 \t \b E M P L O Y E E
06 S A
0f20 L A R Y 03 07 00 00 01 05 03   00 02 1c 00 00 \n \b E M P L O Y E E \b S
U P E
0f40 R S S N 03 \t 00 00 01 03 03 00 00 00 17 00 00 0b \b E M P L O Y E E 03 D
N O 03
0f60 01 00 00 01 03 03 00 00 00 01 00 00 \f þ R 00 00 \r 04 J o h n 01 B 05 S m i
t h \t
0f80 1 2 3 4 5 6 7 8 9 \n 1 9 6 5 - 0 1 - 0 9 18 7 3 1   F o n d
r e n
0fa0 ,   H o u s t o n ,   T X 01 M \b 3 0 0 0 0 . 0 0 \t 3 3 3 4
4 5 5
0fc0 5 5 01 5 R 00 00 0e \b F r a n k l i n 01 T 04 W o n g \t 3 3 3 4
4 5 5
0fe0 5 5 \n 1 9 5 5 - 1 2 - 0 8 15 6 3 8   V o s s ,   H o u s t
o n ,
1000   T X 01 M \b 4 0 0 0 0 . 0 0 \t 8 8 8 6 6 5 5 5 5 01 5 T 00 00
0f 06 A
1020 l i c i a 01 J 06 Z e l a y a \t 9 9 9 8 8 7 7 7 7 \n 1 9 6 8
- 0 7
1040 - 1 9 17 3 3 2 1   C a s t l e ,   S p r i n g ,   T X 01 F
\b 2 5
1060 0 0 0 . 0 0 \t 9 8 7 6 5 4 3 2 1 01 4 W 00 00 10 \b J e n n i f
e r 01
1080 S 07 W a l l a c e \t 9 8 7 6 5 4 3 2 1 \n 1 9 4 1 - 0 6 - 2
0 17 2
10a0 9 1   B e r r y ,   B e l l a i r e ,   T X 01 F \b 4 3 0 0
0 . 0
10c0 0 \t 8 8 8 6 6 5 5 5 5 01 4 V 00 00 11 06 R a m e s h 01 K 07 N a
r a y
10e0 a n \t 6 6 6 8 8 4 4 4 4 \n 1 9 6 2 - 0 9 - 1 5 18 9 7 5   F
i r e
1100   O a k ,   H u m b l e ,   T X 01 M \b 3 8 0 0 0 . 0 0 \t 3
3 3 4
1120 4 5 5 5 5 01 5 S 00 00 12 05 J o y c e 01 A 07 E n g l i s h \t 4
5 3 4
1140 5 3 4 5 3 \n 1 9 7 2 - 0 7 - 3 1 16 5 6 3 1   R i c e ,   H
o u s
1160 t o n ,   T X 01 F \b 2 5 0 0 0 . 0 0 \t 3 3 3 4 4 5 5 5 5 01
5 S 00
1180 00 13 05 A h m a d 01 V 06 J a b b a r \t 9 8 7 9 8 7 9 8 7 \n 1
9 6 9
11a0 - 0 3 - 2 9 17 9 8 0   D a l l a s ,   H o u s t o n ,   T
X 01 M
11c0 \b 2 5 0 0 0 . 0 0 \t 9 8 7 6 5 4 3 2 1 01 4 G 00 00 14 05 J a m
```

```
e  s  01
11e0 E  04 B  o  r  g  \t 8  8  8  6  6  5  5  5  5  \n 1  9  3  7  -  1  1  -  1  0  16 4
5  0
1200 S  t  o  n  e  ,     H  o  u  s  t  o  n  ,     T  X  01 M  \b 5  5  0  0  0  .  0  0
û  01 1
1220 01 00 00 15 þ
```

Die Analyse des Datenverkehrs zeigt, daß im Falle der JDBC-basierten Kommunikation ein gegenüber der nativen Schnittstelle um 3529 Byte vergrößertes Datenaufkommen ausgetauscht wird. Diese zusätzliche Datenmenge fällt jedoch nur einmal zum Zeitpunkt des JDBC-Verbindungsaufbaus statisch an. (Vgl. Mitschnitt der mehrfachen Ausführung einer SQL-Anfrage innerhalb einer bestehenden JDBC-Verbindung)
Zusätzlich offenbart Zeile 0x40 des Datenverkehrs die verschlüsselte Übermittlung des Paßwortes des Anwenders *mario*. Allerdings werden die per Anfrage ermittelten Nutzdaten (ab Zeile 0xe40) unverschlüsselt über die Netzwerkschnittstelle übertragen und stellen somit ein potentielles Angriffsziel dar.
Abhilfe hierfür kann die Tunnelung des Datenverkehrs, beispielsweise mittels SSH, durch eine sichere Verbindung bieten.

Zugriff mit JDBC auf nicht-relationale Quellen

Zwar legt die JDBC-Schnittstelle inhärent eine relationale und damit tabellenorientierte Sicht der zu verarbeitenden Datenquelle zugrunde, jedoch ist die Nutzung der JDBC keinesweges auf jeden Anwendungsfall beschränkt.
Beispielsweise steht mit dem Produkt *Sunopsis XML Driver (JDBC Edition)* ein JDBC-Treiber für den dateibasierten Zugriff auf XML-Inhalte zur Verfügung.
Der Code des Beispiels 66 zeigt die Verwendung.

Als Beispiel dient folgende XML-Datei:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Employees>
        <Employee>
                <FName>John</FName>
                <Minit>B</Minit>
                <LName>Smith</LName>
        </Employee>
        <Employee>
                <FName>Franklin</FName>
                <Minit>T</Minit>
                <LName>Wong</LName>
        </Employee>
        <Employee>
                <FName>Alicia</FName>
                <Minit>J</Minit>
                <LName>Zelaya</LName>
        </Employee>
        <Employee>
                <FName>Jennnifer</FName>
                <Minit>S</Minit>
                <LName>Wallace</LName>
        </Employee>
        <Employee>
                <FName>Ramesh</FName>
                <Minit>K</Minit>
                <LName>Narayan</LName>
        </Employee>
        <Employee>
                <FName>Joyce</FName>
                <Minit>A</Minit>
                <LName>English</LName>
        </Employee>
</Employees>
```

Beispiel 66: JDBC-artige Verarbeitung von XML-Dateien

```
(1)import java.sql.Connection;
(2)import java.sql.DriverManager;
(3)import java.sql.ResultSet;
(4)import java.sql.SQLException;
(5)import java.sql.Statement;
(6)
(7)public class JDBCXML {
(8)     public static void main(String[] args) {
(9)             try {
(10)                    Class.forName("com.sunopsis.jdbc.driver.xml.SnpsXmlDriver");
(11)            } catch (ClassNotFoundException cnfe) {
(12)                    System.err.println("Driver class not found");
(13)                    cnfe.printStackTrace();
(14)            }
(15)            Connection con = null;
(16)            try {
(17)                    con = (Connection) DriverManager.getConnection("jdbc:snps:xml");
(18)            } catch (SQLException e1) {
(19)                    System.err.println("Error establishing database connection");
(20)                    Throwable t = sqle;
(21)                    while (t != null) {
(22)                    System.err.println("Type: " + t.getClass().getName());
(23)                    System.err.println("Message: " + t.getMessage());
(24)                    System.err.println("-----");
(25)                    t = t.getCause();
(26)                    }
(27)            }
(28)            Statement stmt = null;
(29)            try {
(30)                    stmt = con.createStatement();
(31)            } catch (SQLException sqle) {
(32)                    Throwable t = sqle;
(33)                    while (t != null) {
(34)                    System.err.println("Type: " + t.getClass().getName());
(35)                    System.err.println("Message: " + t.getMessage());
(36)                    System.err.println("-----");
(37)                    t = t.getCause();
(38)                    }
(39)            }
(40)            try {
(41)                    stmt.execute("load file \"employee.xml\" on schema EMP readonly");
(42)                    stmt.execute("set schema EMP");
(43)                    ResultSet rs = con.getMetaData().getTables("", "EMP", "%", null);
(44)
(45)                    while (rs.next()) {
(46)                            System.out.println("TABLE=" + rs.getString("TABLE_NAME"));
(47)                            System.out.print("(");
(48)                            ResultSet colRS =
(49)                                    con.getMetaData().getColumns(
(50)                                            "",
(51)                                            "EMP",
(52)                                            rs.getString("TABLE_NAME"),
(53)                                            "%");
(54)                            while (colRS.next()) {
(55)                                    System.out.println(colRS.getString("COLUMN_NAME")
+ ",");
(56)                            }
(57)                            System.out.println(")\n");
(58)                    }
(59)            } catch (SQLException sqle) {
(60)                    Throwable t = sqle;
(61)                    while (t != null) {
(62)                    System.err.println("Type: " + t.getClass().getName());
(63)                    System.err.println("Message: " + t.getMessage());
(64)                    System.err.println("-----");
(65)                    t = t.getCause();
(66)                    }
(67)            }
(68)
(69)            ResultSet rs = null;
(70)            try {
(71)                    rs = (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
(72)                    while (!rs.isLast()) {
(73)                            rs.next();
(74)                            System.out.println(rs.getString("FNAME"));
(75)                    }
```

```
(76)            } catch (SQLException sqle) {
(77)                    Throwable t = sqle;
(78)                    while (t != null) {
(79)                    System.err.println("Type: " + t.getClass().getName());
(80)                    System.err.println("Message: " + t.getMessage());
(81)                    System.err.println("-----");
(82)                    t = t.getCause();
(83)                    }
(84)            }
(85)
(86)            try {
(87)                    stmt.executeUpdate("INSERT INTO EMPLOYEE (FNAME, MINIT, LNAME)
VALUES('James', 'E', 'Borg');");
(88)            } catch (SQLException sqle) {
(89)                    Throwable t = sqle;
(90)                    while (t != null) {
(91)                    System.err.println("Type: " + t.getClass().getName());
(92)                    System.err.println("Message: " + t.getMessage());
(93)                    System.err.println("-----");
(94)                    t = t.getCause();
(95)                    }
(96)            }
(97)
(98)    }
(99)}
```

Download des Beispiels
Download der Ergebnisdatei

Web-Referenzen 9: Weiterführende Links

- JDBC @ SUN
- JDBC learning center @ SUN
- JDBC Tutorial
- JDBC FAQ @ JGuru.com
- G. Reese: *Database Programming with JDBC and Java*. O'Reilly, 1997
- Verhältnis von X/Open CLI und ODBC

3.2 Enterprise Java Beans (EJB)

Neben den bereits aus anderen Veranstaltungen bekannten Servlets und den davon abgeleiteten Java Server Pages bildet die Technik der *Enterprise Java Beans* (EJB) einen weiteren zentralen Baustein der Java 2 Enterprise Plattform. Als serverseitige Komponenten kommt den EJBs heute große Bedeutung in der Realisierung komplexer Anwendungen, insbesondere durch Umsetzung der sog. „Business Logik“, d.h. den nicht-interaktiven fachlichen Anwendungsteilen, zu.

Der Begriff der Enterprise Java Bean stützt sich auf dem historisch älteren der *Java Bean*. Eine solche stellt eine abgeschlossene wiederverwendbare Softwarekomponente dar, die nach ihrer Erstellung über festgelegte Schnittstellen parametrisiert und manipuliert werden kann. Hierzu muß eine Bean eine festgelegte Interaktionsschnittstelle bieten, die durch die Java Bean Spezifikation definiert ist. Es handelt sich dabei um eine Reihe von Konventionen, der eine Bean gehorchen muß, jedoch um keine festgelegte API, die durch eine Komponente zu implementieren ist.
Der Begriff der Enterprise Java Bean greift diese inhaltliche Fundierung auf und präzisiert gleichzeitig die technische Umsetzung. So stellt eine Enterprise Java Bean eine Softwarekomponente dar, die in einer festgelegten Ausführungsumgebung, welche durch die EJB-Spezifikation festgelegte Dienste zur Nutzung durch die Beans anbieten kann. Eine solche Ausführungsumgebung wird als *Container* bezeichnet.
Ziel der Trennung in Komponente und Ausführungsumgebung ist die Zielsetzung, die Enterprise Java Bean ausschließlich zur Umsetzung fachlicher Aufgaben heranzuziehen und alle infrastrukturellen Fragestellungen wie Betriebsmittelverwaltung, Persistenz oder Sicherheit durch die Ausführungsumgebung in gleicher Weise für alle Komponenten bereitzustellen.
Ein EJB-Container wird zumeist im Rahmen eines Application-Servers bereitgestellt.
Die gelegentlich anzutreffende Hervorhebung der anfänglich für Java Beans intendierten *visuellen Manipulationsmöglichkeit* trifft für Enterprise Java Beans nicht zu und hat sich für Java Beans auch nur begrenzte Bedeutung erlangt.

Spezifikationsgemäß können EJB-Container folgende Eigenschaften offerieren:

- **Betriebsmittelverwaltung**
  Typischerweise verwaltet ein einziger EJB-Container gleichzeitig eine Reihe verschiedener Enterprise Java Beans. Zur Organisation und Aufrechterhaltung der Ausführbarkeit obliegt dem Container die Zuteilung von Betriebsmitteln wie Hauptspeicher, CPU-Zeit oder Netzwerkressourcen an die verwalteten EJB.

Hierunter fällt insbesondere auch die Einlagerung, Instanziierung und Entfernung der EJBs selbst.

- **Zustandsverwaltung**
  In praktischen Anwendungen ist oft die Nutzung zustandsbehafteter Kommunikation, die sich über verschiedene Einzelinteraktionen erstreckt gewünscht. Die hierfür notwendigen technischen Voraussetzungen (Zustandsspeicherung, Korrelation der Einzelinteraktionen) werden durch den Container bereitgestellt.
- **Transaktionsverwaltung**
  Erweiterung der Zustandsverwaltung. Zur Gewährleistung des benötigten Verhaltens müssen EJBs keine eigenen Implementierungen zur Verfügung stellen, sondern können vorhandene Dienste des Containers nutzen.
- **Sicherheit**
  Die Sicherheit von EJBs kann durch Vergabe von Zugriffsrechten und Rollen auf Containerebene gesteuert werden.
- **Persistenz**
  Der interne Zustand einer verwalteten EJB kann wahlfrei in persistiert und zu einem späteren Zeitpunkt wiederhergestellt werden.
- **Entfernter Zugriff**
  Der Zugriff auf EJBs erfolgt mittels Remote Method Invocation und ist daher Lokationstransparent.

Neben den in der Aufzählung dargestellten Eigenschaften dürfen Container zusätzlich Weitere wahlfrei implementieren.

EJB-Typen

Grundsätzlich lassen sich alle EJBs drei Typen zuordnen: *Session Beans*, *Entity Beans* und *Message Driven Beans*. Während erstere hauptsächlich zur Abbildung von Abläufen eingesetzt werden, dienen Entity Beans der Abwicklung von Zugriffen auf Daten. Eine Sonderstellung nehmen die *Message Driven Beans* ein, die lediglich hinsichtlich ihres Kommunikationsverhaltens festgelegt sind.

Session Beans dienen der Abbildung von Abläufen im Rahmen der Programmierung der sog. Business Logik. Die Lebensdauer (d.h. Zeitspanne zwischen Erzeugung im und Entfernung aus dem Hauptspeicher) ist daher identisch mit der einer durch den Client erfolgenden Anfrage. Jede zu einem Zeitpunkt existierende Session Bean repräsentiert daher eine zugehörige Clientinstanz.
Nach ihrer internen Ausgestaltung werden *stateless* und *statefull* Session Beans unterschieden. Während Erstere keinen über einen einzigen Aufruf hinausgehenden Zustand verwalten und daher seiteneffektfrei lediglich auf den durch den Aufruf übermittelten Daten operieren erhält das zustandserhaltende Pendant die Daten eines Aufrufs und kann diese auch in nachfolgenden Aufrufen verarbeiten.

Entity Beans sind programmiersprachliche Stellvertreter datenbankresidenter Objekte. Sie dienen dem erleichterten Zugriff auf persistent vorliegende Datenbestände. Ihre interne Realisierung ist eng mit der Technik relationaler Datenbankmanagement System verbunden. So werden Sie durch einen anwenderdefinierten Primärschlüssel dauerhaft identifiziert.

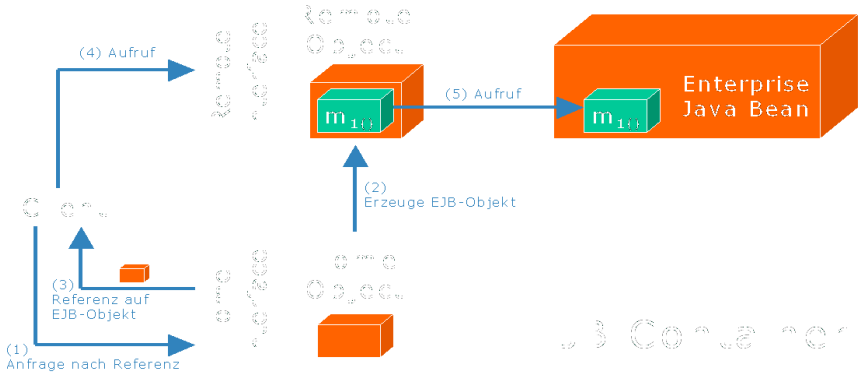
Message Driven Beans sind hinsichtlich ihres Kommunikationsverhaltens auf asynchrone Aufrufe beschränkt. Die Realisierung des eigentlichen Verhaltens wird durch eine Ausprägung eines der anderen Beantypen geboten.

Session Beans

Konzeptionell umfaßt jede EJB-Anwendung, die Session Beans einsetzt, die in Abbildung 6 dargestellten Teile:

- Implementierung der EJB selbst.
- *Remote*-Schnittstelle zum Zugriff auf die durch die Bean publizierten Methoden.
- *Home*-Schnittstelle zur Ermittlung einer Referenz auf das Bean-Objekt.

Abbildung 6: *Aufrufstruktur einer zustandslosen EJB*



(click on image to enlarge!)

Gegenüber der Realisierung als RMI-Anwendung benötigt die Umsetzung als zustandslose Session Bean die Erstellung einer sog. „Home-Schnittstelle“ (*Home Interface*), welche die Operation `create` zur Instanziierung des serverseitigen EJB-Objekts bietet.
Sie ist im Beispiel 67 dargestellt.

Beispiel 67: Home-Schnittstelle einer EJB

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.CreateException;
(3)import javax.ejb.EJBHome;
(4)
(5)public interface SayHelloHome extends EJBHome {
(6)     public SayHello create() throws RemoteException, CreateException;
(7)}
```

Download des Beispiels

Die anwenderdefinierte Home-Schnittstelle erweitert die durch die Standard-API vorgegebene Schnittstelle `EJBHome`. Diese definiert Operationen zur Entfernung existierender EJB-Objekte aus dem Hauptspeicher (`remove`) sowie zur Ermittlung von Metadaten (`getEJBMetaData`) oder zum Erhalt eines netzwerkunabhängigen Verweises auf das EJB-Objekt (`getHomeHandle`).
Im Einzelnen sind dies die Operationen:

- `EJBMetaData getEJBMetaDate()`
  Liefert ein `EJBMetaData`-Objekt welches einzelne Eigenschaften einer EJB näher beschreibt. Hierzu zählen:
  - Klasse der Home-Schnittstelle
  - Klasse des Primärschlüssels (nur vorhanden sofern es sich um eine Entity Bean handelt)
  - Klasse der Remote-Schnittstelle
  - Boole'scher Wert, der angibt, ob es sich um eine Session Bean handelt
  - Boole'scher Wert, der angibt, ob es sich um eine zustandslose Session Bean handelt
- `HomeHandle getHomeHandle()`
  Liefert ein Objekt des Typs `HomeHandle` zurück, welches eine netzwerkunabhängige Abstraktion des Verweises auf das Home-Objekt realisiert.
- `void remove (Handle h)`
  Entfernt ein durch den Objektverweis (`Handle`) identifiziertes EJB-Objekt aus dem Hauptspeicher.
- `void remove (Object pk)`
  Entfernt ein durch das übergebene Primärschlüsselobjekt identifiziertes EJB-Objekt aus dem Hauptspeicher.

Interessanterweise definiert die Schnittstelle zwar Operationen zur Ermittlung von Daten über bestehende Objekte und zur Entfernung dieser Objekte aus dem Hauptspeicher, nicht jedoch zu ihrer Erzeugung.
Dies liegt in der durch die Programmiersprache Java angestrebten statischen Typsicherheit begründet, die es nicht gestattet Operationen mit variablen Parameterlisten --- wie sie für die zum API-Erstellungszeitpunkt unbekannten spezifischen Initialisierungsparameter aller denkbaren EJBs benötigt würden --- zu versehen.
Aus diesem Grunde definiert die EJB-Spezifikation informell, daß ein diese Schnittstelle erweiternde eigene Home-Schnittstelle zusätzlich die Methode `create`, deren Signatur als Rückgabetyp den Typ der Remote-Schnittstelle vorsehen muß definiert. Zusätzlich enthält diese Operation die zur Initialisierung der Bean benötigten Parameter in ihrer Parameterliste.

Die im Beispiel 68 ist Remote-Schnittstelle dargestellt, deren Ausprägungen von Home-Objekten angesprochenen werden:

Beispiel 68: Remote-Schnittstelle einer EJB

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.EJBObject;
(3)
(4)public interface SayHello extends EJBObject {
(5)     public String sayHello(String name) throws RemoteException;
(6)}
```

Download des Beispiels

Schnittstellen dieses Typs enthalten ausschließlich die fachlichen Operationen, d.h. die Signaturen der Methoden, die später durch den Client benutzt werden.
Jede Remote-Schnittstelle erweitert zusätzlich die vorgegebene Schnittstelle `EJBObject`, welche, ähnlich zur Home-Schnittstelle, einige Operationen zur Verwaltung eines EJB-Objektes vorgibt:

- `EJBHome getEJBHome()`
  Liefert die Home-Schnittstelle einer EJB.
- `Handle getHandle()`
  Liefert ein Objekt des Typs `HomeHandle` zurück, welches eine netzwerkunabhängige Abstraktion des Verweises auf das Home-Objekt realisiert.
- `Object getPrimaryKey()`

Liefert das Primärschlüsselobjekt einer Entity Bean.

- `boolean isIdentical(EJBObject eo)`
  Prüft ob das übergebene EJB-Objekt dasselbe wie das Objekt ist auf dem die Methode ausgeführt wird.
- `void remove()`
  Entfernt das EJB-Objekt aus dem Bean-Container.

Beispiel 69 zeigt die Implementierung der Bean selbst:

Beispiel 69: Realisierung einer Session Bean

```
(1)import java.rmi.RemoteException;
(2)import java.util.Date;
(3)import javax.ejb.EJBException;
(4)import javax.ejb.SessionBean;
(5)import javax.ejb.SessionContext;
(6)
(7)public class HelloWorldBean implements SessionBean {
(8)     public HelloWorldBean() {
(9)     }
(10)
(11)    public String sayHello(String name) {
(12)            return ("Hello " + name + " it is now " + new Date().toString());
(13)    }
(14)
(15)    public void setSessionContext(SessionContext arg0)
(16)            throws EJBException, RemoteException {
(17)    }
(18)    public void ejbRemove() throws EJBException, RemoteException {
(19)    }
(20)
(21)    public void ejbCreate() throws EJBException {
(22)    }
(23)
(24)    public void ejbActivate() throws EJBException, RemoteException {
(25)    }
(26)
(27)    public void ejbPassivate() throws EJBException, RemoteException {
(28)    }
(29)}
```

Download des Beispiels

Die programmiersprachliche Umsetzung der Bean enthält die Methoden der in der Remote-Schnittstelle bekanntgegebenen fachlichen Operationen. Zusätzlich muß ein Konstruktur expliziert werden, dessen Parameterliste mit den für die Operation `create` des Home-Interfaces gegebenen übereinstimmen. Spezifikationsgemäß muß jede Session Bean die gleichnamige API-Schnittstelle implementieren. Diese definiert einige Operationen zur Behandlung unterschiedlicher Lebenszyklusstadien einer EJB. Hierunter fallen Methoden, die beim Erzeugen (`ejbCreate`), Entfernen (`ejbRemove`), bei der Passivierung (d.h. Auslagerung auf Hintergrundspeicher) (`ejbPassivate`) und dessen Reaktivierung (`ejbActivate`) eines EJB-Objekts durch die Ausführungsumgebung aufgerufen werden.

Bei der zwingend zu implementierenden Schnittstelle `SessionBean` handelt es sich nicht nur um eine Konvention um die Umsetzung der Lebenszyklusschnittstelle sicherzustellen, sondern auch um die Kategorisierung der Bean selbst. So stellt die im Beispiel verwandte Schnittstelle `SessionBean` neben `EntityBean` und `MessageDrivenBean` eine Spezialisierung der (operationslosen) Schnittstelle `EnterpriseBean` dar, deren „Implementierung“ durch eine Klasse lediglich zur Kennzeichnung dieser als EJB herangezogen wird.
Die genannten Spezialisierungen dieser Schnittstelle erfüllen daher sowohl den Zweck der Ausübung des Implementierungszwanges für die in ihnen aufgeführten Operationen als auch den der typisierenden Kennzeichnung.
Darüberhinaus ist `EnterpriseBean` als Spezialisierung der Standard-Schnittstelle `Serializable` angelegt. In der Konsequenz muß jedes EJB-Objekt durch die Javasprachmechnismen serialisierbar sein. Diese Eigenschaft wird insbesondere für die Passivierung und im Rahmen der Entity Beans genutzt.

Konzeptionell erinnert die Trennung in publizierte fachliche Schnittstelle (Remote-Schnittstelle) und deren technischer Umsetzung durch die EJB an die aus der Betrachtung des Remote Method Invocation Mechanismus bekannte Struktur.
Allerdings weicht die Umsetzung der Bean von der dort anzutreffenden Konvention ab, die publizierte Schnittstelle selbst durch die realisierende Klasse zu implementieren. Dies liegt vor allem an der gegenüber RMI veränderten Struktur der publizierten Schnittstelle begründet. Während RMI für die Schnittstelle die Spezialisierung der operationslosen Standardschnittstelle `Remote` fordert die EJB-Spezifikation die Erweiterung der Schnittstelle `EJBObject`, welche selbst die oben dargestellten Operationen definiert. Aus diesem Grunde würde die Aufnahme der Remote-Schnittstelle, obwohl konzeptionell durchaus zu rechtfertigen, in die Umsetzungsliste der EJB gleichzeitig die Implementierung

von zumindest leeren Methodenrümpfen für die in EJBObject definierten Operationen notwendig werden lassen.
Abgesehen von dieser Ausnahme rekonstruiert das Verhältnis zwischen EJB und deren Remote-Schnittstelle die aus RMI bekannte Beziehung zwischen Schnittstelle und Umsetzung.

Die Nutzung einer durch eine Java Bean angebotenen Funktionalität erfolgt gemäß dem in Abbildung 6 dargestellten Schema. Ein dies umsetzender Client ist in Beispiel 70 dargestellt.

Beispiel 70: Zugriff auf eine Session Bean

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.CreateException;
(3)import javax.naming.Context;
(4)import javax.naming.InitialContext;
(5)import javax.naming.NamingException;
(6)import javax.rmi.PortableRemoteObject;
(7)
(8)public class CallHelloWorldBean {
(9)     public static void main(String[] args) {
(10)            try {
(11)                    Context initial = new InitialContext();
(12)                    Object objRef = initial.lookup("helloBean");
(13)
(14)                    SayHelloHome home =
(15)                            (SayHelloHome) PortableRemoteObject.narrow(
(16)                                    objRef,
(17)                                    SayHelloHome.class);
(18)                    SayHello sh = home.create();
(19)
(20)                    System.out.println(sh.sayHello("Mario"));
(21)
(22)            } catch (NamingException ne) {
(23)                    ne.printStackTrace();
(24)            } catch (RemoteException re) {
(25)                    re.printStackTrace();
(26)            } catch (CreateException ce) {
(27)                    ce.printStackTrace();
(28)            }
(29)    }
(30)}
```

Download des Beispiels

Zunächst ermittelt der Client unter Nutzung der JNDI-API eine Referenz auf die EJB. Dies geschieht durch Anfrage (lookup) an den JNDI-Dienst unter Übergabe des bekannten Klarnamens (*helloBean*).
Die erhaltene generische Referenz wird durch Aufruf der statischen Methode narrow der Klasse PortableRemoteObject typsicher in eine Ausprägung der Home-Schnittstelle konvertiert.
Der Aufruf der in dieser Schnittstelle durch den Anwender definierten create-Methode sorgt für die serverseitige Instanziierung der EJB, die als Ausprägung der Remote-Schnittstelle geliefert wird.
Tatsächlich wird nicht das EJB-Objekt selbst durch den Methodenaufruf retourniert, sondern lediglich ein netzwerktransparenter Verweis darauf, der jedoch clientseitig einer lokalen Objektreferenz gleichgestellt verwendet werden kann.
Ferner wird serverseitig zur Kommunikation mit der EJB ein Home-Objekt erzeugt, welchem eine Stellvertreterrolle für den anfragenden Client zukommt.
Der Aufruf der durch die EJB zur Verfügung gestellten Methode erfolgt identisch zu dem einer Lokalen.

Entity Beans

Die zweite zentrale Klasse von Enterprise Java Beans bilden die zur serverseitigen Persistierung von Objekten dienenden *Entity Beans*.
Sie kapseln Datenbankinhalte durch Objekte, die gemäß der EJB-Spezifikation instanziierbar und zugreifbar sind. Die Verwaltung der gekapselten Dateninhalte erfolgt durch ein frei festlegbares Datenbankmanagementsystem, die der Objekte selbst durch den EJB-Container.
Ziel dieser Technik ist es die Komplexität der Persistenzlogik für den Verwender der bereitgestellten Bean vollkommen transparent zu gestalten und serverseitig zu realisieren.
Die Familie der Entity Beans selbst zerfällt in zwei Untertypen, welche sich entlang des Realisierungspunktes der Persistenzlogik separieren: *Bean Managed Persistence*, der Bean obliegt die Umsetzung der Persistenzlogik, und *Container Managed Persistence*, hierbei wird die Persistenzlogik durch den EJB-Container realisiert.

Das nachfolgende Beispiel zeigt die Umsetzung einer Entity Bean mit Bean Managed Persistence. Es kapselt die Verwaltung und den Zugriff auf Objekte, die Personen beschreiben. Jedes Personen-Objekt enthält Daten zu Name, Geburtsdatum und Wohnstraße. Der Name dient als eindeutige Identifikation und

daher datenbankseitig als Primärschlüssel. Die notwendige Datenbanktabelle wurde erzeugt durch den SQL-Ausdruck:

```
CREATE TABLE PERSON( Name VARCHAR(20) PRIMARY KEY, Birthdate DATE, Street VARCHAR(30));
```

Wie bereits für die Realisierung von Session Beans eingeführt, werden auch zur Publikation der extern zugänglichen Schnittstellen Home und Remote Interfaces benötigt.

Struktur und Aufbau der Home-Schnittstelle ähnelt konzeptionell der für Session Beans eingeführten. Dieser Schnittstellentyp dient auch für Entity Beans zur Aufnahme der Verwaltungsoperationen zur Erzeugung (`create`) und zur Suche existierender EJBs (`findByPrimaryKey`).
Beispiel 71 zeigt die Home-Schnittstelle des Beispiels.

Beispiel 71: Home-Schnittstelle einer Entity Bean

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.CreateException;
(3)import javax.ejb.EJBHome;
(4)import javax.ejb.FinderException;
(5)
(6)public interface PersonHome extends EJBHome {
(7)     public Person create(int year, int month, int day, String name, String street)
throws CreateException, RemoteException;
(8)     public Person findByPrimaryKey(String name) throws FinderException,
RemoteException;
(9)}
```

Download des Beispiels

Die Home-Schnittstelle zeigt die `create`-Operation zur Erzeugung einer neuen EJB-Instanz. Ihre Übergabeparameter dienen zur Konstruktion des neuen Objekts und werden durch die Bean-Implementierung interpretiert.
Ferner enthält die Schnittstelle mit `findByPrimaryKey` eine Operation, deren Implementierung eine Entity-Bean anhand ihres Primärschlüssels identifiziert und liefert. Aus diesem Grunde erhält die Methode den zu suchenden Wert vom Typ des Primärschlüssels übergeben.
Hinsichtlich der verwendeten Typen zeigt sich bereits hier, daß eine Abbildung der durch die Programmiersprache Java bereitgestellten Typen auf die des eingesetzten Persistenzsystems stattfinden muß. In der im Beispiel gewählten Ausführungsform der durch die EJB selbst verwalteten Persistenz muß diese Abbildung manuell durch den Programmierer bereitgestellt werden.

Die Remote-Schnittstelle gibt die für Nutzer der Bean zugänglichen Geschäftsfunktionen wieder. Daher enthält dieser Schnittstellentyp lediglich Operationen zum Zugriff auf die verwalteten Daten, nicht jedoch zur technischen Verwaltung und Interaktion mit dem Bean-Container.
Per Konvention muß diese Schnittstelle als Spezialisierung der Schnittstelle `EJBObject` definiert sein. Diese Standardschnittstelle definiert allgemeine Interaktionsformen, wie Löschen (`remove`), Vergleich (`isIdentical`) und Ermittlung des Primärschlüsselwertes (`getPrimaryKey`) die für alle Entity Bean Objekte gleichermaßen benötigt werden.
`isIdentical` liefert den Vergleich zweier serverseitiger EJB-Objekte und ermittelt so, ob zwei Java-Objektreferenzen auf dasselbe Datenbankobjekt zugreifen.
`getPrimaryKey` ermittelt den Wert des Primärschlüssels eines gegebenen EJB-Objekts aus der Datenbank.
Zur Lösung von datenbankresidenten Objekten wird `remove` eingesetzt. Der Aufruf dieser Methode entfernt ausschließlich die durch die EJB repräsentierten Datenbanktupel, die programmiersprachliche Objektrepräsentation bleibt jedoch über die gesamte Laufzeit (sofern nicht durch Gültigkeitsbereiche oder explizite `NULL`-Setzung explizit anders gehandhabt) intakt.

Beispiel 72: Remote-Schnittstelle einer Entity Bean

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.EJBObject;
(3)
(4)public interface Person extends EJBObject {
(5)     public int getAge() throws RemoteException;
(6)     public String getStreet() throws RemoteException;
(7)     public void setStreet(String street) throws RemoteException;
(8)}
```

Download des Beispiels

Beispiel 73 zeigt den vollständigen Code der Bean. Sie implementiert mit `EntityBean` die Standardschnittstelle aller Entity Beans, welche als Spezialisierung der ausschließlich markierenden Schnittstelle `EnterpriseBean` die notwendigen Basisoperationen zur Abwicklung der persistenten Speicherung bieten.

Im Falle einer *Bean Managed Persistence* enthalten die Methoden der durch die Schnittstelle definierten und der zusätzlich im Rahmen der Spezifikation textuell definierten Operationen die notwendigen Aufrufe zur Ablage eines Objekts in der Datenbank und zu seiner späteren Extraktion daraus.
Im Einzelnen sind dies die Operationen:

Tabelle 24: Persistenzoperationen einer Entity Bean

Operation	Semantik	Zugehörige SQL-Anweisung
`ejbCreate`	Wird nach dem Erzeugen eines Java-Objektes aufgerufen um dieses in der Datenbank abzulegen. Diese Operation ist nicht Bestandteil der Schnittstelle, da ihre Parameter, die den Übergabeparametern des Objektkonstruktors entsprechen, zum Schnittstellenerzeugungszeitpunkt nicht feststehen.	`INSERT`
`ejbFindByPrimaryKey`	Liefert den Wert des Primärschlüssels zurück, sofern ein Datenbankeintrag existiert, der durch diesen Primärschlüssel identifiziert wird. Diese Operation ist nicht Bestandteil der Schnittstelle, da ihre Parameter, die in Typ, Name und Reihenfolge der Zusammensetzung des Primärschlüssels entsprechen, zum Schnittstellenerzeugungszeitpunkt nicht feststehen. Diese Methode wird nicht durch den Anwender direkt aufgerufen, sondern stattdessen auf einer Ausprägung der Home-Schnittstelle das dort zur Verfügung stehende Analogon `findByPrimaryKey`, welches das durch den Primärschlüssel identifizierte EJB-Objekt zurückliefert. Diese Methode greift intern auf ausschließlich den Schlüssel liefernde Methode der Bean zu.	`SELECT`
`ejbRemove`	Entfernt das EJB-Objekt aus der Datenbank.	`DELETE`
`ejbStore`	Synchronisiert das Java-Objekt mit dem EJB-Objekt und aktualisiert so die Datenbankinhalte. Diese Methode wird nach jedem Zugriff mittels einer in der Remote-Schnittstelle aufgeführten Operation auf das EJB-Objekt ausgeführt.	`UPDATE`

Beispiel 73: Eine Entity Bean

```
(1)import java.rmi.RemoteException;
(2)import java.sql.Connection;
(3)import java.sql.DriverManager;
(4)import java.sql.ResultSet;
(5)import java.sql.SQLException;
(6)import java.sql.Statement;
(7)import java.util.Calendar;
(8)import java.util.GregorianCalendar;
(9)
(10)import javax.ejb.CreateException;
(11)import javax.ejb.EJBException;
(12)import javax.ejb.EntityBean;
(13)import javax.ejb.EntityContext;
(14)import javax.ejb.FinderException;
(15)import javax.ejb.RemoveException;
(16)
(17)public class PersonBean implements EntityBean {
(18)    //persistent fields
(19)    private String name;
(20)    private GregorianCalendar birthdate;
(21)    private String street;
(22)
(23)    public PersonBean() {
(24)    }
(25)
```

```
(26)    //methods which are part of the remote interface
(27)    public int getAge() {
(28)            return (
(29)                    new GregorianCalendar().get(Calendar.YEAR)
(30)                            - birthdate.get(Calendar.YEAR));
(31)    }
(32)    public String getStreet() {
(33)            return street;
(34)    }
(35)    public void setStreet(String street) throws RemoteException {
(36)            if (street.length() <= 30) {
(37)                    this.street = street;
(38)            } else {
(39)                    throw new RemoteException("cannot update street");
(40)            }
(41)    }
(42)    //      ---------------------------------------------------------------------
(43)    public String ejbCreate(
(44)            int year,
(45)            int month,
(46)            int day,
(47)            String name,
(48)            String street)
(49)            throws CreateException {
(50)            if (year > 1900
(51)                    && month >= 1
(52)                    && month <= 12
(53)                    && day >= 1
(54)                    && day <= 31
(55)                    && name.length() <= 20
(56)                    && street.length() <= 30) {
(57)
(58)                    this.name = name;
(59)                    this.birthdate = new GregorianCalendar(year, month, day);
(60)                    this.street = street;
(61)                    Statement stmt = getStatement();
(62)                    String s =
(63)                            new String(
(64)                                    "INSERT INTO PERSON VALUES ('"
(65)                                            + name
(66)                                            + "','"
(67)                                            + birthdate.get(Calendar.YEAR)
(68)                                            + "-"
(69)                                            + birthdate.get(Calendar.MONTH)
(70)                                            + "-"
(71)                                            + birthdate.get(Calendar.DATE)
(72)                                            + "','"
(73)                                            + "'"
(74)                                            + street
(75)                                            + "'"
(76)                                            + ");");
(77)                    try {
(78)                            stmt.executeUpdate(s);
(79)                    } catch (SQLException e) {
(80)                            e.printStackTrace();
(81)                    }
(82)            } else {
(83)                    throw new CreateException("Invalid values supplied");
(84)            }
(85)            return name;
(86)    }
(87)    public void ejbPostCreate(
(88)            int year,
(89)            int month,
(90)            int day,
(91)            String name,
(92)            String street) {
(93)    }
(94)    //      ---------------------------------------------------------------------
(95)    public int ejbHomeGetAge() {
(96)            return 0;
(97)    }
(98)
(99)    public String ejbHomeGetStreet() {
(100)           return new String();
(101)   }
```

```
(102)
(103)   public void ejbHomeSetStreet(String street) {
(104)   }
(105)
(106)   public Statement getStatement() {
(107)           try {
(108)                   Class.forName("com.mysql.jdbc.Driver");
(109)           } catch (ClassNotFoundException e) {
(110)                   e.printStackTrace();
(111)           }
(112)           Connection con = null;
(113)           try {
(114)                   con =
(115)                           (Connection) DriverManager.getConnection(
(116)                                   "jdbc:mysql://10.0.0.1/Address/",
(117)                                   "mario",
(118)                                   "thePassword");
(119)           } catch (SQLException e1) {
(120)                   e1.printStackTrace();
(121)           }
(122)           try {
(123)                   return ((Statement) con.createStatement());
(124)           } catch (SQLException e2) {
(125)                   e2.printStackTrace();
(126)           }
(127)           return null; //never gets here
(128)   }
(129)
(130)   // ------------------------------------------------------------------
(131)   public void setEntityContext(EntityContext ectx)
(132)           throws EJBException, RemoteException {
(133)   }
(134)
(135)   public void unsetEntityContext() throws EJBException, RemoteException {
(136)   }
(137)
(138)   public void ejbRemove()
(139)           throws RemoveException, EJBException, RemoteException {
(140)           Statement stmt = getStatement();
(141)           String s = new String("DELETE FROM PERSON WHERE Name='" + name + "';");
(142)           try {
(143)                   stmt.executeUpdate(s);
(144)           } catch (SQLException e) {
(145)                   e.printStackTrace();
(146)           }
(147)   }
(148)
(149)   public void ejbActivate() throws EJBException, RemoteException {
(150)   }
(151)
(152)   public void ejbPassivate() throws EJBException, RemoteException {
(153)   }
(154)
(155)   public void ejbLoad() throws EJBException, RemoteException {
(156)   }
(157)
(158)   public void ejbStore() throws EJBException, RemoteException {
(159)           Statement stmt = getStatement();
(160)           String s =
(161)                   new String(
(162)                           "UPDATE PERSON SET Name='"
(163)                                   + name
(164)                                   + "', BDATE='"
(165)                                   + birthdate.get(Calendar.YEAR)
(166)                                   + "-"
(167)                                   + birthdate.get(Calendar.MONTH)
(168)                                   + "-"
(169)                                   + birthdate.get(Calendar.DATE)
(170)                                   + "', Street = '"
(171)                                   + street
(172)                                   + "' WHERE Name = '"
(173)                                   + name
(174)                                   + "';");
(175)           try {
(176)                   stmt.executeUpdate(s);
(177)           } catch (SQLException e) {
```

```
(178)                   e.printStackTrace();
(179)           }
(180)   }
(181)
(182)   public String ejbFindByPrimaryKey(String name) throws FinderException {
(183)           Statement stmt = getStatement();
(184)           String s =
(185)                   new String("SELECT Name FROM PERSON WHERE Name='" + name + "';");
(186)           try {
(187)                   ResultSet rs = stmt.executeQuery(s);
(188)                   rs.first();
(189)                   return rs.getString("Name");
(190)           } catch (SQLException e) {
(191)                   e.printStackTrace();
(192)           }
(193)           return null; //never gets here
(194)   }
(195)}
```

Download des Beispiels

Zusätzlich enthält die Bean des Beispiels mit `getAge` eine zwar in der Remote-Schnittstelle veröffentlichte Operation, die keinen direkt abgespeicherten Wert liefert, sondern diesen dynamisch zur Ausführungszeit anhand der verfügbaren Daten berechnet.

Alle anderen in der Remote-Schnittstelle aufgeführten Operationen (etwa: `getStreet`, `setStreet`) modifizieren lediglich, den durch die Attribute repräsentierten Java-Objektzustand und greifen nicht direkt auf die Datenbank zu.

Innerhalb der Datenbankzugreifenden Methoden muß durch den Anwender die Abbildung der Java-Datentypen auf die des verwendeten Datenbankmanagementsystems erfolgen. Die mit dem Präfix `ejb` versehenden Methoden zeigen dies für die lesenden und schreibenden DB-Zugriffe. So kann die im Beispiel für `name` und `street` verwendete Java-Repräsentation `String` vergleichsweise leicht in den SQL-Typ `VARCHAR` abgebildet werden, sofern alle Methoden, die Datenbankinhalte schreiben, sicherstellen, daß nur zum Datenbankschema konforme Werte eingefügt werden. Die Beispielimplementierung zeigt dies examplarisch anhand der Methoden `ejbCreate` und `setStreet`.
Für programmiersprachliche Typen, die nicht direkt in DB-Typen abbildbar sind, muß im Falle der Bean Managed Persistence der Bean-Entwickler selbst Sorge für die adäquate Abbildung tragen. Das Beispiel illustriert dies anhand des Java-Datumstyps `GregorianCalendar`, der manuell in die durch das DBMS erwartete ISO 8601-konforme Darstellung zu überführen ist.

Einige der möglichen Interaktionen mit der Bean zeigt der Code des Clients aus Beispiel 74:

Beispiel 74: Client der auf eine Entity Bean zugreift

```
(1)import java.rmi.RemoteException;
(2)import javax.ejb.CreateException;
(3)import javax.ejb.FinderException;
(4)import javax.ejb.RemoveException;
(5)import javax.naming.Context;
(6)import javax.naming.InitialContext;
(7)import javax.naming.NamingException;
(8)import javax.rmi.PortableRemoteObject;
(9)
(10)public class CallPersonBean {
(11)    public static void main(String[] args) {
(12)            try {
(13)                    Context initial = new InitialContext();
(14)                    Object objRef = initial.lookup("personBean");
(15)
(16)                    PersonHome home =
(17)                            (PersonHome) PortableRemoteObject.narrow(
(18)                                    objRef,
(19)                                    PersonHome.class);
(20)                    Person p1 = home.create(1911, 1, 1, "Alice", "streetA");
(21)                    Person p2 = home.create(1922, 2, 2, "Bob", "streetB");
(22)                    System.out.println("Alice's Age: " + p1.getAge());
(23)
(24)                    System.out.println("Alice's primary key: "+p1.getPrimaryKey());
(25)                    p1.remove();
(26)
(27)                    Person p3 = home.findByPrimaryKey("Bob");
(28)                    System.out.println(
(29)                            "Bob's modified street (before modification): "
(30)                                    + p3.getStreet());
```

```
(31)                    p3.setStreet("streetC");
(32)                    System.out.println(
(33)                            "Bob's modified street (after modification): "
(34)                                    + p3.getStreet());
(35)                    System.out.println("also the other reference: " + p2.getStreet());
(36)
(37)                    System.out.println("Are both references the same Java object: "+
(p2==p3));
(38)                    System.out.println("Are both references the same EJBObject (calls
isIdentical): "+p2.isIdentical(p3));
(39)
(40)            } catch (NamingException ne) {
(41)                    ne.printStackTrace();
(42)            } catch (RemoteException re) {
(43)                    re.printStackTrace();
(44)            } catch (CreateException ce) {
(45)                    ce.printStackTrace();
(46)            } catch (RemoveException e) {
(47)                    e.printStackTrace();
(48)            } catch (FinderException e) {
(49)                    e.printStackTrace();
(50)            }
(51)    }
(52)}
```

Download des Beispiels

Der Client ermittelt zunächst per JNDI eine Referenz auf die Bean, welche unter dem Namen *personBean* im Verzeichnisdienst registriert ist.
Die erhaltene generische Referenz wird durch Aufruf der statischen Methode `narrow` der Klasse `PortableRemoteObject` typsicher in eine Ausprägung der Home-Schnittstelle (`PersonHome`) konvertiert.
Der Aufruf der in dieser Schnittstelle durch den Anwender definierten `create`-Methode sorgt für die serverseitige Instanziierung der EJB, die als Ausprägung der Remote-Schnittstelle geliefert wird.
Tatsächlich wird nicht das EJB-Objekt selbst durch den Methodenaufruf retourniert, sondern lediglich ein netzwerktransparenter Verweis darauf, der jedoch clientseitig einer lokalen Objektreferenz gleichgestellt verwendet werden kann.
Ferner wird serverseitig zur Kommunikation mit der EJB ein Home-Objekt erzeugt, welchem eine Stellvertreterrolle für den anfragenden Client zukommt.
Der Aufruf der durch die EJB zur Verfügung gestellten Methode erfolgt identisch zu dem einer Lokalen.

So dient der Aufruf der Methode `create` zur Erzeugung von serverseitig instanziiert und transparent persistierten EJB-Objekten sowie den lokalen Java-(Stellvertreter-)Objekten für den Zugriff darauf.
Der Aufruf von `getAge` zeigt die Nutzung einer in der Remote-Schnittstelle veröffentlichten Zugriffsmethode. Mit `getPrimaryKey` wird die, in der durch die Remote-Schnittstelle erweiterten Schnittstelle `EJBObject` angesiedelte, Operation zur Ermittlung des Primärschlüsselwertes eines EJB-Objektes aufgerufen.
Die Methode `remove` stellt dagegen eine durch die Home-Schnittstelle definierte Operation dar. Durch den Aufruf dieser Methode auf dem durch `p1` referenzierten Objekt wird durch Ausführung der Beanmethode `ejbRemove` die die Bean serverseitig repräsentierenden Datenbankeinträge entfernt sowie der durch die Bean belegte Speicherbereich als frei markiert. Alle Versuche nach Aufruf dieser Methode auf der clientseitigen hauptspeicherrepräsentation Wertänderungen durchzuführen führen daher zu einem Fehler.
Die Ermittlung von Referenzen auf existierende EJB-Objekte erfolgt durch die in der Remote-Schnittstelle definierte Methode `findByPrimaryKey`. Der EJB-Container stellt sicher, daß verschiedene Referenzen auf dasselbe EJB-Objekt synchronisiert in die Datenbank abgebildet werden, so daß keine Inkonsistenzen entstehen.

Für den Betrieb einer Enterprise Java Bean ist neben den bisher betrachteten Schnittstellen-Komponenten und der Realisierung der Bean selbst auch ein als *Deployment Deskriptor* bezeichnetes XML-Konfigurationsfile notwendig, welches verschiedene Einstellungsdaten sowie die Schnittstellendaten enthält.
Beispiel 75 zeigt ein Beispiel hierfür:

Beispiel 75: Deployment Deskriptor der Entity Bean

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)
(3)<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'
'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>
(4)
(5)<ejb-jar>
(6)  <display-name>Ejb1</display-name>
(7)  <enterprise-beans>
(8)    <entity>
(9)      <display-name>PersonBean</display-name>
(10)     <ejb-name>PersonBean</ejb-name>
(11)     <home>PersonHome</home>
(12)     <remote>Person</remote>
(13)     <ejb-class>PersonBean</ejb-class>
(14)     <persistence-type>Bean</persistence-type>
(15)     <prim-key-class>java.lang.String</prim-key-class>
(16)     <reentrant>False</reentrant>
(17)     <security-identity>
(18)       <description></description>
(19)       <use-caller-identity></use-caller-identity>
(20)     </security-identity>
(21)   </entity>
(22)  </enterprise-beans>
(23)  <assembly-descriptor>
(24)    <method-permission>
(25)      <unchecked />
(26)      <method>
(27)        <ejb-name>PersonBean</ejb-name>
(28)        <method-intf>Remote</method-intf>
(29)        <method-name>getAge</method-name>
(30)        <method-params />
(31)      </method>
(32)      <method>
(33)        <ejb-name>PersonBean</ejb-name>
(34)        <method-intf>Home</method-intf>
(35)        <method-name>remove</method-name>
(36)        <method-params>
(37)          <method-param>javax.ejb.Handle</method-param>
(38)        </method-params>
(39)      </method>
(40)      <method>
(41)        <ejb-name>PersonBean</ejb-name>
(42)        <method-intf>Home</method-intf>
(43)        <method-name>getHomeHandle</method-name>
(44)        <method-params />
(45)      </method>
(46)      <method>
(47)        <ejb-name>PersonBean</ejb-name>
(48)        <method-intf>Remote</method-intf>
(49)        <method-name>isIdentical</method-name>
(50)        <method-params>
(51)          <method-param>javax.ejb.EJBObject</method-param>
(52)        </method-params>
(53)      </method>
(54)      <method>
(55)        <ejb-name>PersonBean</ejb-name>
(56)        <method-intf>Home</method-intf>
(57)        <method-name>remove</method-name>
(58)        <method-params>
(59)          <method-param>java.lang.Object</method-param>
(60)        </method-params>
(61)      </method>
(62)      <method>
(63)        <ejb-name>PersonBean</ejb-name>
(64)        <method-intf>Remote</method-intf>
(65)        <method-name>getHandle</method-name>
(66)        <method-params />
(67)      </method>
(68)      <method>
(69)        <ejb-name>PersonBean</ejb-name>
(70)        <method-intf>Home</method-intf>
(71)        <method-name>findByPrimaryKey</method-name>
(72)        <method-params>
(73)          <method-param>java.lang.String</method-param>
(74)        </method-params>
(75)      </method>
```

```
(76)      <method>
(77)        <ejb-name>PersonBean</ejb-name>
(78)        <method-intf>Home</method-intf>
(79)        <method-name>getEJBMetaData</method-name>
(80)        <method-params />
(81)      </method>
(82)      <method>
(83)        <ejb-name>PersonBean</ejb-name>
(84)        <method-intf>Remote</method-intf>
(85)        <method-name>getPrimaryKey</method-name>
(86)        <method-params />
(87)      </method>
(88)      <method>
(89)        <ejb-name>PersonBean</ejb-name>
(90)        <method-intf>Remote</method-intf>
(91)        <method-name>remove</method-name>
(92)        <method-params />
(93)      </method>
(94)      <method>
(95)        <ejb-name>PersonBean</ejb-name>
(96)        <method-intf>Remote</method-intf>
(97)        <method-name>getEJBHome</method-name>
(98)        <method-params />
(99)      </method>
(100)   </method-permission>
(101)   <container-transaction>
(102)     <method>
(103)       <ejb-name>PersonBean</ejb-name>
(104)       <method-intf>Home</method-intf>
(105)       <method-name>create</method-name>
(106)       <method-params>
(107)         <method-param>int</method-param>
(108)         <method-param>int</method-param>
(109)         <method-param>int</method-param>
(110)         <method-param>java.lang.String</method-param>
(111)         <method-param>java.lang.String</method-param>
(112)       </method-params>
(113)     </method>
(114)     <trans-attribute>Never</trans-attribute>
(115)   </container-transaction>
(116)   <container-transaction>
(117)     <method>
(118)       <ejb-name>PersonBean</ejb-name>
(119)       <method-intf>Remote</method-intf>
(120)       <method-name>setStreet</method-name>
(121)       <method-params>
(122)         <method-param>java.lang.String</method-param>
(123)       </method-params>
(124)     </method>
(125)     <trans-attribute>Never</trans-attribute>
(126)   </container-transaction>
(127)   <container-transaction>
(128)     <method>
(129)       <ejb-name>PersonBean</ejb-name>
(130)       <method-intf>Remote</method-intf>
(131)       <method-name>getStreet</method-name>
(132)       <method-params />
(133)     </method>
(134)     <trans-attribute>Never</trans-attribute>
(135)   </container-transaction>
(136)   <container-transaction>
(137)     <method>
(138)       <ejb-name>PersonBean</ejb-name>
(139)       <method-intf>Home</method-intf>
(140)       <method-name>remove</method-name>
(141)       <method-params>
(142)         <method-param>javax.ejb.Handle</method-param>
(143)       </method-params>
(144)     </method>
(145)     <trans-attribute>Never</trans-attribute>
(146)   </container-transaction>
(147)   <container-transaction>
(148)     <method>
(149)       <ejb-name>PersonBean</ejb-name>
(150)       <method-intf>Home</method-intf>
(151)       <method-name>remove</method-name>
```

```
(152)        <method-params>
(153)          <method-param>java.lang.Object</method-param>
(154)        </method-params>
(155)      </method>
(156)      <trans-attribute>Never</trans-attribute>
(157)    </container-transaction>
(158)    <container-transaction>
(159)      <method>
(160)        <ejb-name>PersonBean</ejb-name>
(161)        <method-intf>Remote</method-intf>
(162)        <method-name>remove</method-name>
(163)        <method-params />
(164)      </method>
(165)      <trans-attribute>Never</trans-attribute>
(166)    </container-transaction>
(167)    <container-transaction>
(168)      <method>
(169)        <ejb-name>PersonBean</ejb-name>
(170)        <method-intf>Remote</method-intf>
(171)        <method-name>getAge</method-name>
(172)        <method-params />
(173)      </method>
(174)      <trans-attribute>Never</trans-attribute>
(175)    </container-transaction>
(176)    <container-transaction>
(177)      <method>
(178)        <ejb-name>PersonBean</ejb-name>
(179)        <method-intf>Home</method-intf>
(180)        <method-name>findByPrimaryKey</method-name>
(181)        <method-params>
(182)          <method-param>java.lang.String</method-param>
(183)        </method-params>
(184)      </method>
(185)      <trans-attribute>Never</trans-attribute>
(186)    </container-transaction>
(187)  </assembly-descriptor>
(188)</ejb-jar>
(189)
```

Download des Beispiels

3.3 Java Data Objects (JDO)

Grundidee

Hintergrund des Ansatzes der *Java Data Objects* (JDO) ist es, die bestehenden Schnittstellenmechanismen dahingehend weiterzuentwickeln, daß die Persistenz von Objekten und Objektgraphen für den Programmierer vollständig transparent durch Komponenten der Laufzeitumgebung zur Verfügung gestellt werden.
Gleichzeitig etabliert JDO eine Abstraktion der verschiedenen Speicherungsmöglichkeiten und erlaubt es beispielsweise die dateibasierte Ablage innerhalb des Programmes identisch zur Objektspeicherung in einem Datenbankmanagementsystem zu handhaben. Auf dieser Basis läßt sich im Bedarfsfalle den Persistenzdienstleister auszutauschen ohne Änderungen am Programmcode zu erfordern.
Plakativ wird der Ansatz daher, in Anlehnung an die Zielsetzung der Programmiersprache Java des *write once -- run anywhere*, als *write once -- store anywhere* charakterisiert.

Technik

Um die weitestgehend transparente Handhabung der Objektpersistenz zu gewährleisten bedient sich JDO eines Ansatzes der über das alleinige Angebot einer Programmierschnittstelle hinausreicht. Die Zielsetzung der möglichst einfach handzuhabenden Interaktion mit den generischen Persistenzmechanismen läßt sich zwar durch das Angebot von durch den Programmierer zu implementierenden Schnittstellen und Persistenzklassen erreichen, jedoch ist der Einsatz signifikant komplexer als der bestehenden Persistenzschnittstellen. Darüberhinaus konterkariert der Zwang bei der Programmerstellung vorgegebene Schnittstellen zu berücksichtigen die Zielsetzung weitestgehender Transparenz der angebotenen Speichermechanismen.

Daher führt JDO die Technik der sog. *Bytecodeanreicherung* (engl. *bytecode enhancing*) ein. Hierbei wird durch eine Programmkomponente vorübersetzer Bytecode so abgeändert, daß die notwendigen Persistenzanweisungen in den bereits erzeugten ausführbaren Bytecode eingewoben werden.

Die benötigte Übersetzerkomponente wird durch die jeweilige JDO-Implementierung zur Verfügung gestellt und muß durch den Programmierer im Bedarfsfalle lediglich geeignet parametrisiert werden.

Im Falle der Referenzimplementierung müssen daher alle Klassen, die Objekte ausprägen, welche persistiert werden sollen, mit dem Werkzeug entsprechend nachbearbeitet werden. Der notwendige Aufruf hat folgende Struktur: `java com.sun.jdori.enhancer.Main -d enhanced de/jeckle/jdotest/Employee.class de/jeckle/jdotest/Employee.jdo`.
Dieser Aufruf reichert die bereits übersetzte Klasse `Emplyoee` innerhalb der Pakethierarchie `de.jeckle.jdotest` um Persistenzdaten an und legt das Ergebnis innerhalb des Dateisystemkatalogs `de/jeckle/jdotest` ab. Zur Anreicherung wird die Konfigurationsdatei `Employee.jdo` herangezogen, die im selben Pfad abgelegt ist wie die Quellcodedatei.

Alternativ zu diesem Ansatz steht auch die Möglichkeit zur Verfügung die benötigten Anweisungen bereits im Quellcode vorzusehen um so dasselbe Resultat zu erzielen, welches durch den Anreicherungsprozeß erzeugt wird. Diese Vorgehensweise hat jedoch wegen der damit verbundenen Aufwände kaum praktische Bedeutung erlangt und wird daher im folgenden nicht vertieft betrachtet.

Die Beispiele dieses Kapitels basieren auf der kostenfrei verfübaren JDO-Referenzimplementierung von SUN. Diese beschränkt zwar die unterstützten Persistenzmechanismen auf ausschließlich dateibasierte Speicherung und sieht keine Ablage in Datenbankmanagmenetsystemen vor.
Konzeptionell und programmierseitig ist die Interaktion mit dieser Implementierung jedoch identisch zu kommerziell verfügbaren Lösungen und können daher ohne weiteres auf diese und damit beliebige Persistenzdienstleister übertragen werden.

Konfiguration des Persistenzdienstleisters

Die Abbildung der in der Programmiersprache formulierten Interaktionen auf den konkreten physischen Persistenzdienstleister erfolgt sinnvollerweise an einer für alle JDO-nutzenden Applikationen zugänglichen Stelle im Rahmen einer Property-Datei.
Die Inhalte dieser Datei unterscheiden naturgemäß bei den verschiedenen JDO-Herstellen und inhärent mit dem gewählten Persistenztyp. So benötigt die dateibasierte Objektablage offenkundig andere Festlegungen als der Zugriff auf ein relationales Datenbankmanagementsystem.
Beispiel 76 zeigt die notwendigen Einstellung zur Konfiguration der dateibasierten Speicherung mit der SUN-Referenzimplementierung. Dort wird mit der `PersistenceManagerFactoryClass` diejenige Klasse innerhalb des JDO-Rahmenwerkes benannt, welche dem Programmierer die Persistenzdienste zur Verfügung stellt. `ConnectionURL` bildet das Bindeglied der Abbildung auf die physische Datei und benennt daher den Speicherort aller persistierten Objekte. Die zusätzlichen Angaben dienen der Authentisierung und Zugriffssteuerung beim Zugriff auf die erstellte Datei.

Beispiel 76: Konfiguration einer JDO-Implementierung

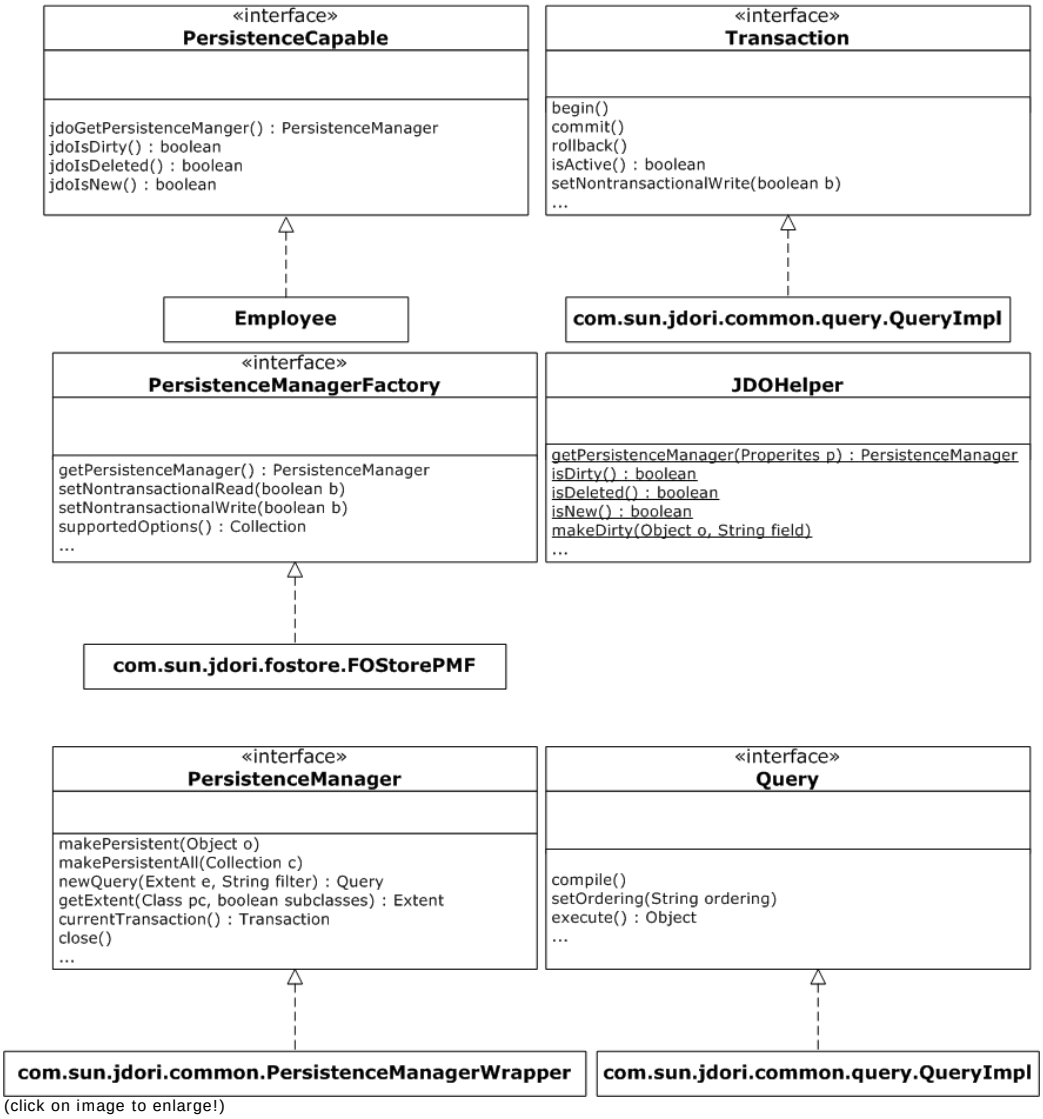
```
(1)javax.jdo.PersistenceManagerFactoryClass=com.sun.jdori.fostore.FOStorePMF
(2)javax.jdo.option.ConnectionURL=fostore:jdoriDB
(3)javax.jdo.option.ConnectionUserName=mario
(4)javax.jdo.option.ConnectionPassword=thePassword
```

Download des Beispiels

Struktur der JDO-API

Die JDO-API ist im Rahmen des Java Community Prozesses als Java-Schnittstellensammlung nebst zugehöriger Semantikdefinition spezifiziert. Die Implementierung der Schnittstellen erfolgt durch den Anbieter der jeweiligen JDO-Implementierung und erfolgt auf den jeweiligen Persistenztyp abgestimmt.
Abbildung 7 zeigt die grundlegenden Schnittstellen der JDO-API sowie die sie anbietenden Klassen der Referenzimplementierung.

Abbildung 7: *Grundlegende Struktur der JDO-API*



(click on image to enlarge!)

Die Schnittstelle `PersistenceCapable` bildet das Rückgrat der gesamten Persistenzbemühungen. Jede Klasse, deren Speicherung durch JDO verwaltet werden soll (in Beispiel die Klasse `Employee`) muß diese Schnittstelle zwingend implementieren.
Typischerweise erfolgt diese Implementierung jedoch nicht direkt durch den Applikationsprogrammierer, sondern wird im Rahmen der Bytecodeanreicherung nachträglich hinzugefügt.
Zur Interaktion mit Klassen, deren Implementierung der in `PersistenceCapable` deklarierten Methoden erst nach dem initialen Übersetzungsvorgang hinzugefügt werden kann der JDO-Anbieter die Hilfsklasse `JDOHelper` anbieten. Diese definiert verschiedene, ausschließlich als statisch deklarierte, Methoden um mit Objekten von Klassen zu operieren, als würden diese die Schnittstelle `PersistenceCapable` umsetzen, ohne deren Klassen zur tatsächlichen Schnittstellenimplementierung verpflichten.
Damit stellt `JDOHelper` die unabdingbare Voraussetzung zur Anwendungsentwicklung unter Verwendung der Bytecodeanreicherung dar, da diese erst nach dem Übersetzungsvorgang Implementierungen derjenigen Schnittstellen hinzufügt, die bereits im Code verwendeten wurden. Ferner bietet die Klasse die Möglichkeit den aktuellen Persistenzzustand eines JDO-verwalteten Objektes auszulesen.

Zur Erzeugung von Objekten, die später den Zugriff auf das physische Speichermedium regeln dienen die Umsetzungen der Schnittstelle `PersistenceManagerFactory`. Sie erlaubt die Parametrisierung und Verwaltung der Verbindung zum Persistenzmedium. Bereitgestellt wird die Implementierung, im Falle der Referenzimplementierung, durch die Klasse `com.sun.jdori.fostore.FOStorePMF`.
Die Verbindung zwischen Schnittstelle und tatsächlicher Implementierung wird im Rahmen der in Beispiel 76 gezeigten JDO-Konfiguration definiert. Zum Wechsel des Persistenzanbieters -- etwa von der durch die Referenzimplementierung angebotenen dateibasierten Speicherung auf eine datenbankgestützte Umsetzung -- genügt im die Abänderung dieses Eintrages in der Konfigurationsdatei.

Klassen, welche die Schnittstelle `PersistenceManagerFactory` implementieren, werden zur Erzeugung von sog. `PersistenceMangern` herangezogen. Umsetzungen dieser Schnittstelle (im Falle der Referenzimplementierung ist dies die Klasse `com.sun.jdori.common.PersistenceManagerWrapper`) dienen zur Interaktion mit der Persistenzveraltung innerhalb der JDO nutzenden Applikation. Alle Änderungen des Zustandes eines persistenten Objektes werden durch diese Klasse abgewickelt.

JDO wickelt sämtliche Zugriffe auf die persistenten Daten transaktionsgesichert ab. Dieser Mechanismus wird auf der abstrakten Ebene der API durch die Schnittstelle `Transaction` definiert und steht daher für alle Persistenzanbieter gleichermaßen zur Verfügung.

Die Schnittstelle definiert alle zur Transaktionssteuerung benötigten Operationen (darunter `begin`, `commit` und `rollback`) an.
Im Falle der Referenzimplementierung wird die Schnittstelle durch die Klasse `com.sun.jdori.common.query.QueryImpl` umgesetzt.

Zusätzlich sieht JDO eine abstrakte Möglichkeit zur Formulierung von Anfragen auf den verwalteten Datenbestand vor. Die notwendige Schnittstelle wird durch `Query` bereitgestellt.
Hierfür müssen die verschiedenen JDO-Implementierungen ebenfalls eigene Umsetzungen vorsehen.

Erzeugen eines persistenten Objektspeichers

Zur Erzeugung eines Objektspeichers ist bereits die Nutzung der Implementierungen der zentralen JDO-Schnittstellen sowie die der Transaktionssteuerung notwendig.
Das Beispiel zeigt die notwendigen Schritte zur Erzeugung eines persistenten Objektspeichers.
Zunächst lädt das Beispiel die Konfiguration aus der Eigenschaftsdatei des Beispiels 76. Anschließend wird durch die mit `true` belegte implementierungsspezifische Eigenschaft `com.sun.jdori.option.ConnectionCreate` festgelegt, daß im Rahmen des Verbindungsaufbaus auch notwendigenfalls der Objektspeicher neu erzeugt wird.
Die Interaktion mit JDO beginnt durch die Erzeugung eines `PersistenceManagerFactory` konformen Objektes durch den Aufruf `getPersistenceManagerFactory` unter Auswertung der zuvor geladenen und ergänzten Konfigurationseigenschaften.
Nach der Erzeugung des Factory-Objektes kann mittels diesem durch den Aufruf `getPersistenceManager` ein Objekt erzeugt werden, das die Interaktion mit dem Objektspeicher bereitstellt. Durch die Ermittlung des Persistenzmanagers wird gleichzeitig eine Verbindung zum Persistenzanbieter aufgebaut.
Ausgehend von diesem Verwaltungsobjekt kann durch Definition einer „leeren“ Transaktion -- d.h. einer Transaktion, die jenseits der Erzeugung des transaktionalen Kontexts und seines Abschlusses mit *committ*, keine Operationen definiert -- der Objektspeicher erzeugt werden.
Den Abschluß der Interaktion mit dem Objektspeicher bildet die Beendigung der Verbindung durch Ausführung der Methode `close` des Verbindungsobjektes.

Beispiel 77: Erzeugung eines persistenten Objektspeichers

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Properties;
(4)
(5)import javax.jdo.JDOHelper;
(6)import javax.jdo.PersistenceManager;
(7)import javax.jdo.PersistenceManagerFactory;
(8)import javax.jdo.Transaction;
(9)
(10)public class JDOCreateDB {
(11)    public static void main(String args[]) {
(12)            Properties props = new Properties();
(13)            try {
(14)                    InputStream is = ClassLoader.getSystemResourceAsStream("jdo.
properties");
(15)                    props.load(is);
(16)                    props.put("com.sun.jdori.option.ConnectionCreate","true");
(17)            } catch (IOException ioe) {
(18)                    System.out.println("Error loading properties");
(19)                    System.exit(1);
(20)            }
(21)            PersistenceManagerFactory pmf = JDOHelper.getPersistenceManagerFactory
(props);
(22)            PersistenceManager pm = pmf.getPersistenceManager();
(23)
(24)            Transaction tx = pm.currentTransaction();
(25)            tx.begin();
(26)            tx.commit();
(27)            pm.close();
(28)    }
(29)}
```

Download des Beispiels

Parametrisierung der Persistenz

Grundsätzlich können Ausprägungen jeder beliebigen Javaklasse durch JDO persistiert werden, solange diese Klassen die Schnittstelle `PersistentCapable` explizit im Quellcode implementieren oder die benötigte Implementierung im Rahmen der Bytecodeanreicherung hinzugefügt wird.
Zur Steuerung des konkreten Persistenzverhaltens wird eine zusätzliche Konfigurationsdatei benötigt. Diese bedient sich der bekannten XML-Sytnax und definiert das Persistenzverhalten der durch JDO zu

verwaltenden Klasseninstanzen näher.
Beispiel 78 zeigt zunächst die zu persistierende Klasse Employee.

Beispiel 78: Zu persistierende Javaklasse

```
(1)package de.jeckle.jdotest;
(2)
(3)import java.util.HashSet;
(4)import java.util.Iterator;
(5)
(6)public class Employee {
(7)     private String name;
(8)     private String department;
(9)     private HashSet projects = new HashSet();
(10)
(11)    public String getName() {
(12)            return name;
(13)    }
(14)    public String getDepartment() {
(15)            return department;
(16)    }
(17)    public void setName(String name) {
(18)            this.name = name;
(19)    }
(20)    public void setDepartment(String department) {
(21)            this.department = department;
(22)    }
(23)    public void addProject(String project) {
(24)            projects.add(project);
(25)    }
(26)
(27)    public String toString() {
(28)            String result="Employee named "+name+" works in department "+department;
(29)            result+="\nworks in: ";
(30)            Iterator i = projects.iterator();
(31)            while (i.hasNext()) {
(32)                    result+=(String) i.next();
(33)                    result+=", ";
(34)            }
(35)            return (result);
(36)    }
(37)}
```

Download des Beispiels

Die Nutzung JDO-gestützter Objektpersistenz impliziert keinerlei Modifikationen oder Ergänzungen am Quellcode. Ebenso sind keinerlei Umsetzungskonventionen einzuhalten, die im Beispiel definierten get- und set-Methoden dienen lediglich der vereinfachten Interaktion.

Das Beispiel 79 illustriert eine Parameterdatei zur Definition des spezifischen Persistenzverhaltens von Objekten der Klasse Employee.

Beispiel 79: Parametrisierung der Objektpersistenz

```
(1)<?xml version="1.0"?>
(2)<!DOCTYPE jdo PUBLIC "-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 1.0//
EN" "http://java.sun.com/dtd/jdo_1_0.dtd">
(3)<jdo>
(4)     <package name="de.jeckle.jdotest">
(5)             <class name="Employee">
(6)                     <field
(7)                             name="name"
(8)                             persistence-modifier="persistent"/>
(9)                     <field
(10)                            name="department"
(11)                            persistence-modifier="persistent"/>
(12)                    <field
(13)                            name="projects">
(14)                            <collection
(15)                                    element-type="java.lang.String"
(16)                                    embedded-element="true"/>
(17)                    </field>
(18)            </class>
(19)    </package>
(20)</jdo>
```

Download des Beispiels

Die XML-Datei definiert zunächst den Paket- und Klassennamen der zu persistierenden Klasse mittels des Attributs `name` der XML-Elemente `package` und `class`.
Innerhalb eines `class`-Elements kann für jedes Attribut der Javaklasse ein mit `field` benanntes Element zur näheren Charakterisierung des Speicherungsverhaltens angegeben werden.
Ein solches Element trägt zunächst im Attribut `name` den klassenweit eindeutigen Namen des Attributs und erlaubt die Festlegung des spezifischen Persistenzverhaltes mittels der Belegung des Attributs `persistence-modifier`. Ist dieses mit dem Wert `persistent` versehen, so wird ein so gekennzeichnetes Attribut durch JDO im Datenspeicher persistiert. Trägt das XML-Attribut den Wert none, so wird das Javaattribut bei der Abbildung in den JDO-Datenspeicher ignoriert.
Zusätzlich besteht die Möglichkeit durch die Belegung mit `transactional` die Zwischenspeicherung des Attributwertes während der Abarbeitung einer Transaktion zu erzwingen, um so eine spätere Wiederherstellung (nach einem Aufruf von `rollback`) zu gewährleisten. Jedoch werden Felder, die so gekennzeichnet sind, nicht persistent in den Datenspeicher übernommen, sondern stehen nur während der Programmlaufzeit zur Verfügung.
Fehlt diese Spezifikation zu einem Attribut in der XML-Datei, so wird vorgabegemäß die Belegung mit `persistent` angenommen, sofern es in der beherbergenden Javaklasse nicht als `static`, `transient` oder `final` ausgewiesen ist.

Attribute vom Typ einer Sammlungsklasse, wie sie durch die *Collection API* definiert werden müssen zusätzlich mit einem `collection`-Element, welches innerhalb des `field`-Elements plaziert ist, charakterisiert. Das `collection`-Element spezifiziert durch sein Attribut `element-type` den Typ der Elemente in der Sammlung festlegt. Zusätzlich kann durch das Boole'sche-Attribut `embedded-element` gesteuert werden, ob die Inhalte des Sammlungsobjektes zusammen mit dem die Sammlung referenzierenden Objekt persistiert werden sollen.

Das Beispiel legt für alle Attribute der Klasse `Employee` ihre persistente Speicherung fest (Belegung des XML-Attributs `persistence-modifier` für alle Attribute `persistent`); ebenso wird die in Objekten des Typs *Employee*, unter dem Namen `projects`, enthaltene Sammlungsinstanz einschließlich ihrer Inhaltsobjekte des Standard-API-Typs `String` dauerhaft abgespeichert.

Über diese Festlegungen hinaus gestattet das Parametrisierungsformat die Festlegung spezifischer Konsistenzsemantik in Gestalt der Auszeichnung eines *Primärschlüssels*. Dieses aus dem relationalen Modell bekannte Konstrukt fordert die Eindeutigkeit eines Attributs oder einer Kombination von Attributen über die gesamte Menge der Ausprägungen eines Typs.
Durch die Unterstützung als abstraktes JDO-Konstrukt steht dieses Konzept zur Konsistenzsicherung auch für Applikationen zur Verfügung, die sich nicht relationaler Datenbanken als Persistenzdienstleister bedienen.
Zur Realisierung des Primärschlüsselkonzeptes ist das als Schlüssel zu interpretierende Attribut in der XML-Beschreibung zusätzlich mit dem XML-Attribut `primary-key` zu versehen, welches den Wert `true` tragen muß. Zusätzlich ist innerhalb des Elements `class` diejenige Klasse anzugeben, welche das Attribut beherbergt, das als Schlüssel herangezogen werden soll.
80 zeigt die notwendigen Modifikationen an der Parameterdatei des Beispiels 79 um das Java-Attribut `name` als Primärschlüssel festzulegen. Die primärschlüsselanbietende Klasse ist in diesem Falle die Klasse `Employee` selbst, weshalb sich ihr Name auch im XML-Attribut `objectid-class` des `class`-Elements findet.

Beispiel 80: Parametrisierung der Objektpersistenz und Definition eines Primärschlüssels

```
(1)<?xml version="1.0"?>
(2)<!DOCTYPE jdo PUBLIC "-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 1.0//
EN" "http://java.sun.com/dtd/jdo_1_0.dtd">
(3)<jdo>
(4)     <package name="de.jeckle.jdotest">
(5)             <class name="Employee"
(6)                        objectid-class="Employee">
(7)                     <field
(8)                             primary-key="true"
(9)                             name="name"
(10)                            persistence-modifier="persistent"/>
(11)                    <field
(12)                            name="department"
(13)                            persistence-modifier="persistent"/>
(14)                    <field
(15)                            name="projects">
(16)                            <collection
(17)                                    element-type="java.lang.String"
(18)                                    embedded-element="true"/>
(19)                    </field>
(20)            </class>
(21)    </package>
(22)</jdo>
```

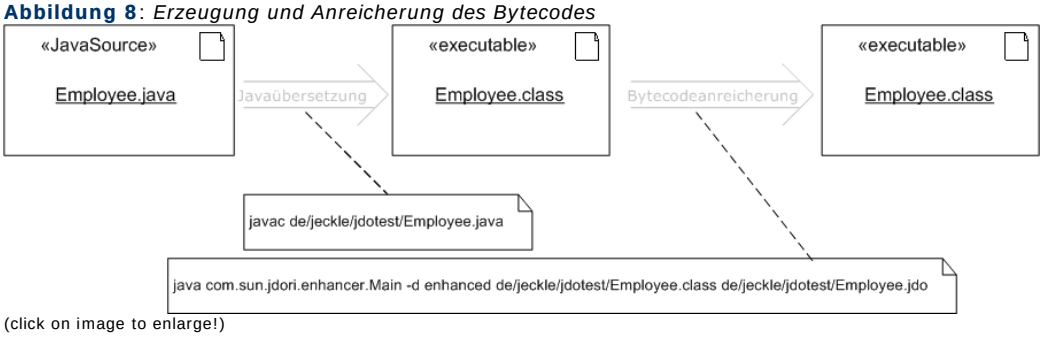
Download des Beispiels

Konsequenz der Einführung eines Primärschlüsselattributs ist die Überwachung der damit einhergehenden Konsistenzbedingungen durch das JDO-Laufzeitsystem. So führen Versuche zwei Objekte, die sich in der Belegung des als Primärschlüssel definierten Attributs nicht unterscheiden ebenso zu Fehlern wie schreibende Zugriffe auf dergestalt ausgezeichnete Attribute.

Anreicherung des Bytecodes

Voraussetzung der Persistenzverwaltung eines Objektes durch JDO ist die entsprechende Modifikation dieses Objektes, konkret die Implementierung der in `PersistenceCapable` festgelegten Operationen durch Methoden der objekterzeugenden Klasse.
Dies wird jedoch nur in Ausnahmefällen durch den Applikationsprogrammierer direkt vorgenommen. Häufigste Eionsatzform der JDO-API ist die Anwendung der Bytecodeanreicherung, welche die Implementierung der notwendigen Funktionalität automatisiert vornimmt und diese nach dem eigentlichen Übersetzungsvorgang in den erstellten Bytecode einbringt.
Abbildung 8 zeigt die daher notwendigen zwei Übersetzungsschritte.

Abbildung 8: *Erzeugung und Anreicherung des Bytecodes*



(click on image to enlarge!)

Die Illustration versammelt die zur Erzeugung und Anreicherung des Bytecodes der per JDO zu persistierenden Klasse `Employee` aus Beispiel 78. Zur Anreicherung des Bytecodes werden die in Beispiel 79 getroffenen Parametrisierungen herangezogen.
Zunächst wird der im Paket `de.jeckle.jdotest` abgelegte Quellcode `Employee.java` mit dem Javacompiler in (gewöhnlichen) Bytecode übersetzt.
Anschließend wird dieser vermöge des in der JDO-Referenzimplementierung vorhandenen Werkzeuges `Enhancer` um die Implementierung der in der Schnittstelle `PersistenceCapable` definierten Operationen angereichert. Hierzu wird dem Enhancer (bereitgestellt durch die Klasse `com.sun.jdori.enhancer.Main` zunächst das Zielverzeichnis des zu erzeugenden Bytecodes mittels des Parameters `d` übergeben. Naheliegernderweise kann der aus dem ursprünglichen Bytecode durch Erweiterung erzeugte nicht die Ausgangsdatei überschreiben, daher wird der angereicherte Bytecode im Verzeichnis `enhanced` gespeichert. Zusätzlich ist dem Enhancer der vollqualifizierte Name der anzureichernden Klasse sowie der vollqualifizierte Pfad der Parameterdatei (im Beispiel: `de/jeckle/jdotest/Employee.jdo`) zu übergeben. Diese muß im Falle des Einsatzes der Referenzimplementierung zwingend die Extension `jdo` besitzen.

Status JDO-verwalteter Objekte

Im Zusammenspiel zwischen transienter Objektverwaltung durch die Applikation im Hauptspeicher und persistenter Objektverwaltung durch JDO im Hintergrundspeicher werden verschiedene Status eines verwalteten Objekts unterschieden zwischen denen explizite Übergänge durch API-Aufrufe vorgegeben sind bzw. implizit durch Operationen auf den involvierten Objekten bestehen.
Im Detail werden folgende Status unterschieden:

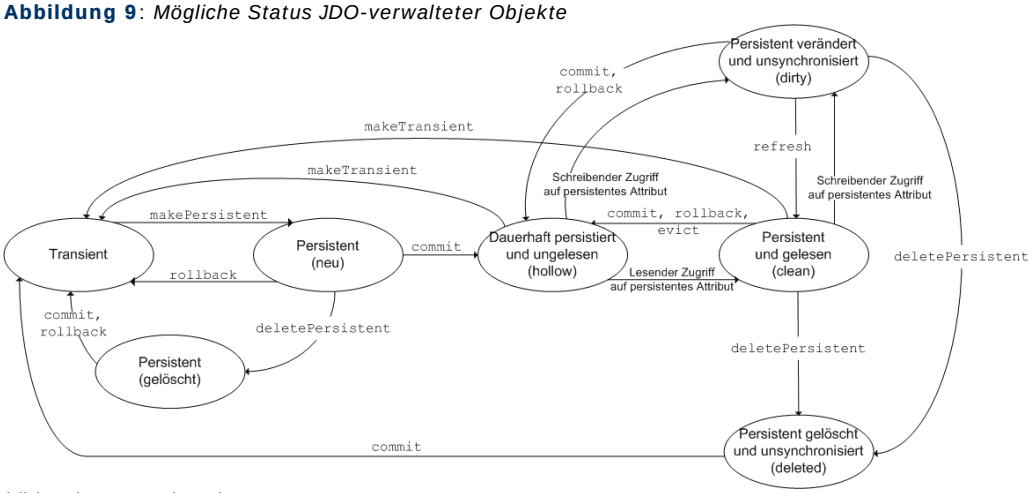
- **Transient**: Instantiierte Objekte im Hauptspeicher. Hierunter fallen alle noch nicht innerhalb von Transaktionen persistierten Objekte ebenso wie unangereicherte Javaobjekte und solche die ausschließlich über Attribut verfügen, die als in der XML-Parametrisierungsdatei als `transient` gekennzeichnet sind.
- **Persistent (neu)**: Objekte, die innerhalb einer laufenden (d.h. weder durch `commit` noch `rollback` abgeschlossenen) Transaktion erzeugt wurden.
  Endet eine Transaktion, etwa durch Programmabbruch, in diesem Zustand, so werden die Objekte mit diesem Status nicht dauerhaft gespeichert.
- **Persistent (gelöscht)**: Objekte, die in einer noch nicht abgeschossenen Transaktion persistiert und anschließend gelöscht wurden.
  Endet eine Transaktion, etwa durch Programmabbruch, in diesem Zustand, so werden die Objekte mit diesem Status nicht dauerhaft gespeichert, da sowohl der Persistierungs- als auch der anschließende Löschvorgang noch nicht durch `commit` bestätigt wurden.
- **Dauerhaft persistiert und ungelesen (hollow)**: Objekte, die durch Abschluß einer Transaktion mit `commit` dauerhaft gespeichert wurden und auf die noch kein Zugriff (weder lesend noch schreibend) erfolgte.
- **Persistent und gelesen (clean)**: Objekte, die persistent im Hintergrundspeicher abgelegt wurden und auf die bisher lediglich lesende Zugriffe erfolgten.
- **Persistent verändert und unsynchronisiert (dirty)**: Persistentes Objekt, dessen Inhalt im Rahmen einer noch nicht abgeschlossenen Transaktion verändert wurde.
  Wurden zwar Attributinhalte eines persistenten Objektes verändert, jedoch keine Wertänderungen

vorgenommen, d.h. in ein Attribut wird mit demselben Wert belegt, den es bereits enthält, dann steht es dem JDO-Implementierer frei diese Schreiboperation so zu implementieren, daß nicht der Zustand *dirty* eingenommen wird.
Zusätzlich kann jedes Objekt durch Aufruf der API-Methode `makeDirty` manuell in diesen Zustand versetzt werden.
Als Folge der Schreiboperation im Hauptspeicher differieren dessen Inhalte von denen des persistenten Objektspeichers.
Durch Aufruf der API-Methode `refresh` werden die Inhalte von Haupt- und Hintergrundspeicher synchronsiert, d.h. Inhalte des Hintergrundspeichers werden in den Hauptpeicher übernommen.

- **Persistent gelöscht und unsynchronisiert (deleted)**: Persistentes Objekt, das im Rahmen einer noch nicht abgeschlossenen Transaktion gelöscht wurde.
  Als Folge der Löschoperation im Hauptspeicher differieren dessen Inhalte von denen des persistenten Objektspeichers und alle Leseoperationen auf nicht-Primärschlüsselfelder führen zu Laufzeitfehlern.

Abbildung 9 zeigt die verschiedenen JDO-Status sowie die Ereignisse, die zu Zustandsübergängen führen, in der Übersicht.

Abbildung 9: *Mögliche Status JDO-verwalteter Objekte*



(click on image to enlarge!)

Speicherung von Objekten

Zur Speicherung von Objekten, deren Klassen durch den Bytecodeanreicherungsprozeß nachbearbeitet wurden, bietet die JDO-API die Aufrufe `makePersistent` und `makePersistentAll` an. Diese werden innerhalb eines Transaktionskontextes als Methoden eines `PersistenceManager`-Objekte ausgeführt.
Beispiel 81 zeigt die Speicherung von drei Objekten der Klasse `Emplyoee`, deren übersetzter Bytecode durch Anreicherung zur JDO-Kompatibilität modifiziert wurde.
Zunächst wird mit `empCol` vom Standardtyp `Vector` eine Sammlungsobjekt zur Aufnahme von Objektreferenzen definiert. Dieser Objektsammlung werden die Referenzen auf die erzeugten `Emplyoee`-Objekte (`emp1`, `emp2` und `emp3`) hinzugefügt.
Als Voraussetzung der Interaktion mit dem Objektspeicher muß zunächst eine Transaktion eröffnet werden. Hierzu muß zunächst durch Aufruf der Methode `currentTransaction` die der `PersistenceManager`-Instanz zugeordnete Transaktion ermittelt werden. Ausgehend vom gelieferten Ergebnisobjekt kann durch Ausführung der Methode `begin` eine neuer Transaktionskontext eröffnet werden.
Der Aufruf von `makePersistentAll` persistiert bei Übergabe der Objektsammlung alle in der Sammlung referenzierten Objekte. Alternativ können Einzelobjekte durch die Methode `makePersistent` in den Zustand dauerhafter Speicherung überführt werden.
Zur Übernahme in den Hintergrundspeicher muß der Transaktionskontext durch Aufruf von `commit` abgeschlossen werden. Der Aufruf von `rollback` würde stattdessen alle in der Transaktion vorgenommenen Änderungen verwerfen und auf den im Hintergrundspeicher verwalteten Datenzustand zurückgesetzt.

Beispiel 81: Speicherung von Objekten mit JDO

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Properties;
(4)import java.util.Vector;
(5)
(6)import javax.jdo.JDOHelper;
(7)import javax.jdo.PersistenceManager;
(8)import javax.jdo.PersistenceManagerFactory;
(9)
(10)import de.jeckle.jdotest.Employee;
(11)
(12)public class JDOStoreObj {
(13)    public static void main(String args[]) {
(14)            Properties props = new Properties();
```

```
(15)            try {
(16)                    InputStream is =
(17)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
(18)                    props.load(is);
(19)            } catch (IOException ioe) {
(20)                    System.out.println("Error loading properties");
(21)                    System.exit(1);
(22)            }
(23)            PersistenceManagerFactory pmf =
(24)                    JDOHelper.getPersistenceManagerFactory(props);
(25)            PersistenceManager pm = pmf.getPersistenceManager();
(26)
(27)            Vector empCol = new Vector();
(28)
(29)            Employee emp1 = new Employee();
(30)            emp1.setName("Mario Jeckle");
(31)            emp1.setDepartment("D001");
(32)            emp1.addProject("P001");
(33)            emp1.addProject("P002");
(34)            empCol.add(emp1);
(35)
(36)            Employee emp2 = new Employee();
(37)            emp2.setName("John DoeX");
(38)            emp2.setDepartment("D003");
(39)            emp2.addProject("P001");
(40)            emp2.addProject("P042");
(41)            empCol.add(emp2);
(42)
(43)            Employee emp3 = new Employee();
(44)            emp3.setName("John Doe");
(45)            emp3.setDepartment("B042");
(46)            empCol.add(emp3);
(47)
(48)            Employee emp4 = new Employee();
(49)            emp4.setName("Barnie Bar");
(50)            emp4.setDepartment("B042");
(51)
(52)            pm.currentTransaction().begin();
(53)            pm.makePersistentAll(empCol);
(54)            pm.makePersistent(emp4);
(55)
(56)            pm.currentTransaction().commit();
(57)            pm.close();
(58)    }
(59)}
```

Download des Beispiels

Würde wie im Beispiel 80 gezeigt das Attribut `name` der Klasse `Employee` als Primärschlüssel definiert sein, so würde der Persistierungsversuch des durch `emp3` referenzierten Objektes einen Laufzeitfehler liefern, da mit `emp2` bereits ein Objekt mit derselben Belegung des Attributs `name` persistiert wurde.

Rücksetzen von Transaktionen

Treten während der Interaktion mit dem Persistenzspeicher, d.h. während eines noch nicht mit `commit` abgeschlossenen Transaktionskontextes Fehler auf, so können durch Aufruf der Methode `rollback` alle im aktuellen Kontext vorgenommen Änderungen auf den Stand vor Beginn der Transaktion zurückgesetzt werden.
Beispiel 82 zeigt das Verhalten der Methode `rollback` am Beispiel. Durch die Schreiboperation innerhalb der geöffneten Transaktion wird der Wert des Attributs `name` zwar verändert, jedoch durch Aufruf von `rollback` wieder auf den ursprünglichen Wert zurückgesetzt.

Beispiel 82: Transaktionen mit JDO

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Iterator;
(4)import java.util.Properties;
(5)
(6)import javax.jdo.JDOHelper;
(7)import javax.jdo.PersistenceManager;
(8)import javax.jdo.PersistenceManagerFactory;
(9)
(10)import de.jeckle.jdotest.Employee;
(11)
(12)public class JDORollback {
(13)
(14)    public static void main(String args[]) {
(15)            Properties props = new Properties();
(16)            try {
(17)                    InputStream is =
(18)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
(19)                    props.load(is);
(20)            } catch (IOException ioe) {
(21)                    System.out.println("Error loading properties");
(22)                    System.exit(1);
(23)            }
(24)            PersistenceManagerFactory pmf =
(25)                    JDOHelper.getPersistenceManagerFactory(props);
(26)            PersistenceManager pm = pmf.getPersistenceManager();
(27)
(28)            Employee e = new Employee();
(29)            e.setName("Marta Mayer");
(30)
(31)            pm.currentTransaction().begin();
(32)            pm.makePersistent(e);
(33)            pm.currentTransaction().commit();
(34)            displayPersistedObjects(pm);
(35)
(36)            System.out.println("Martha gets married and changes her name");
(37)
(38)            pm.currentTransaction().begin();
(39)            e.setName("Marta Smith");
(40)            pm.makePersistent(e);
(41)            displayPersistedObjects(pm);
(42)            System.out.println("Suppose and error happens now ...\nRolling back");
(43)            pm.currentTransaction().rollback();
(44)
(45)            displayPersistedObjects(pm);
(46)    }
(47)    private static void displayPersistedObjects(PersistenceManager pm) {
(48)            Iterator i = pm.getExtent(Employee.class, false).iterator();
(49)            while (i.hasNext()) {
(50)                    System.out.println((Employee) i.next());
(51)            }
(52)    }
(53)}
```

Download des Beispiels

Die Ausführung des Beispiels liefert folgende Ausgabe:

```
Employee named Marta Mayer works in department null
works in projects:
Martha gets married and changes her name
Employee named Marta Smith works in department null
works in projects:
Suppose and error happens now ...
Rolling back
Employee named Marta Mayer works in department null
works in projects:
```

Schreiboperationen ohne Transaktionsschutz

Ist in bestimmten Anwendungsfällen die Arbeit ohne Transaktionsschutz -- und damit ohne die Möglichkeit der expliziten Rücksetzung von Änderungen mittels `rollback` oder der impliziten Rücksetzung nach einem

Systemausfall -- gewünscht, so kann dies durch Aktivierung der Schreibfunktionalität ohne Transaktionsschutz erreicht werden.
Hierzu muß de Methode `setNontransactionalWrite` mit dem Übergabeparameter `true` für eine Transaktion aufgerufen werden.
Das nachfolgende Beispiel zeigt als Modifikation von Beispiel 81 die persistente Übernahme einer Wertänderung ohne Transaktionsschutz.

Beispiel 83: Schreiboperation ohne Transaktionsschutz

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Iterator;
(4)import java.util.Properties;
(5)
(6)import javax.jdo.JDOHelper;
(7)import javax.jdo.PersistenceManager;
(8)import javax.jdo.PersistenceManagerFactory;
(9)
(10)import de.jeckle.jdotest.Employee;
(11)public class JDONonTransact {
(12)    public static void main(String args[]) {
(13)            Properties props = new Properties();
(14)            try {
(15)                    InputStream is =
(16)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
(17)                    props.load(is);
(18)            } catch (IOException ioe) {
(19)                    System.out.println("Error loading properties");
(20)                    System.exit(1);
(21)            }
(22)            PersistenceManagerFactory pmf =
(23)                    JDOHelper.getPersistenceManagerFactory(props);
(24)            PersistenceManager pm = pmf.getPersistenceManager();
(25)
(26)            Employee e = new Employee();
(27)            e.setName("Marta Mayer");
(28)            pm.currentTransaction().setNontransactionalWrite(true);
(29)
(30)            pm.currentTransaction().begin();
(31)            pm.makePersistent(e);
(32)            pm.currentTransaction().commit();
(33)            displayPersistedObjects(pm);
(34)
(35)            //martha gets married and changes her name
(36)
(37)            e.setName("Marta Smith");
(38)
(39)            displayPersistedObjects(pm);
(40)    }
(41)    private static void displayPersistedObjects(PersistenceManager pm) {
(42)            Iterator i = pm.getExtent(Employee.class, false).iterator();
(43)            while (i.hasNext()) {
(44)                    System.out.println((Employee) i.next());
(45)            }
(46)    }
(47)}
```

Download des Beispiels

Traversierung des persistenten Objektbestandes

Zugriffe auf alle im Hintergrundspeicher verwalteten Objekte werden ebenfalls einheitlich durch Methoden der Implementierung der Schnittstelle `PersistenceManager` abgewickelt. Zur Traversierung des vollständigen Bestandes aller Instanzen einer Klasse bietet diese Schnittstelle die Operation `getExtent` an. Sie liefert alle Elemente der Extension (d.h. der Gesamtheit von Ausprägungen) einer gegebenen Klasse.
Beispiel 84 zeigt die Verwendung der Methode. Als Parameter wird diejenige Klasse übergeben, deren Ausprägungen zu ermitteln sind. Zusätzlich kann durch einen Boole'schen Schalter gesteuert werden, ob auch Subklassen der übergebenen Klasse retourniert werden sollen.
Der Aufruf liefert eine Sammlung von Objekten des Typs, welcher der Methode `getExtent` übergeben wurde.

Beispiel 84: Traversierung des Objektbestandes

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Iterator;
(4)import java.util.Properties;
(5)
(6)import javax.jdo.JDOHelper;
(7)import javax.jdo.PersistenceManager;
(8)import javax.jdo.PersistenceManagerFactory;
(9)
(10)import de.jeckle.jdotest.Employee;
(11)
(12)public class JDOListObj {
(13)    public static void main(String args[]) {
(14)            Properties props = new Properties();
(15)            try {
(16)                    InputStream is =
(17)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
(18)                    props.load(is);
(19)            } catch (IOException ioe) {
(20)                    System.out.println("Error loading properties");
(21)                    System.exit(1);
(22)            }
(23)            PersistenceManagerFactory pmf =
(24)                    JDOHelper.getPersistenceManagerFactory(props);
(25)            PersistenceManager pm = pmf.getPersistenceManager();
(26)
(27)            Iterator i = pm.getExtent(Employee.class, false).iterator();
(28)            while (i.hasNext()) {
(29)                    System.out.println((Employee)i.next());
(30)            }
(31)    }
(32)}
```

Download des Beispiels

Anfragen an den persistenten Objektbestand

Als mächtige Alternative zur manuellen Traviersierung einer Objektextension spezifiziert JDO die Verwendung einer eigenen Anfragesprache auf Basis des Standards der *Object Query Language* (OQL) der Object Database Management Group (ODMG).
Diese -- als *JDO Object Query Language* (JDOQL) bezeichnete -- Anfragesprache ist direkt in die JDO-API integriert und wird über verschiedene Einzelmethoden genutzt. Aus diesem Grunde sind JDOQL-Anfragen nicht direkt mit den konsizsen SQL- oder OQL-Anfragen vergleichbar.
Beispiel 85 zeigt die Einbettung der Anfragesprache in die JDO-API.

Beispiel 85: Anfrage auf den persistenten Objektbestand mittels OQL

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Collection;
(4)import java.util.Iterator;
(5)import java.util.Properties;
(6)
(7)import javax.jdo.Extent;
(8)import javax.jdo.JDOHelper;
(9)import javax.jdo.PersistenceManager;
(10)import javax.jdo.PersistenceManagerFactory;
(11)import javax.jdo.Query;
(12)
(13)import de.jeckle.jdotest.Employee;
(14)public class JDOQuery {
(15)    public static void main(String args[]) {
(16)            Properties props = new Properties();
(17)            try {
(18)                    InputStream is =
(19)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
(20)                    props.load(is);
(21)            } catch (IOException ioe) {
(22)                    System.out.println("Error loading properties");
(23)                    System.exit(1);
(24)            }
(25)            PersistenceManagerFactory pmf =
(26)                    JDOHelper.getPersistenceManagerFactory(props);
```

```
(27)            PersistenceManager pm = pmf.getPersistenceManager();
(28)
(29)            Extent ext = pm.getExtent(Employee.class, false);
(30)            String filter = "department == \"B042\"";
(31)            Query qry = pm.newQuery(ext, filter);
(32)            qry.setOrdering("name ascending");
(33)            qry.compile();
(34)            Collection c = (Collection) qry.execute();
(35)
(36)            Iterator i = c.iterator();
(37)            while (i.hasNext()) {
(38)                    System.out.println(i.next());
(39)            }
(40)    }
(41)
(42)}
```

Download des Beispiels

Das Beispiel illustriert eine Anfrage, die alle `Employee`-Objekte liefert, deren `department`-Attribut mit dem Wert `B042` belegt ist und liefert die nach dem Inhalt des Attributes `name` in aufsteigender Reihenfolge sortiert.
Hierzu wird zunächst die vollständige Extension der Klasse `Employee` ermittelt. Allerdings Extrahiert dieser Aufruf noch keine Werte aus dem persistenten Objektspeicher, sondern schafft nur die Grundlagen einer späteren manuellen Traversierung oder der Anfrage via JDOQL.
Zur Vorbereitung der tatsächlichen physischen Anfrage wird zunächst eine Zeichenkette geeignet belegt, um als Filterausdruck dienen zu können, der auf die vollständige Extension angewandt wird. Im Beispiel ist dieser Filterausdruck mit `department == \"B042\"` belegt. Aus Gründen der Zeichenkettenverarbeitung in Java muß hierzu der notwendige Einschluß des zu suchenden Wertes in Anführungszeichen geeignet maskiert werden.
Nach diesen Vorbereitungsschritten kann durch den Aufruf der durch das `PersistenceManager`-kompatible Objekt bereitgestellten Methode `newQuery` ein neues Anfrageobjekt (vom Typ `Query`) erzeugt werden.
Dieses Objekt erlaubt nach der gezeigten Festlegung des Anfrageumfanges die Parametrisierung der Anfrage. Das Beispiel illustriert dies am Aufruf der Methode `setOrdering`, die es erlaubt eine bestimmte Sortierreihenfolge der gelieferten Ergebnisse vorzugeben.
Zusätzlich kann durch die optionale Ausführung der Methode `compile` eine Prüfung der zusammengestellten Anfrage erfolgen, die zusätzlich auch interne implementierungsspezifische Optimierungen vornehmen kann.
Abschließend erfolgt die Ausführung der Anfrage durch Aufruf der Methode `execute`, welche die Anfrageergebnisse konform zur Standardschnittstelle `Collection` zurückliefert.

Löschen von Objekten

Zur Entfernung eines Objektes aus dem Objektspeicher stellt die Schnittstelle `PersistenceManager` die Methode `deletePersistent` zur Verfügung, welche ein einzelnes hauptspeicherresidentes Objekt aus dem persistenten Speicher löscht, bzw. mit `deletePersistentAll` eine Möglichkeit alle durch eine Sammlung referenzierten Objekte zu entfernen.
Da es sich hierbei um einen schreibenden Zugriff handelt, muß dieser in einen Transaktionskontext eingebettet werden oder explizit transaktionslos durchgeführt werden wie in Beispiel 83 gezeigt.
Beispiel 86 zeigt die Löschung unter Verwendung eines Transaktionskontextes.

Beispiel 86: Löschen eines Objektes aus dem persistenten Objektbestand

```
(1)import java.io.IOException;
(2)import java.io.InputStream;
(3)import java.util.Collection;
(4)import java.util.Properties;
(5)
(6)import javax.jdo.Extent;
(7)import javax.jdo.JDOHelper;
(8)import javax.jdo.PersistenceManager;
(9)import javax.jdo.PersistenceManagerFactory;
(10)import javax.jdo.Query;
(11)
(12)import de.jeckle.jdotest.Employee;
(13)
(14)public class JDODeleteObj {
(15)    public static void main(String args[]) {
(16)            Properties props = new Properties();
(17)            try {
(18)                    InputStream is =
(19)                            ClassLoader.getSystemResourceAsStream("jdo.properties");
```

```
(20)                    props.load(is);
(21)            } catch (IOException ioe) {
(22)                    System.out.println("Error loading properties");
(23)                    System.exit(1);
(24)            }
(25)            PersistenceManagerFactory pmf =
(26)                    JDOHelper.getPersistenceManagerFactory(props);
(27)            PersistenceManager pm = pmf.getPersistenceManager();
(28)
(29)            Extent ext = pm.getExtent(Employee.class, false);
(30)            String filter = "name == \"Marta Smith\"";
(31)            Query qry = pm.newQuery(ext, filter);
(32)            Collection c = (Collection) qry.execute();
(33)
(34)            pm.currentTransaction().begin();
(35)            pm.deletePersistentAll(c);
(36)            pm.currentTransaction().commit();
(37)            System.out.println("Object deleted");
(38)    }
(39)}
```

Download des Beispiels

Migration zu einem anderen Persistenzdienstleister

Der JDO-Ansatz tritt mit dem Versprechen auf vollständig sowohl unabhängig vom verwendeten Persistenzmedium (etwa: Datenbank, Dateisystem, etc.) als auch der eingesetzten JDO-Implementierung zu sein. Diese Zielsetzung wird nachfolgend auf Basis des im vorhergehenden diskutierten `Employee`-Beispiels untersucht. Hierzu wird die frei verfügbare JDO-Implementierung *TJDO* eingesetzt, welche verschiedene Datenbankmanagementsysteme zur Speicherung der Javaobjekte heranziehen kann. Im Beispiel wird das DBMS *MySQL* Persistierung der Applikationsobjekte genutzt.

Zur Portierung der bestehenden Applikation ist lediglich die Anpassung der JDO-Eigenschaften (Property-Datei) vorzunehmen, um den neuen Persistenzdienstleister sowie die verschiedenen DBMS-Spezifika zu berücksichtigen.
Beispiel 87 zeigt die neuen Inhalte.

Beispiel 87: Konfiguration der JDO-Implementierung TJDO

```
(1)javax.jdo.PersistenceManagerFactoryClass=com.triactive.jdo.PersistenceManagerFactoryImpl
(2)javax.jdo.option.ConnectionURL=jdbc:mysql://localhost/jdotest/
(3)javax.jdo.option.ConnectionDriverName=com.mysql.jdbc.Driver
(4)javax.jdo.option.ConnectionUserName=mario
(5)javax.jdo.option.ConnectionPassword=thePassword
(6)com.triactive.jdo.autoCreateTables=true
```

Download des Beispiels

Zunächst werden die bereits in der Konfiguration der Referenzimplementierung durch Beispiel 76 genutzten Eigenschaften zur Identifikation derjenigen Klasse, welche die JDO-Schnittstelle `PersistenceManagerFactory` implementiert sowie zur Festlegung der Verbindungs-URL und des zu verwendenden Benutzernamens uns Passwortes an die neuen Gegebenheiten adaptiert. Konkret wird die durch TJDO bereitgestellte Klasse `com.triactive.jdo.PersistenceManagerFactoryImpl` als `PersistenceManagerFactory` konforme Implementierung sowie die Identifikation der zu verwendenden Datenbank nebst Benutzername und Anmeldekennwort bekanntgegeben.
Zusätzlich wird mit `com.triactive.jdo.autoCreateTables` eine implementierungsspezifische Eigenschaft mit `true` belegt, die TJDO veranlaßt im Bedarfsfalle benötigte Tabellenstrukturen automatisiert zu erzeugen.

Zusätzlich erfordert die verwendete JDO-Implementierung die Adaption der im Rahmen des Bytecodeanreicherungsprozesses herangezogenen Konfigurationsdatei (Beispiel 88). Auf diesem Wege wird dem Programmierer die Möglichkeit eröffnet die Abbildung auf relationale Tabellenstrukturen beeinflussen. In der Konsequenz erfordert der Wechsel der JDO-Implementierung die Wiederholung des Anreicherungslaufes für den Bytecode der zu persistierenden Klassen.

Beispiel 88: Parametrisierung der Objektpersistenz

```
(1)<?xml version="1.0"?>
(2)  <!DOCTYPE jdo PUBLIC "-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 1.0//
EN" "http://java.sun.com/dtd/jdo_1_0.dtd">
(3)  <jdo>
(4)      <package name="de.jeckle.jdotest">
(5)          <class name="Employee">
(6)              <field name="name">
(7)                  <extension vendor-name="triactive" key="length" value="max 32"/>
(8)              </field>
(9)              <field name="department">
(10)                  <extension vendor-name="triactive" key="length" value="max 32"/>
(11)              </field>
(12)          </class>
(13)      </package>
(14)  </jdo>
```

Download des Beispiels

Weitere Änderungen an den zu persistierenden Klassen oder den mit deren Objekten operierenden Applikationen ist nicht notwendig, alle Zugriffe werden nach den oben beschriebenen Änderungen transparent und ohne Neuübersetzung datenbankbasiert abgewickelt.

Vergleich der verschiedenen Persistenzansätze

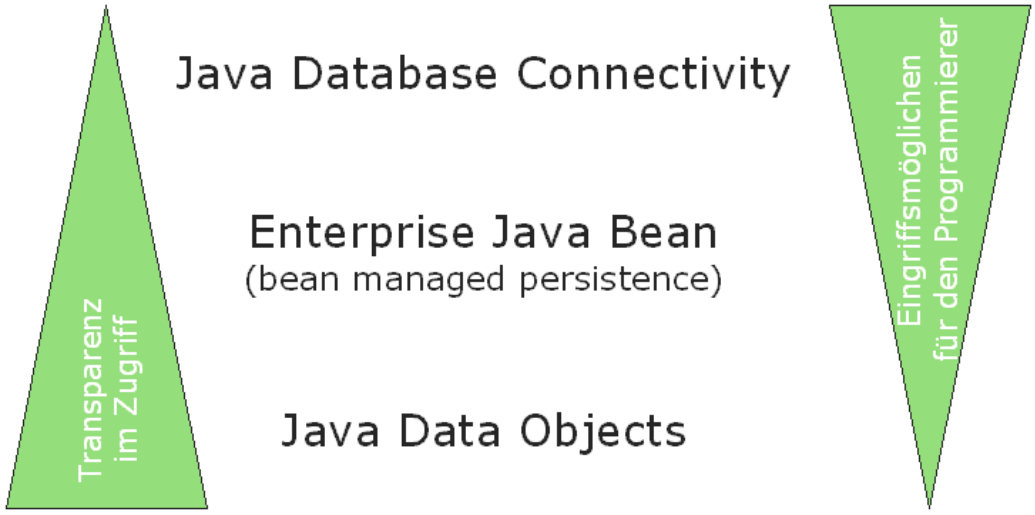
Abschließend seien die charakteristischen Eigenschaften der drei diskutierten Persistenzansätze JDBC, EJB und JDO kurz vergleichend nebeneinandergestellt.

Merkmal	JDBC	EJB	JDO
Transaktionsunterstützung	✓	✓	✓
Anfragemöglichkeit	✓	✓	✓
Standardisiertes API	✓	✓	✓
Standardanfragesprache	✓ SQL	✓ SQL/EJBQL	✓ JDOQL
Unterstützte Hintergrundspeicher	RDBMS	RDMBS Integrationsmiddleware	RDBMS, ORDBMS Integrationsmiddleware, Dateisystem, bel. andere
Transparenter Zugriff auf persistierte Daten	⊘	✓	✓
Berücksichtigung existierender relationaler Strukturen	✓	✓ bei bean managed persistence	✓ allerdings nicht im Standard vorgesehen

Die Tabelle zeigt klar, daß alle drei Persistenzmechanismen grundlegende Eigenschaften teilen, sich jedoch auch in zentralen Charakteristika unterscheiden.
Während sowohl JDBC als auch EJBs die direkte Verwendung von SQL-Anfragen gestatten bietet JDO mit JDOQL eine eigenständige Anfragesprache, die direkt in die Sprach-API eingebettet ist. Für EJBs existiert neben den in Kapitel 1.2 gezeigten Mechanismen auch die Möglichkeit der Verwendung der EJB-spezifischen Anfragesprache *EJBQL*, die jedoch hier nicht betrachtet wurde.
Hinsichtlich der jeweils unterstützten Hintergrundspeicherarchitekturen zur Realisierung der Persistenz treten jedoch deutliche Unterschiede zu Tage. So ist der Einsatz der JDBC-API auf relationale Datenquellen, bzw. Datenquellen die eine relationale Sicht anbieten, beschränkt. Innerhalb der EJB-Architektur können hingegen neben den -- hier diskutierten JDBC-basierten Mechanismen -- auch die Dienste einer Integrationsmiddleware zu Speicherung herangezogen werden und so eine gewisse Unabhängigkeit vom physischen Speichermedium erreicht werden. Einzig JDO bietet durch seine starke Abstraktion die Möglichkeit beliebige Persistenzdienstleister zu nutzen.
Zur effizienten Abwicklung dieses speicherformunabhängigen Zugriffs etabliert JDO notwendigerweise eine stark abstrahierte API, deren Funktionen keinerlei Rückschlüsse auf den verwendeten Persistenzmechanismus zulassen. Für EJB läßt sich dies prinzipiell auch realisieren, allerdings müssen für die Variante der bean managed persistence innerhalb der Entity Bean die Interaktionen mit dem Persistenzdienstleister expliziert werden, beispielsweise durch JDBC. Daher verhält sich dieser Ansatz intern ähnlich zur direkten Verwendung der JDBC-API, die inhärent jeden angebundenen

Persistenzmechanismus mit relationaler Zugriffssemantik belegt.
Aufgrund des vorherrschenden relationalen Speicherparadigmas kann die Einbindung bestehender Tabellenstrukturen in den API-Mechanismus gewünscht sein. Dies ist ausschließlich mit Ansätzen möglich, welche die anwenderdefinierte Strukturierung der Zugriffsausdrücke -- etwa durch die Verwendung von SQL -- gestatten. Dies ist ausschließlich für JDBC und EJB (sofern bean managed persistence verwendet wird) möglich; JDO sieht dies generell nicht vor.

Abbildung 10: *Vergleich zwischen den diskutierten Persistenztechniken*



(click on image to enlarge!)

Abschließend lassen sich die vorgestellten Schnittstellen hinsichtlich ihrer Möglichkeiten zur Bereitstellung eines transparenten Zugriffs auf den Hintergrundspeicher und der manuellen Eingriffsmöglichkeiten zur Kontrolle der Persistenz durch den Programmierer kategorisieren.
Prinzipiell läßt sich festhalten, daß diese Eigenschaftstypen konkurrierende Zielsetzungen darstellen. So bietet JDBC zweifelsohne die größten Möglichkeiten zum steuernden Eingriff durch den Programmierer, wobei dieser Ansatz in der Interaktion auch die größte Menge Wissen des Programmierers über die etablierten Speicherstrukturen erfordert. Daher realisiert JDBC generell die geringste Transparenz im Zugriff auf den Objektspeicher.
Auf der anderen Seite realisiert JDO die größtmögliche Transparenz im Objektzugriff, wobei dieser Freiheitsgrad zu generell zu Lasten der Eingriffsmöglichkeiten durch den Programmierer umgesetzt werden.

Web-Referenzen 10: Weiterführende Links
- TJDO -- eine freie JDO-Implementierung
- JDO @ SUN
- JDOCentral.com -- Die Anlaufstelle der JDO-Entwickler
- JDO-Spezifikation

---

4 Funktionsintegration

4.1 Java Message Service (JMS)

Einführung und Motivation

Der Austausch von Daten zwischen Rechnersystem ist so alt wie die Vernetzung zunächst separierter Rechnersysteme selbst. Während die Abwicklung des Austausches von Nachrichten zwischen den vernetzen Systemen anfänglich individuell für jede Plattform, d.h. spezifisch für jede Programmiersprache, das zugrundeliegende Betriebssystem und die zur physischen Datenübertragung eingesetzte Netzwerkinfrastruktur, gelöst werden mußte bildeten sich zur Steigerung der Interoperabilität und Portabilität bei gleichzeitiger Komplexitätsreduktion im Laufe der Zeit eigenständige generische Kommunikationskomonenten heraus, die zum Versand beliebiger Nachrichten eingesetzt werden können.

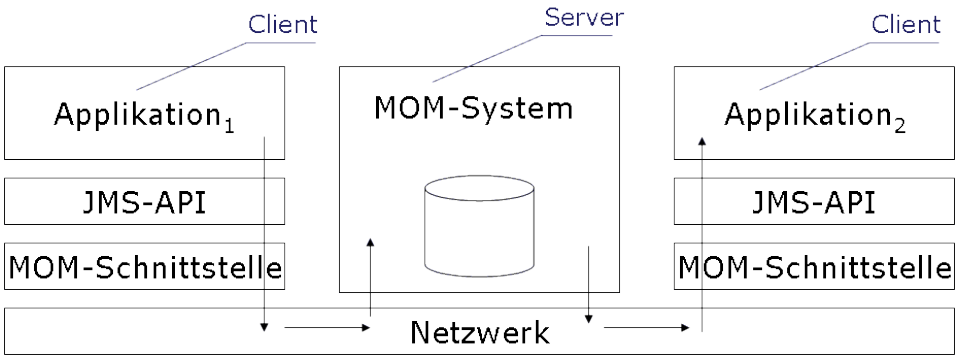
Ziel dieser -- als *message-oriented Middleware* bezeichneten -- Softwareinfrastruktur ist die deutliche Komplexitätsreduktion bei der Erstellung vernetzt kommunizierender Applikationen durch Einführung einer standardisierten, plattformübergreifend verfügbaren Lösung, welche dem Applikationsprogrammierer eine abstrahierte und vergleichsweise simple Programmierschnittstelle anbietet.

Zentrales Architekturmerkmal nachrichtenorientierter Middlewarekomponenten ist die gleichzeitige Betrachtung von ausschließlich zwei kommunizierenden Partnern, die als *client* und *server* bezeichnet werden, die in asynchroner Weise miteinander Nachrichten austauschen.

Im Rahmen der abgewickelten Interaktion versendet der Client dabei beliebige Daten an den Server, oder empängt diese von ihm. Der Server ist dabei die Message-orientierte Middleware, welche Sender und tatsächlichen Empfänger physische voneinander entkoppelt.
Die Begriffsbildung *asynchron* bezeichnet hierbei einen Kommunikationsstil, der die beteiligten Kommunikationspartner weder zeitlich noch prozedural koppelt.

Das Verhältnis zwischen Clients und Server, sowie das Zusammenspiel der verschiedenen physischen Architekturelemente ist in Abbildung 11 dargestellt.
Die Abbildung zeigt JMS-API, welche seitens der beteiligten Clients den Zugriff auf die systemspezifische Schnittstelle der verwendeten Middlware-Implementierung kapselt. Durch Pfeile ist der Ablauf eines Nachrichtenversandes von $\texttt{Applikation}_1$ an $\texttt{Applikation}_2$ dargestellt. Zunächst wird die Nachricht, durch Nutzung der involvierten Schnittstellen, an das MOM-System übertragen, welches es anschließend an den durch $\texttt{Applikation}_2$ vorgehaltenen Client ausliefert bzw. diesem zur Abholung bereitstellt.
Entlang des Pfades, welchen eine versandte Nachricht zurücklegt können auch mehrere MOM-System hintereinandergeschaltet auftreten. Diese agieren dann als Client bezüglich der in der Nachrichtenkette angrenzenden MOM-Systeme.

Abbildung 11: *Physische Architektur Nachrichten-orientierter Middleware*



(click on image to enlarge!)

Zusätzlich zeigt die Abbildung die typischerweise in MOM-Systemen präsente Datenbankkomponente, welche zur Sicherstellung der verläßlichen Nachrichtenübertragung dient. Im Normalfall wird dem versendenden Client erst die korrekte Übernahme der Nachricht in das MOM-System signalisiert, wenn sie persistent in der Datenbank abgelegt wurde. Im selben Sinne wird eine datenbankresidente Nachricht erst dann dauerhaft gelöscht, wenn sie dem Zielclient korrekt übermittelt wurde.
Durch diese Mimik entsteht, auch im Falle des Hintereinanderschaltens einzelner MOM-Systeme, eine verlässliche Nachrichtenstrecke, welche den Versand der Nachricht auch im Falle des Ausfalls einzelner Streckenabschnitte (d.h. MOM-Systeme) sicherstellen kann.

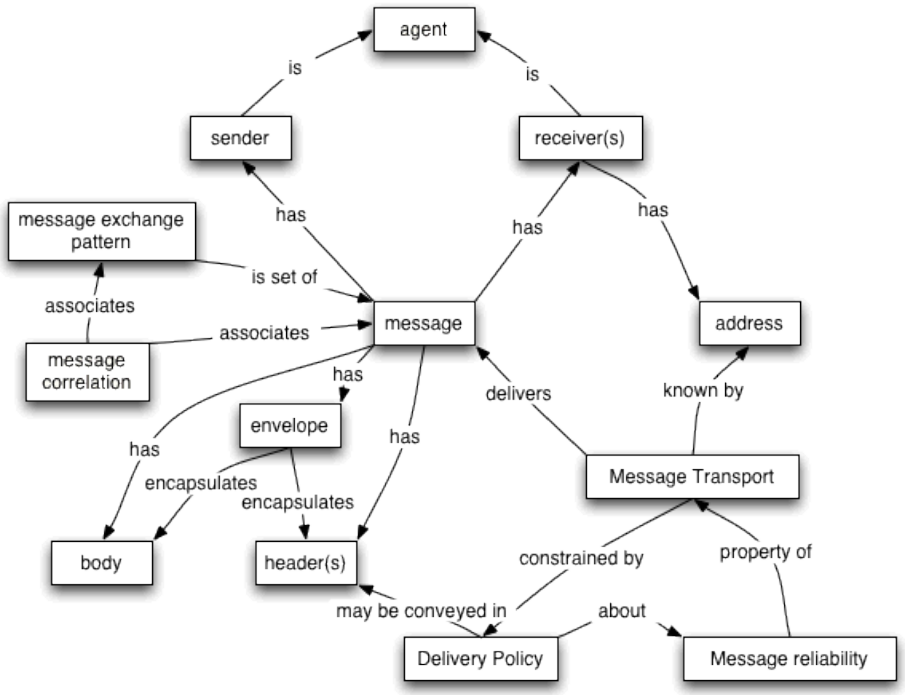
Abbildung Abbildung 12 zeigt, als Ausschnitt der Untersuchungen des W3C zu Web-Service-Architekturen, eine aktuelle Bestandsaufnahme der Komponenten einer Nachrichten-orientierten Architektur:

Tabelle 25: Elemente einer Nachrichten-orientierten-Middleware-Architektur

Architekturkomponente	Beschreibung
Address	Zielpunkt an den der Nachrichtentransportmechanismus die Nachricht ausliefert. Der syntaktische Aufbau einer Zieladresse kann vom verwendeten Transportprotokoll abhängen.
Delivery Policy	Art der Abwicklung des Nachrichtenversandes. Hierunter fallen qualitative Aspekte wie einzuhaltende Sicherheitsrestriktionen sowie die garantierte Auslieferung der Nachricht bis zu einem vorbestimmten Zeitpunkt.
Message	Logisches Konstrukt, das die auszuliefernde Nachricht repräsentiert. Sie besteht aus dem Nutzdateninhalt (Message Body) und optionalen beschreibenden Metadaten (Message Header).
Message Body	Der durch den Anwender festlegbare Nutzdateninhalt einer Nachricht.

Message Correlation	Einordnung einer Nachricht in einen Kontext. Korrelationsmechanismen stellen sicher, das zustandsbehaftete Protokolle -- die mehr als genau einer Kommunikation bedürfen -- über zustandslose Transportmechanismen korrekt abgewickelt werden können.
Message Envelope	Physische Implementierung der logischen Nachricht, daß Nachrichtenköpfe und Nutzdaten beinhaltet.
Message Header	Eine Datenstruktur, die deskriptive Metainformation über Teile einer Nachricht beinhaltet.
Message Exchange Pattern	Stil der Kommunikationsabwicklung hinsichtlich temoraler und logischer Abhängigkeiten. Synchrone und asynchrone Kommunikation sind zwei Beispiele konkreter Nachrichtenaustauschmuster. In gewissem Sinne stellen Nachrichtenaustauschmuster eine leichtgewichtige Korrelationsform dar.
Message Receiver	Agent, der die Nachricht entgegenimmt.
Message Reliability	Grad der Verläßlichkeit, daß eine Nachricht sicher zugestellt wird. Zwischen Sender und Empfänger muß diesbezüglich eine Übereinkunft hinsichtlich der Operationalität des Begriffs bestehen.
Message Sender	Agent, der in der Rolle als Client die Nachricht versendet.
Message Sequence	Geordnete Menge einzelner Nachrichten.
Message Transport	Technischer Mechanismus, der durch Agenten zum physischen Nachrichtenversand genutzt wird.

Abbildung 12: *Logische Architekturkomponenten einer Nachrichten-orientierter Middleware*



(click on image to enlarge!)

Innerhalb der Java-J2EE-API bietet die Programmierschnittstelle *Java Message Service* (JMS) einfachen Zugriff auf die Prinzipien Nachrichten-orientierter Kommunikation, die durch beliebige Implementierungen

Nachrichten-orientierter-Middleware-Systeme zur Verfügung gestellt werden können.
Für die nachfolgenden Codebeispiele wurde die Implementierung der *Sun Java Message Queue*, welche als Bestandteil der J2EE-Referenzimplementierung ausgeliefert wird, genutzt. Die Beispiele sollten sich jedoch mit geringem Anpassungsaufwand auch auf andere Middlwaresysteme übertragen lassen.

Äquivalenz von synchroner und asynchroner Kommuniktion

Die JMS-API

Die JMS-API besteht aus einer Reihe von Java-Schnittstellenspezifikationen, die durch das verwendete Middlewaresystem implementiert werden.

Nachfolgend ist eine Übersicht der zentralen Schnittstellen und der durch sie angebotenen Funktionalität gegeben.

Tabelle 26: Schnittstellen der JMS-API

Schnittstelle	Beschreibung
`Connection`	Repräsentiert die Verbindung eines Clients zu einem Server über die Nachrichten versandt werden können.
`ConnectionFactory`	Kapselt verschiedene Konfigurationseinstellungen und wird clientseitig zur Erzeugung eines `Connection`-Objekts eingesetzt.
`Destination`	Dient der Repräsentation einer Transportmechanismen-spezifischen Adresse.
`MessageProducer`	Stellt dem Client die zum Nachrichtenversand nötigen Operationen zur Verfügung.
`MessageConsumer`	Stellt dem Client die zum Nachrichtenempfang benötigten Operationen zur Verfügung.>
`Queue`	Repräsentiert eine Nachrichtenstrecke.
`Session`	Liefert die logische Umgebung zum Versand und Empfang von Nachrichten.
`TextMessage`	Die zu versendene Nachricht. JMS gestatt spezifikationsgemäß ausschließlich die Übermittlung von Nachrichten, die durch Objekte der Standard-Klasse `String` repräsentiert werden.
`Topic`	Repräsentiert eine semantische Untermenge von Nachrichten in Form eines logischen Kanals.
`TopicConnection`	Repräsentiert einen logischen Kanal der durch Abonennten subscribiert werden und an alle abonnierenden Clients spontan Nachrichten versenden kann.
`TopicConnectionFactory`	Kapselt verschiedene Konfigurationseinstellungen und wird zur Erzeugung eines `TopicConnection`-Objekts eingesetzt.
`Destination`	Dient der Repräsentation einer Transportmechanismen-spezifischen Adresse.
`MessageProducer`	Stellt dem Client die zum Nachrichtenversand nötigen Operationen zur Verfügung.

MessageConsumer	Stellt dem Client die zum Nachrichtenempfang benötigten Operationen zur Verfügung.
TopicPublisher	Durch den Client eingesetzte Schnittstelle, die zum Versand von Nachrichten an abonnierte Kanäle dient.
TopicSubscriber	Durch den Client eingesetzte Schnittstelle, die zum Abruf von Nachrichten dient, welche an abonnierte Kanäle versandt wurden.
TopicSession	Schnittstelle, die Operationen zur Erzeugung von TopicPublisher-, TopicSubscriber- und TemporaryTopic-Objekten bereitstellt.

Synchroner Nachrichtenverkehr

TODO(Hier fehlt noch eine textuelle Erklärung)

Beispiel 89: Versand einer Nachricht

```
(1)import java.util.Date;
(2)import javax.jms.Connection;
(3)import javax.jms.ConnectionFactory;
(4)import javax.jms.Destination;
(5)import javax.jms.JMSException;
(6)import javax.jms.MessageProducer;
(7)import javax.jms.Queue;
(8)import javax.jms.Session;
(9)import javax.jms.TextMessage;
(10)import javax.jms.Topic;
(11)import javax.naming.Context;
(12)import javax.naming.InitialContext;
(13)import javax.naming.NamingException;
(14)public class SProducer {
(15)    public static void main(String[] args) {
(16)            final int NUM_MSGS;
(17)            if ((args.length < 2) || (args.length > 3)) {
(18)                    System.out.println("Program takes two or three arguments: "+
"<dest_name> <queue|topic> " + "[<number-of-messages>");
(19)                    System.exit(1);
(20)            }
(21)            String destName = new String(args[0]);
(22)            String destType = new String(args[1]);
(23)            System.out.println("Destination name is " + destName + ", type is " +
destType);
(24)            if (args.length == 3) {
(25)                    NUM_MSGS = (new Integer(args[2])).intValue();
(26)            } else {
(27)                    NUM_MSGS = 1;
(28)            }
(29)            Context jndiContext = null;
(30)            try {
(31)                    jndiContext = new InitialContext();
(32)            } catch (NamingException e) {
(33)                    System.out.println("Could not create JNDI " + "context: "
(34)                                    + e.toString());
(35)                    System.exit(1);
(36)            }
(37)            ConnectionFactory connectionFactory = null;
(38)            Destination dest = null;
(39)            try {
(40)                    connectionFactory = (ConnectionFactory) jndiContext.lookup("jms/
QueueConnectionFactory");
(41)                    if (destType.equals("queue")) {
(42)                            dest = (Queue) jndiContext.lookup(destName);
(43)                    } else if (destType.equals("topic")) {
(44)                            dest = (Topic) jndiContext.lookup(destName);
(45)                    } else {
(46)                            throw new Exception("Invalid destination type" + "; must
be queue or topic");
(47)                    }
```

```
(48)            } catch (Exception e) {
(49)                    System.out.println("JNDI lookup failed: " + e.toString());
(50)                    e.printStackTrace();
(51)                    System.exit(1);
(52)            }
(53)            Connection connection = null;
(54)            MessageProducer producer = null;
(55)            try {
(56)                    connection = connectionFactory.createConnection();
(57)                    Session session = connection.createSession(false,
(58)                                    Session.AUTO_ACKNOWLEDGE);
(59)                    producer = session.createProducer(dest);
(60)                    TextMessage message = session.createTextMessage();
(61)                    for (int i = 0; i < NUM_MSGS; i++) {
(62)                            message.setText("This is message " + (i + 1) + " sent at "
(63)                                            + new Date());
(64)                            System.out.println("Sending message: " + message.getText
()));
(65)                            producer.send(message);
(66)                    }
(67)                    producer.send(session.createMessage());
(68)            } catch (JMSException e) {
(69)                    System.out.println("Exception occurred: " + e.toString());
(70)            } finally {
(71)                    if (connection != null) {
(72)                            try {
(73)                                    connection.close();
(74)                            } catch (JMSException e) {
(75)                                    System.err.println("JMSException occured while
closing connection");
(76)                            }
(77)                    }
(78)            }
(79)    }
(80)}
```

Download des Beispiels
Download der Ergebnisdatei

Aufruf mit: `j2ee14/bin/appclient -client SProducer.jar jms/Queue queue 5`

TODO(Hier fehlt noch eine textuelle Erklärung)

Beispiel 90: Empfang einer Nachricht

```
(1)import javax.jms.Connection;
(2)import javax.jms.ConnectionFactory;
(3)import javax.jms.Destination;
(4)import javax.jms.JMSException;
(5)import javax.jms.Message;
(6)import javax.jms.MessageConsumer;
(7)import javax.jms.Queue;
(8)import javax.jms.Session;
(9)import javax.jms.TextMessage;
(10)import javax.jms.Topic;
(11)import javax.naming.Context;
(12)import javax.naming.InitialContext;
(13)import javax.naming.NamingException;
(14)public class SConsumer {
(15)    public static void main(String[] args) {
(16)            String destName = null;
(17)            String destType = null;
(18)            Context jndiContext = null;
(19)            ConnectionFactory connectionFactory = null;
(20)            Connection connection = null;
(21)            Session session = null;
(22)            Destination dest = null;
(23)            MessageConsumer consumer = null;
(24)            TextMessage message = null;
(25)            if (args.length != 2) {
(26)                    System.out.println("Program takes two arguments: " + "<dest_name>
<queue|topic>");
(27)                    System.exit(1);
(28)            }
(29)            destName = new String(args[0]);
(30)            destType = new String(args[1]);
```

```
(31)            System.out.println("Destination name is " + destName + ", type is "
(32)                            + destType);
(33)            try {
(34)                    jndiContext = new InitialContext();
(35)            } catch (NamingException e) {
(36)                    System.out.println("Could not create JNDI " + "context: "
(37)                                    + e.toString());
(38)                    System.exit(1);
(39)            }
(40)            try {
(41)                    connectionFactory = (ConnectionFactory) jndiContext
(42)                                    .lookup("jms/QueueConnectionFactory");
(43)                    if (destType.equals("queue")) {
(44)                            dest = (Queue) jndiContext.lookup(destName);
(45)                    } else if (destType.equals("topic")) {
(46)                            dest = (Topic) jndiContext.lookup(destName);
(47)                    } else {
(48)                            throw new Exception("Invalid destination type" + "; must
be queue or topic");
(49)                    }
(50)            } catch (Exception e) {
(51)                    System.out.println("JNDI lookup failed: " + e.toString());
(52)                    System.exit(1);
(53)            }
(54)            try {
(55)                    connection = connectionFactory.createConnection();
(56)                    session = connection.createSession(false, Session.
AUTO_ACKNOWLEDGE);
(57)                    consumer = session.createConsumer(dest);
(58)                    connection.start();
(59)                    Message m;
(60)                    while (true) {
(61)                            m = consumer.receive(1);
(62)                            if (m != null) {
(63)                                    if (m instanceof TextMessage) {
(64)                                            message = (TextMessage) m;
(65)                                            System.out.println("Reading message: " +
message.getText());
(66)                                    } else
(67)                                            break;
(68)                            }
(69)                    }
(70)            } catch (JMSException e) {
(71)                    System.out.println("Exception occurred: " + e.toString());
(72)            } finally {
(73)                    if (connection != null) {
(74)                            try {
(75)                                    connection.close();
(76)                            } catch (JMSException e) {
(77)                                    System.err.println("JMSException occured while
closing connection");
(78)                            }
(79)                    }
(80)            }
(81)    }
(82)}
```

Download des Beispiels
Download der Ergebnisdatei

Aufruf mit: `j2ee14/bin/appclient -client SProducer.jar jms/Queue queue 5`

Asynchroner Nachrichtenverkehr

TODO(Hier fehlt noch eine textuelle Erklärung)

Aufruf mit: `j2ee14/bin/appclient -client PSProducer.jar`

Beispiel 91: Versand einer Nachricht

```
(1)import javax.jms.DeliveryMode;
(2)import javax.jms.JMSException;
(3)import javax.jms.Message;
(4)import javax.jms.Session;
(5)import javax.jms.TextMessage;
(6)import javax.jms.Topic;
(7)import javax.jms.TopicConnection;
(8)import javax.jms.TopicConnectionFactory;
(9)import javax.jms.TopicPublisher;
(10)import javax.jms.TopicSession;
(11)import javax.naming.Context;
(12)import javax.naming.InitialContext;
(13)import javax.naming.NamingException;
(14)
(15)public class PSProducer {
(16)    public static void main(String[] args) {
(17)            Topic t = null;
(18)            TopicConnectionFactory tcf = null;
(19)            TopicConnection tc = null;
(20)
(21)            try {
(22)                    Context jndiContext = new InitialContext();
(23)                    tcf = (TopicConnectionFactory) jndiContext.lookup("jms/
TopicConnectionFactory");
(24)                    //t = (Topic) jndiContext.lookup("StockQuote");
(25)            } catch (NamingException ne) {
(26)                    System.out.println(ne+ " does not exist");
(27)                    System.exit(1);
(28)            }
(29)
(30)            try {
(31)                    tc = tcf.createTopicConnection();
(32)                    TopicSession ts = tc.createTopicSession(false, Session.
AUTO_ACKNOWLEDGE);
(33)                    t = ts.createTopic("StockQuote");
(34)
(35)                    TopicPublisher tp = ts.createPublisher(t);
(36)
(37)                    TextMessage msg1 = ts.createTextMessage();
(38)                    TextMessage msg2 = ts.createTextMessage();
(39)
(40)                    msg1.setText("Quote");
(41)                    msg1.setStringProperty("Company","DCX");
(42)                    msg1.setDoubleProperty("USD",41.67);
(43)                    msg1.setDoubleProperty("EUR",34.99);
(44)                    msg1.setStringProperty("Date","2004-05-23");
(45)
(46)                    tp.publish(msg1, DeliveryMode.PERSISTENT, Message.
DEFAULT_PRIORITY, 7*24*3600*1000L);
(47)
(48)                    try {
(49)                            Thread.sleep(30000);
(50)                    } catch(InterruptedException ie) {
(51)                            System.err.println("InterruptedException caught");
(52)                    }
(53)
(54)                    msg2.setText("Quote");
(55)                    msg2.setStringProperty("Company","AZ");
(56)                    msg2.setDoubleProperty("USD",10.04);
(57)                    msg2.setDoubleProperty("EUR",9.30);
(58)                    msg2.setStringProperty("Date","2004-05-23");
(59)
(60)                    tp.publish(msg2, DeliveryMode.PERSISTENT, Message.
DEFAULT_PRIORITY, 7*24*3600*1000L);
(61)            } catch (JMSException e) {
(62)                    System.err.println("JMSException caught");
(63)                    e.printStackTrace();
(64)            }
(65)    }
(66)}
```

Download des Beispiels

TODO(Hier fehlt noch eine textuelle Erklärung)

Aufruf mit: `j2ee14/bin/appclient -client PSConsumer.jar`

Beispiel 92: Empfang einer Nachricht

```
(1)import java.util.Date;
(2)
(3)import javax.jms.JMSException;
(4)import javax.jms.Message;
(5)import javax.jms.MessageListener;
(6)import javax.jms.Session;
(7)import javax.jms.TextMessage;
(8)import javax.jms.Topic;
(9)import javax.jms.TopicConnection;
(10)import javax.jms.TopicConnectionFactory;
(11)import javax.jms.TopicSession;
(12)import javax.jms.TopicSubscriber;
(13)import javax.naming.Context;
(14)import javax.naming.InitialContext;
(15)import javax.naming.NamingException;
(16)public class PSConsumer {
(17)    public static void main(String[] args) {
(18)            Topic t = null;
(19)            TopicConnectionFactory tcf = null;
(20)            TopicConnection tc = null;
(21)
(22)            try {
(23)                    Context jndiContext = new InitialContext();
(24)                    tcf = (TopicConnectionFactory) jndiContext.lookup("jms/
TopicConnectionFactory");
(25)                    //t = (Topic) jndiContext.lookup("StockQuote");
(26)            } catch (NamingException ne) {
(27)                    System.err.println(ne+" does not exist");
(28)                    System.exit(1);
(29)            }
(30)
(31)            try {
(32)                    tc = tcf.createTopicConnection();
(33)                    tc.setClientID("MariosClient");
(34)                    TopicSession ts = tc.createTopicSession(false, Session.
AUTO_ACKNOWLEDGE);
(35)                    t = ts.createTopic("StockQuote");
(36)
(37)                    TopicSubscriber tSubDCX=ts.createDurableSubscriber(t,"mySub",
"Company LIKE '%DCX'",false);
(38)                    TopicSubscriber tSubAll=ts.createSubscriber(t);
(39)
(40)                    tSubDCX.setMessageListener( new MyTopicListener("DCX") );
(41)                    tSubAll.setMessageListener( new MyTopicListener("All"));
(42)
(43)                    tc.start();
(44)
(45)                    for (;;){
(46)                            System.out.println("("+new Date()+") waiting for
messages...");
(47)                            try {
(48)                                    Thread.sleep(1000);
(49)                            } catch(InterruptedException ie) {
(50)                                    System.err.println("InterruptedException caught");
(51)                            }
(52)                    }
(53)
(54)            } catch (JMSException e) {
(55)                    System.err.println("JMSException caught");
(56)                    e.printStackTrace();
(57)            }
(58)    }
(59)}
(60)
(61)class MyTopicListener implements MessageListener {
(62)    private String title;
(63)
(64)    public MyTopicListener(String title){
(65)            this.title = title;
(66)            System.out.println("Message listener "+title+" created");
(67)    }
(68)
```

```
(69)    public void onMessage(Message msg) {
(70)            try {
(71)            TextMessage tm = (TextMessage) msg;
(72)            System.out.println("("+this.title+") Message received: "+tm.getText());
(73)            System.out.println("Company="+tm.getStringProperty("Company"));
(74)            System.out.println("USD="+tm.getDoubleProperty("USD"));
(75)            System.out.println("EUR="+tm.getStringProperty("EUR"));
(76)            System.out.println("--------------------------");
(77)            } catch (JMSException je) {
(78)                    System.err.println("JMSException caught");
(79)                    je.printStackTrace();
(80)            }
(81)    }
(82)
(83)}
```

Download des Beispiels

Message-driven Beans

Die ursprünglich eigenständig entwickelte JMS-API ist inzwischen in die *Java 2 Enterprise Architecture* integriert worden. Implementierungen dieser Architektur können daher als MOM-System eingesetzt werden.

Darüber hinaus bildet die JMS-Funktionalität der Basis der *Message-driven Beans*. Diese Spielart der Entity Beans kann durch Empfang einer per JMS versandten Nachricht angesprochen werden. Voraussetzung hierzu ist die Implementierung der Operation `onMessage` der Schnittstelle `MessageListener` sowie die Implementierung der Schnittstelle `MessageDrivenBean`.

Das nachfolgende Beipiel zeigt die Reforumulierung des Beispiels 92 als Message-driven Bean.

Beispiel 93: Empfang einer Nachricht durch eine Message-driven Bean

```
(1)import javax.ejb.CreateException;
(2)import javax.ejb.MessageDrivenBean;
(3)import javax.ejb.MessageDrivenContext;
(4)import javax.jms.JMSException;
(5)import javax.jms.Message;
(6)import javax.jms.MessageListener;
(7)import javax.jms.TextMessage;
(8)
(9)public class MDBeanExample implements MessageDrivenBean, MessageListener{
(10)    private MessageDrivenContext ctx;
(11)    private TextMessage tm;
(12)
(13)    public void setMessageDrivenContext(MessageDrivenContext ctx) {
(14)            this.ctx = ctx;
(15)    }
(16)
(17)    public void ejbCreate() throws CreateException {
(18)            System.err.println("Message-driven Bean created");
(19)    }
(20)
(21)    public void ejbRemove() {
(22)            //does nothing
(23)    }
(24)
(25)    public void onMessage(Message msg)  {
(26)            if (msg instanceof TextMessage) {
(27)                    this.tm = (TextMessage) msg;
(28)                    try {
(29)                            System.out.println("Message received: "+this.tm.getText());
(30)                            System.out.println("Company="+this.tm.getStringProperty
("Company"));
(31)                            System.out.println("USD="+this.tm.getDoubleProperty
("USD"));
(32)                            System.out.println("EUR="+this.tm.getStringProperty
("EUR"));
(33)                            System.out.println("--------------------------");
(34)                    } catch (JMSException je) {
(35)                            System.err.println("JMSException caught");
(36)                            je.printStackTrace();
(37)                    }
(38)            }
```

```
(39)    }
(40)}
```

Download des Beispiels

Web-Referenzen 11: Weiterführende Links
- Introduction to Message-oriented Middleware
- JMS FAQ

4.2 Remote Method Invocation (RMI)

Grundidee

Der Java-spezifische Ansatz der *Remote Method Invocation* (RMI) zielt darauf einen lokationstransparenten Aufruf innerhalb von (verteilten) Java-Anwendungen zu ermöglichen.
RMI stellt hierbei die Einzelobjekte einer Applikation in den Vordergrund und gestattete den Aufruf von Methoden in derselben Art und Weise unabhängig davon ob sich das Objekt auf der aufrufenden Maschine selbst befindet oder an einer entfernten Lokation zur Verfügung steht.
Damit stellt sich RMI in die Tradition des objektorientierten Verteilungsmechanismus CORBA, ohne allerdings dessen programmiersprachenunabhängige Zielsetzung zu verfolgen. Gleichzeitig wurde das von DCE übernommene Serialierungsformat für hauptspeicherresidente Inhalte auf die Objektübertragung mittels eines Netzprotkolls ausgedehnt.
Die Grundprimitive des Ansatzes ist daher der entfernte Methodenaufruf (engl. *Remote Procedure Call*, RPC) welcher Kraft der lokalisationstransparenten Abstraktion RMIs dem Programmierer dieselbe Aufrufschnittstelle für entfernte Methoden bietet, die bereits für die Interaktion mit lokalen Objekten zum Einsatz kommt.
Zusätzlich erweitert RMI bestimmte zunächst lokal angebotene Dienstleistungen, wie etwa automatische Speicherverwaltung (*Garbage Collection*) für die verteilte Anwendung mit der Zielsetzung die Verwendung entfernter Objekte der lokal vorliegender gleichzustellen (Lokalisationstransparenz).
Zusammenfassend formuliert die RMI-Spezifikation daher folgende Ziele:

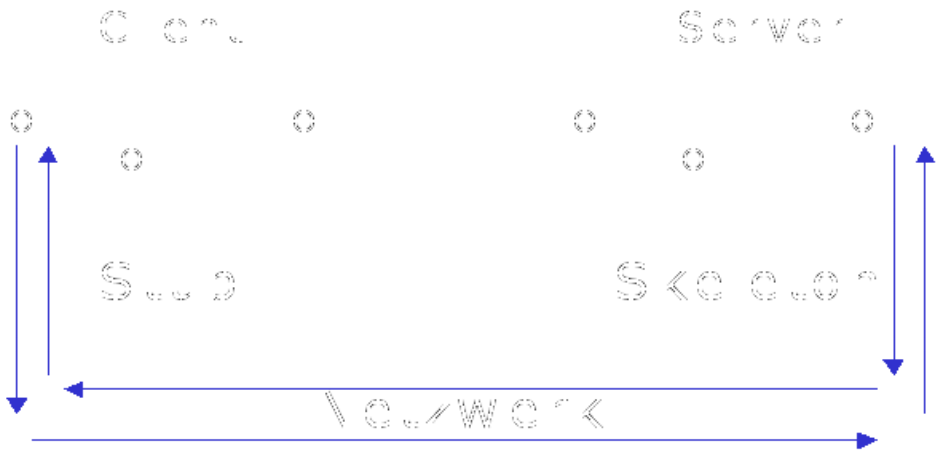
- Transparenter Methodenaufruf von Objekten die durch verschiedene virtuelle Maschinen verwaltet werden.
- Integration des Verteilungsmodells in die Programmiersprache unter weitestgehendem Erhalt der Semantik.
- Transparente Anlage des verteilten Objektmodells.
- Schaffung eines einfachstmöglichen Mechanismus zur Erstellung verteilter Applikationen.
- Erhalt der durch die Laufzeitumgebung geschaffenen Typsicherheit.
- Unterstützung verschiedener Referenzierungssemantiken.
- Erhalt der Java-Sicherheitsumgebung auch für verteilt ablaufende Applikationen.
- Unterstützung einer Call-back-Schnittstelle für Applets.

Technische Grundkonzepte

Integrale Bestandteile des RMI-Ansatzes und direkte Folgerungen der angestrebten Transparenz sind Architekturbestandteile *Stub* und *Skeleton*. Beide erfüllen symmetrische Aufgabenstellungen und werden mittels eines Dienstprogrammes automatisch aus dem übersetzten Bytecode des für den entfernten Zugriff anzubietenden Objekts erstellt.
Zur Ausführungszeit wirkt der Stub als Stellvertreter des entfernten Serverobjektes innerhalb der Adreßumgebung des Clients, analog fungiert der Skeleton als serverseitiger Stellvertreter des aufrufenden Objekts.

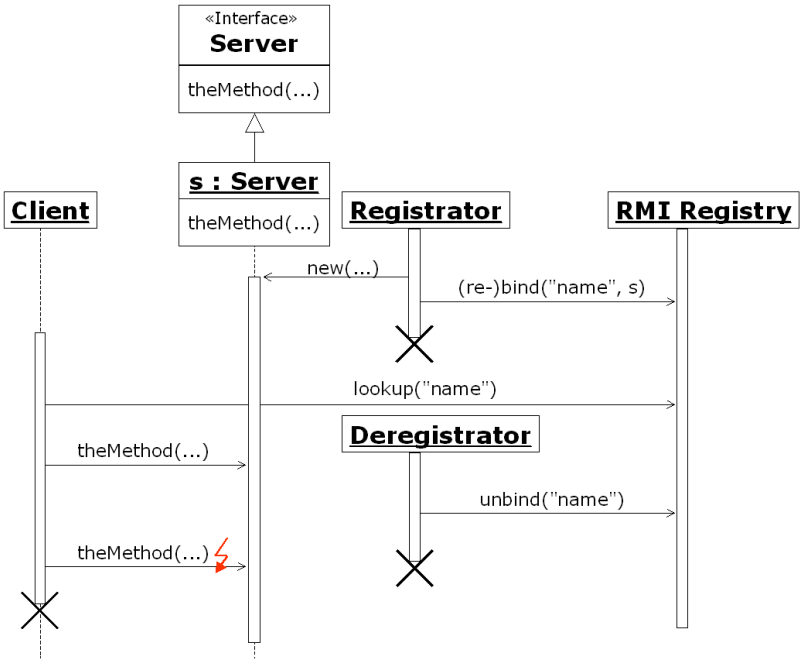
Abbildung 21 zeigt den Ablauf des Aufrufs eines entfernten Objekts mittels RMI. Hierbei spielen die Stub- und die Skeletonkomponente wie folgt zusammen:

1. Stub: Initiierung einer Verbindung mit derjenigen entfernt ablaufenden virtuellen Maschine welche das anzusprechende Objekt verwaltet.
2. Stub: Erzeugung der benötigten Leitungsdarstellung und Übertragung der Aufrufparameter an die entfernte virtuelle Maschine.
3. Stub: Warten auf Ausführungsende und evtl. erzeugten Rückgabeparameter.
4. Skeleton: Empfang und Decodierung der Leitungsdarstellung der Aufrufparameter.
5. Skeleton: Aufruf des (lokalen) Objekts und Entgegennahme etwaig erzeugter Rückgabewerte.
6. Skeleton: Erzeugung der benötigten Leitungsdarstellung und Übertragung der Rückgabewerte an die entferne virtuelle Maschine.
7. Stub: Empfang und Decodierung der Leitungsdarstellung des etwaig übergebenen Rückgabewertes.
8. Stub: Übergabe des Rückgabewertes (falls vorhanden) an (lokales) Aufruferobjekt.



Um das entfernte Objekt lokalisieren zu können, ohne daß dem Aufrufer der Server, auf dem dieses verwaltet wird, bekannt sein muß, wird als zusätzliche Architekturkomponente ein Verzeichnisdienst verwendet. Innerhalb dieses Dienstes wird jedes Objekt, auf das der entfernte Zugriff angeboten werden soll, ein eineindeutiger Name und die physische Objektlokation verwaltet.

Unter Berücksichtigung des Verzeichnisdiensts ergibt sich daher folgender Interaktionsablauf:



Das Bild zeigt das Objekt `s` als Instanz der Klasse `Server` sowie das auf die darin enthaltene Methode `theMethod` zugreifenden `Client`-Objekt. Zusätzlich ist mit `RMI Registry` der Verzeichnisdienst zur Verwaltung der zugreifbaren Objekte dargestellt. Die beiden Objekte `Registrator` bzw. `Deregistrator` dienen zur Ablage (Bindung) des `Server`-Objektes an einen im Verzeichnisdienst gespeicherten Namen bzw. der Aufhebung dieser Bindung.

Da die Bindung objektbezogen erfolgt, muß die registrierende Instanz (im Beispiel `Registrator`) eine Referenz auf das abzulegende Objekt besitzen. Die Beispielinteraktion unterstellt daher die Erzeugung der benötigten `Server`-Instanz durch den `Registrator` selbst. Dieser bindet das `Server`-Objekt unter dem Klarnamen `name` und legt diese Zuordnung durch Aufruf der Methode `bind` oder `rebind` in der Registry ab. Dort kann danach mittels der Methode `lookup`, unter Angabe des Namens eine Referenz auf das gebundene Objekt, erfragt werden. Die Interaktion zeigt diesen Aufruf durch den `Client`.
Ausgehend von dieser Referenz kann durch den `Client` die durch die `Server`-Schnittstelle publizierte Methode `theMethod` aufgerufen werden.
Ferner zeigt der Ablauf die Deregistrierung des `Server`-Objektes durch das `Deregistrator`-Objekt mittels der Methode `unbind`. Für diesen Aufruf ist keine Objektreferenz, sondern lediglich der Klarname des Objektes notwendig.

Vorgehen zur Entwicklung einer RMI-Applikation

Die Entwicklung einer RMI-basierten Applikation vollzieht sich immer im selben Ablauf der Erstellung der notwendigen Komponenten.
Dieser Ablauf ist nachfolgend anhand einer simplen Applikation veranschaulicht:

1. Definition der notwendigen Serverschnittstelle (HelloInterface)
2. Implementierung des Servers (HelloServer)
3. Implementierung des oder der Clientapplikationen (HelloClient)
4. Erzeugung der Stub- und Skeletonklassen (durch das Werkzeug rmic)
5. Start des RMI-Verzeichnisdiensts (rmiregistry)
6. Registrierung des Serverobjektes
7. Start des Servers
8. Ausführung des Clients

Die Schnittstelle HelloInterface:

Beispiel 94: Die Schnittstelle HelloInterface

```
(1)import java.rmi.Remote;
(2)import java.rmi.RemoteException;
(3)
(4)public interface HelloInterface extends Remote {
(5)     public String sayHello() throws RemoteException;
(6)}
```

Download des Beispiels

Die Schnittstelle enthält die Operationen aller Methoden, die ein Serverobjekt zur Verwendung zur Verfügung stellen soll.
Hierbei sind naturgemäß keine als private oder protected deklarierten Operationen zugelassen, da auf sie kein Zugriff möglich wäre.
Für jede Operation ist die möglicherweise auftretende Erzeugung einer Ausnahme des Typs RemoteException vorzusehen. Dieser Typ kapselt durch Kommunikationsfehler verursachte Ausnahmen.
Zusätzlich ist die gesamte Schnittstelle als Spezialisierung der Standardschnittstelle Remote zu definieren. Diese Schnittstelle definiert selbst keine Operationen, sondern dient lediglich zur Kennzeichnung von Schnittstellen, die entfernt aufrufbare Methoden versammeln.

Die Klasse HelloServer:

Beispiel 95: Die Klasse HelloServer

```
(1)import java.net.MalformedURLException;
(2)import java.rmi.Naming;
(3)import java.rmi.RemoteException;
(4)import java.rmi.server.UnicastRemoteObject;
(5)
(6)public class HelloServer
(7)     extends UnicastRemoteObject
(8)     implements HelloInterface {
(9)
(10)    protected HelloServer() throws RemoteException {
(11)            super();
(12)            try {
(13)                    Naming.rebind("rmi://localhost:1099/HelloService", this);
(14)            } catch (RemoteException re) {
(15)                    re.printStackTrace();
(16)            } catch (MalformedURLException mue) {
(17)                    mue.printStackTrace();
(18)            }
(19)    }
(20)
(21)    public static void main(String argv[]) throws RemoteException {
(22)            new HelloServer();
(23)    }
(24)
(25)    public String sayHello() throws RemoteException {
(26)            return (new String("Hello world!"));
(27)    }
(28)}
```

Download des Beispiels

Die Klasse enthält neben der Methode der durch die Schnittstelle exponierten Operation sayHello auch im Konstruktor die Registrierung innerhalb des RMI-Verzeichnisdienstes. Dieser ist jedoch, ebenso so wie alle anderen möglicherweise codierten Methoden, die nicht Bestandteil der explizierten Schnittstelle sind, nicht von einem entfernten Objekt aus zugreifbar.
Formal implementiert die Klasse mit HelloInterface die öffentlich zur Verfügung gestellten Operationen. Zusätzlich muß sie als Spezialisierung der API-Klasse UnicastRemoteObject definiert sein um durch die virtuelle Maschine zur Laufzeit als Objekt mit entfernten Zugriffen behandelt zu werden. Dies zieht von der

Behandlung des lokalen Falles abweichende Handhabungen der Parameterübergabe sowie der Freispeicherverwaltung und -rückgewinnung nach sich.
Die Implementierung der durch die Schnittstelle bekanntgegebenen und später durch entfernte Objekte nutzbaren Operation erfolgt identisch zur herkömmlichen Umsetzung im lokalen Falle. Einzige äußerlich erkennbare Änderung ist die Deklaration der möglicherweise bei der Ausführung auftretenden `RemoteException`. Diese wird jedoch üblicherweise nicht aktiv durch den anwenderdefinierten Code ausgelöst, sondern wird durch die Java-Standardbibliothek bzw. die -Laufzeitumgebung erzeugt. Die Deklaration dieser Ausnahme ist für Operationen, auf deren Implementierung entfernt zugegriffen werden soll, zwingend.
Zusätzlich nimmt die Applikation im Konstruktor die Registrierung des neu erzeugten Objektes unter dem frei gewählten Namen `HelloService` in der RMI-Registry vor. Hierbei wird die Methode `rebind` verwendet, um eine eventuell bereits existierende Objektbindung unter diesem Namen zu überschreiben. Auffallend ist die zur Bindung in der Registry verwendete URI `rmi://localhost:1099/HelloService`. Sie enthält nach dem idenzifizierenden URI-Schema (`rmi`) zwingend die Identifikation derjenigen Maschine, die das bereitzustellende Objekt verwaltet. Einschränkend sei hier jedoch angemerkt, daß RMI lediglich die Registrierung von Objekten vorsieht, die auf derselben physischen Maschine verwaltet werden wie der Registrierungsdienst selbst. Daher ist neben dem Pseudonamen `localhost` lediglich der Name der Maschine bzw. deren IP-Adresse oder die Pseudoadresse `127.0.0.1` zugelassen. Die optionale Portnummer benennt die logische TCP-Portadresse, über welche Kommunikationen mit der RMI-Registry abgewickelt werden. Daran im Anschluß folgt der anwenderdefinierte Namen des abzulegenden Objekts. Obwohl nach erfolgreicher Ausführung der Methode `rebind` die Kontrolle an das neu erzeugte Objekt der Klasse `HelloServer` zurückübergeben wird, terminiert das Programm nach Abarbeitung des Konstruktors (und damit Abschluß der Anweisungen in `main`) nicht, sondern wird durch die RMI-Registry wartend auf Aufrufe der Methode `sayHello` im Hauptspeicher gehalten.

Die Klasse `HelloClient`:

Beispiel 96: Die Klasse HelloClient

```
(1)import java.net.MalformedURLException;
(2)import java.rmi.Naming;
(3)import java.rmi.NotBoundException;
(4)import java.rmi.RemoteException;
(5)
(6)public class HelloClient {
(7)     public static void main(String[] args) {
(8)             HelloInterface obj = null;
(9)             try {
(10)                    obj = (HelloInterface) Naming.lookup("rmi://53.16.71.55:1099/
HelloService");
(11)            } catch (MalformedURLException mue) {
(12)                    mue.printStackTrace();
(13)            } catch (RemoteException re) {
(14)                    System.err.println("Lookup error");
(15)                    re.printStackTrace();
(16)            } catch (NotBoundException nbe) {
(17)                    System.out.println("Object not bount to registry");
(18)                    nbe.printStackTrace();
(19)            }
(20)            System.out.println("retrieved object's oid=" + obj);
(21)            try {
(22)                    System.out.println(obj.sayHello());
(23)            } catch (RemoteException re) {
(24)                    System.err.println("Error while invoking sayHello");
(25)                    re.printStackTrace();
(26)            }
(27)    }
(28)}
```

Download des Beispiels

Die Klasse des Beispiels 96 enthält den notwendigen Code um auf eine entfernt verwaltete und abgelegte Ausprägung der Schnittstelle `HelloInterface` zuzugreifen.
Zunächst wird vermöge der Methode `lookup` eine Referenz eines `HelloInterface`-konformen Objekts aus dem RMI-Verzeichnisdienst erfragt. Hierzu dient die bereits zur Ablage verwendete URI auch zu Ermittlung des Objekts durch das nutzungswillige Clientobjekt. Naturgemäß müssen sich Client- und Serverobjekt nicht auf derselben physischen Maschine befinden, sondern müssen lediglich durch eine bestehende Netzwerkverbindung miteinander kommunizieren können.
Nach Erhalt der Referenz kann das entfernte Objekt einem durch die lokale virtuelle Maschine verwaltetem gleichgestellt verwendet werden.
Da das Server-Objekt auch nach Abarbeitung seiner Methode resident im Speicher der entfernten virtuellen Maschine verbleibt ermittelt jeder Client bei jedem Aufruf der Methode `lookup` denselben Verweis auf dasselbe physische Objekt. Aus diesem Grunde ist bei der Programmierung darauf zu achten, daß sich nebenläufig stattfindende Zugriffe nicht gegenseitig behindern oder inkorrekte Ergebnisse liefern.

Die **Nutzung des Verzeichnisdienstes ist keine notwendige Voraussetzung** zur Ausführung entfernter Methodenaufrufe. Besitzt der Client bereits eine Referenz auf das entfernte Objekt (etwa aus einem Aufruf durch dieses welcher die Objektreferenz als Parameter enthielt), so kann der entfernte Methodenaufruf auch ohne vorherige Ermittlung der Referenz aus dem RMI-Verzeichnisdienst erfolgen.

Die **Parameterübergabe zwischen Client und entferntem Serverobjekt** geschieht anders als in der Java-üblichen Weise, d.h. für primitivwerte Parameter als Wert und Objektwertige als Referenz, durchgängig durch Wertübergabe.
Dies liegt darin begründet, daß der innerhalb der virtuellen Maschine verwandte Referenzierungsmechanismus auf Objekte als Inhalte des Heaps der virtuellen Maschine, nicht über Rechnergrenzen hinweg portabel gestaltet ist. Durch den notwendigen Aufwand der durch die Stub- und Skeletonkomponente zu codierenden bzw. zu decodierenden Daten steigt bei komplexen Objektstrukturen (z.B. Bäumen) sowohl die CPU-Last der Client- als auch der Servermaschine sowie die Netzwerklast.
Einzige Ausnahme von dieser Mimik bilden per RMI zugreifbare (entfernte) Objekte selbst. Sie werden nicht serialisiert und übertragen, sondern transparent durch ihre Stub-Repräsentation ersetzt, die an ihrer statt übertragen wird. Dies gilt insbesondere natürlich auch für Selbstreferenzen des Serverobjektes, die dieses als Rückgabewert liefert.

Naturgemäß ergeben sich auch für die **verteilte Freispeicherverwaltung** (distributed garbage collection) veränderte Rahmenbedinungen, wenngleich gemäß dem RMI-Designziel größtmöglicher Transparenz versucht wurde, eine ähnliche Handhabung wie bereits für den lokalen Fall zu realiseren.
Das in RMI gewählte Vorgehen orientiert sich an dem in der Programmiersprache Modula-3 realisierten: Jede virtuelle Maschine führt eine Tabelle mit einem Boole'schen Eintrag für jedes aufgrund der publizierten Schnittstellendefinition potentiell entfernt zugreifbare Objekt. Jeder Eintrag ist vorgabegemäß mit dem Wert *clean* belegt. Wird eine Referenz auf das Objekt erfragt, so wird dieses als *dirty* markiert. Erfolgt innerhalb einer gewissen Zeit (*lease Zeit*) keine erneute Anfrage nach einer weiteren Referenz oder ein Zugriff auf das Objekt, so wird die Markierung auf *clean* zurückgesetzt und das Objekt so zur Freigabe markiert.
Die Zeitspanne bis zur automatisierten Markierung als nicht mehr benötigtes Objekt kann durch die Java-Property `java.rmi.dgc.leaseValue` kontrolliert werden. Vorgabegemäß ist sie auf zehn Minuten gesetzt.

Die Activation-API

Neben der u.U. speicherplatz- und laufzeitaufwendigen „Dauerinstanziierung“ eines entfernt zugreifbaren Objektes, wie sie das vorherige Beispiel einführt, existiert die Möglichkeit, Objekte bei Bedarf dynamisch erzeugen zu lassen.

Das nachfolgende Beispiel zeigt dies. Der Ablauf in Erstellung und Ausführung ähnelt dem zuvor für einfache RMI-Applikationen gezeigten, jedoch unter Nutzung einer zusätzlichen Ausführungskomponente.

1. Definition der notwendigen Serverschnittstelle (`ActivatableHelloInterface`)
2. Implementierung des Servers (`ActivatableHelloServer`)
3. Implementierung einer Applikation zur Registrierung des Servers (`ActivatableSetup`)
4. Implementierung des oder der Clientapplikationen (`ActivatableHelloClient`)
5. Erzeugung der Stub- und Skeletonklassen (durch das Werkzeug `rmic`)
6. Start des RMI-Verzeichnisdiensts (`rmiregistry`)
7. Start des RMI-Daemons (`rmid`)
8. Registrierung des Serverobjektes
9. Ausführung des Clients

Im Vergleich zum vorhergehenden Ablauf wurde in diesem Beispiel die Registrierung des Servers in eine eigenständige Applikation ausgelagert, da ihr Ablauf im Konstruktor des Serverobjektes bei wiederholter Instanziierung durch die RMI-Umgebung redundant ausgeührt würde.
Ferner entfällt der explizite Start des Serverobjektes, da diese Aufgabe nunmehr automatisiert durch das Framework übernommen wird.
Die Komponente welcher die Kontrolle und Durchführung der Startvorgänge obliegt, der RMI-Daemon, muß dafür einmalig zusätzlich durch den Anwender gestartet werden.

Die Schnittstelle `ActivatableHelloInterface`:

Beispiel 97: Die Klasse ActivatableHelloInterface

```
(1)import java.rmi.Remote;
(2)import java.rmi.RemoteException;
(3)
(4)public interface ActivatableHelloInterface extends Remote {
(5)     public String sayHello() throws RemoteException;
(6)}
```

Download des Beispiels

Seitens der Schnittstellenbeschreibung der exponierten Operationen ergeben sich durch den Übergang auf

die servergesteuerte Instanziierung keine Änderungen.

Die Klasse ActivatableHelloServer:

Beispiel 98: Die Klasse ActivatableHelloServer

```
(1)import java.rmi.MarshalledObject;
(2)import java.rmi.RemoteException;
(3)import java.rmi.activation.Activatable;
(4)import java.rmi.activation.ActivationID;
(5)
(6)public class ActivatableHelloServer
(7)     extends Activatable
(8)     implements ActivatableHelloInterface {
(9)
(10)    public ActivatableHelloServer(ActivationID id, MarshalledObject data)
(11)            throws RemoteException {
(12)            super(id, 0);
(13)    }
(14)
(15)    public String sayHello() throws RemoteException {
(16)            return (new String("Hello world!"));
(17)    }
(18)}
```

Download des Beispiels

Seitens der Implementierung des für entfernte Aufrufe zur Verfügung gestellten Objekts ergeben sich durch die geänderte Instanziierungsmimik keine gravierenden Änderungen.
Lediglich die geänderte Spezialisierung, statt UnicastRemoteObject wird jetzt von der Klasse Remote abgeleitet, fällt zunächst ins Auge. Diese Änderung ist notwendig, um dem Framework die Möglichkeit der Instanziierung zu eröffnen, die bei einer völlig freien Formulierung --- insbesondere des Konstruktors --- nicht möglich wäre. Im Kern definiert die Klasse Activateable, neben einer Reihe von Hilfsmethoden, vier verschiedene Konstruktoren zur Anlage von Objekten. Mindestens einer davon ist zwingend durch die abgeleitete Klasse zu implementieren um dem eine automatisierte Objekterzeugung zu erreichen.
Hinsichtlich der Problemlogik, im Beispiel der Methode sayHello ergeben sich keine Änderungen.

Die Klasse ActivatableHelloClient:

Beispiel 99: Die Klasse ActivatableHelloClient

```
(1)import java.net.MalformedURLException;
(2)import java.rmi.Naming;
(3)import java.rmi.NotBoundException;
(4)import java.rmi.RemoteException;
(5)
(6)public class ActivatableHelloClient {
(7)     public static void main(String[] args) {
(8)             ActivatableHelloInterface obj = null;
(9)             try {
(10)                    obj = (ActivatableHelloInterface) Naming.lookup
("rmi://53.16.71.55:1099/HelloObj");
(11)            } catch (MalformedURLException mue) {
(12)                    mue.printStackTrace();
(13)            } catch (RemoteException re) {
(14)                    System.err.println("Lookup error");
(15)                    re.printStackTrace();
(16)            } catch (NotBoundException nbe) {
(17)                    System.out.println("Object not bount to registry");
(18)                    nbe.printStackTrace();
(19)            }
(20)            System.out.println("retrieved object's oid=" + obj);
(21)            try {
(22)                    System.out.println(obj.sayHello());
(23)            } catch (RemoteException re) {
(24)                    System.err.println("Error while invoking sayHello");
(25)                    re.printStackTrace();
(26)            }
(27)    }
(28)}
```

Download des Beispiels

Wie bereits beim Server gezeigt, ergeben auch Clientseitig durch die Veränderte Erzeugungssemantik keine Änderungen in der Ablauflogik, abgesehen von durch die geänderten Namen dieses Beispiels

bedingte Modifikationen.

Die Klasse `ActivatableSetup`:

Beispiel 100: Die Klasse ActivatableSetup

```
(1)import java.net.MalformedURLException;
(2)import java.rmi.MarshalledObject;
(3)import java.rmi.Naming;
(4)import java.rmi.RMISecurityManager;
(5)import java.rmi.RemoteException;
(6)import java.rmi.activation.Activatable;
(7)import java.rmi.activation.ActivationDesc;
(8)import java.rmi.activation.ActivationException;
(9)import java.rmi.activation.ActivationGroup;
(10)import java.rmi.activation.ActivationGroupDesc;
(11)import java.rmi.activation.ActivationGroupID;
(12)import java.rmi.activation.UnknownGroupException;
(13)import java.util.Properties;
(14)
(15)public class ActivatableSetup {
(16)    public static void main(String argv[]) throws RemoteException {
(17)            System.setSecurityManager(new RMISecurityManager());
(18)            Properties props = new Properties();
(19)            props.put("java.security.policy", "policy");
(20)            ActivationGroupDesc.CommandEnvironment ace = null;
(21)            ActivationGroupDesc exampleGroup = new ActivationGroupDesc(props, ace);
(22)            ActivationGroupID agi = null;
(23)            try {
(24)                    agi = ActivationGroup.getSystem().registerGroup(exampleGroup);
(25)            } catch (RemoteException re) {
(26)                    re.printStackTrace();
(27)            } catch (ActivationException ae) {
(28)                    ae.printStackTrace();
(29)            }
(30)            String location = "file:/home/mario/RMITest/activation/";
(31)            MarshalledObject data = null;
(32)            ActivationDesc desc =
(33)                    new ActivationDesc(agi, "ActivatableHelloServer", location, data);
(34)
(35)            ActivatableHelloInterface stub = null;
(36)            try {
(37)                    stub = (ActivatableHelloInterface) Activatable.register(desc);
(38)            } catch (UnknownGroupException uge) {
(39)                    uge.printStackTrace();
(40)            } catch (RemoteException re) {
(41)                    re.printStackTrace();
(42)            } catch (ActivationException ae) {
(43)                    ae.printStackTrace();
(44)            }
(45)            try {
(46)                    Naming.rebind("rmi://localhost:1099/HelloObj", stub);
(47)            } catch (RemoteException re) {
(48)                    re.printStackTrace();
(49)            } catch (MalformedURLException mue) {
(50)                    mue.printStackTrace();
(51)            }
(52)
(53)    }
(54)}
```

Download des Beispiels

Die nachhaltigsten Veränderungen spiegelt die neu eingeführte Klasse zur Registrierung des Objekts für entfernten Zugriff wieder.
Sie parametrisiert und initialisiert zunächst den RMI-Daemon, der später im Bedarfsfalle die Erzeugung der benötigten Objekte übernimmt. Daran Anschließend wird ein konform zur publizierten Schnittstelle umgesetztes Objekt gemäß dem bisherigen Verfahren innerhalb des RMI-Verzeichnisdienstes an einen frei gewählten Namen gebunden.
Der automatische Aktivierungsprozeß erfordert zur Laufzeit des RMI-Daemons den Zugriff auf ein lokales oder erreichbares Dateisystem. Für alle Fälle des Zugriffs auf ein nichtlokales Dateisystem muß daher eine geeignete Sicherheitseinstellung der Javaausführungsumgebung gewählt werden. Im Standardumfang der API bereits enthalten ist die Klasse `RMISecurityManager`. Sie erlaubt es dynamisch geladenen Code innerhalb einer RMI-Umgebung auszuführen.
Zusätzlich kann über die Java-Property `java.security.policy` eine sog. *policy*-Datei referenziert werden,

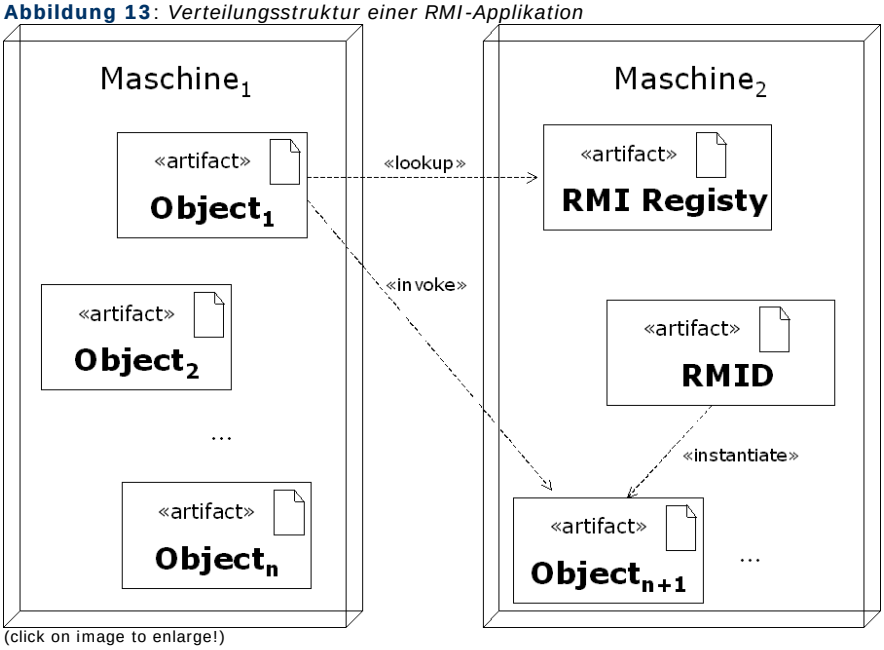
die nach Herkunft abgestufte Zugriffsrechte für signierte Inhalte definiert.
Für das wiedergegebene Beispiel wurde eine eine sehr lose Policy gewählt, die jeglichem Code den Zugriff auf alle Java-Funktionalitäten gestattet. (Policy-File) Mittels der Policyeigenschaften `com.sun.rmi.rmid.ExecPermission` und `com.sun.rmi.rmid.ExecOptionPermission` läßt sich eine noch feinere Zugriffssteuerung erreichen, wobei jedoch diese vorgabegemäß immer unabhängig von der Codeherkunft und einer evtl. existierenden Signatur zu definieren sind.
Die späteren Aktivierungsaufrufe werden innerhalb einer *Aktivierungsgruppe* gebündelt. Eine solche wird mittels einer Beschreibung (als Ausprägung der Klasse `ActivationGroupDesc`) hinsichtlich ihrer eineindeutigen Identität und ihrer Inialisierungsdaten beschrieben (im vorliegenden Beispiel ist sie mit `null` initialisiert).
Diese Aktivierungsgruppe wird mit den durch die Policy-Datei definierten Rechten ausgestattet erzeugt.
Die notwendige eineindeutige Identität wird durch das Aktivierungssystem mittels des Aufrufs der Methode `registerGroup` automatisch zur Verfügung gestellt. Hierbei gilt eine eindeutige Zuordnung zwischen Gruppenkennung (`ActivationGrouID`) und verwaltender virtueller Maschine. Soll daher ein Objekt oder eine ganze Gruppe in einer separaten virtuellen Maschine ablaufen, so genügt es, eine andere Gruppenkennung durch das System zu generieren und bei der nachfolgend diskutierten Registrierung zu übergeben.
Anschließend wird durch das Beschreibungsobjekt mit dem Klassennamen des automatisch zu instanziierenden Objektes (im Beispiel: `ActivatableHelloServer`) sowie der physischen Lokation der Klasse und etwaigen Initalisierungsdaten erzeugt und der Aktivierungsgruppe zugeordnet.
Als letzter Schritt erfolgt die Registrierung der durch die Aktivierungsbeschreibung beschriebenen Klasse von Objekten, um so ihre spätere Erzeugung zu gewährleisten. Dies geschieht durch Aufruf der statischen Methode `register` der als Parameter die erzeugte Aktivierungsbeschreibung übergeben wird.
Der Aufruf dieser Methode liefert den Stub des registrierten Objekts. Aufgrund der Stellvertreterrolle des Stubs (d.h. seiner Konformität zur Schnittstelle `Remote`) kann dieser zur Registrierung der Objekte in der RMI-Registry verwendet werden.

Der Aufruf der Klasse `ActivationSetup` erfolgt durch Übergabe von gültigen Belegungen der notwendigen Laufzeiteigenschaften an die virtuelle Javamaschine: `java -Djava.security.policy=./policy -Djava.rmi.codebase=file:/home/mario/RMITest/activation/ ActivationSetup`.

Objekte werden zwar durch den RMI-Daemon automatisiert aktiviert, sobald ein Zugriff auf sie erfolgt (*lazy activation*), jedoch nicht mehr deaktiviert. Das bedeutet, daß sie bis zur Terminierung der virtuellen Maschine im Hauptspeicher verbleiben. Abhilfe schafft hier der explizite Aufruf der Methode `inactive` der ein bereits bestehendes Objekt passiviert, jedoch nicht aus der Registrierung entfernt.

Ähnlich wie bereits für die RMI-Registry angemerkt, muß auch der RMI-Daemon auf derselben physischen Maschine zur Ausführung gelangen, die das für entfernte Zugriffe bereitzustellende Objekt enthält.
Es ergibt sich daher generell folgendes Verteilungsschema:

Abbildung 13: *Verteilungsstruktur einer RMI-Applikation*



(click on image to enlarge!)

Um die irreführenden Begriffe *Client* und *Server* zu vermeiden, sind die beiden Maschinen neutral durch $\texttt{Maschine}_1$ und $\texttt{Maschine}_2$ bezeichnet (Die Irreführung erklärt sich aus der mangelnden Trennschärfe der beiden Begriffe, sobald ein dienstanbieter Knoten auch gleichzeitig Dienste anderer Knoten nutzt, er damit gleichzeitig Server (seines Anfrages) und Client (bezüglich seiner Anfrage) ist).
Ein $\texttt{Object}_1$, das auf ein entferntes $\texttt{Object}_{n+1}$ zugreift (`invoke`) kann optional mittels der `RMI Registry`, welche auf demselben Rechner wie die virtuelle Maschine die das $\texttt{Object}_{n+1}$ verwaltet ablaufen muß, die notwendige Objektreferenz ermitteln (`lookup`). Sofern das zuzugreifende Objekt (noch) nicht existiert kann es über den Aktivierungsmechanismus gestartet werden. Die notwendige Objekterzeugung (`instantiate`) nimmt hierbei der RMI-Daemon (`RMID`) vor.

Sicherheitsgesichtspunkte

Generell werden die durch Stubs und Skelettons verarbeiteten Objekte oder Primitivwerte lediglich serialsiert, aber keine weiteren Maßnahmen zur Gewährleistung der Vertraulichkeit der übertragenen Daten ergriffen.
Durch die Integration des RMI-Mechanismus in die Java-API besteht die Möglichkeit innerhalb der Programmiersprache Möglichkeiten zur Übertragungssicherung zu ergreifen.
Seitens RMIs ist es hierbei vorgesehen die bestehende (einfache) TCP-Socketverbindung gegen beliebige selbsterstellte auszutauschen. Die Erzeugung der notwendigen Sockets muß mittels der Schnittstellen `RMIServerSocketFactory` bzw. `RMIClientSocket` abgewickelt werden. Auf diesem Wege stellt bleibt die Typsicherheit der Laufzeitumgebung, auch bei abgeänderten Transportprotokollen gewahrt.
Das nachfolgende Beispiel zeigt die Umsetzung einer einfachen gesicherten Verbindung mittels XOR-Bearbeitung aller versandten Inhalte.

Aus Gründen der Übersichtlichkeit sind die inhaltlich unverändert aus dem ersten Beispiel dieses Kapitels übernommenen Dateien `Hello.java` sowie `HelloClient.java` nicht wiedergegeben.

Die Klasse `HelloImpl`:

Beispiel 101: Die Klasse HelloImpl

```
(1)import java.rmi.Naming;
(2)import java.rmi.RMISecurityManager;
(3)import java.rmi.RemoteException;
(4)import java.rmi.server.UnicastRemoteObject;
(5)
(6)public class HelloImpl extends UnicastRemoteObject implements Hello {
(7)     public HelloImpl(String protocol, byte[] pattern) throws RemoteException {
(8)             super(
(9)                     0,
(10)                    new XorClientSocketFactory(protocol, pattern),
(11)                    new XorServerSocketFactory(protocol, pattern));
(12)    }
(13)
(14)    public String sayHello() throws RemoteException {
(15)            StringBuffer sb = new StringBuffer();
(16)            for (int i=0;i<100;i++)
(17)                    sb.append("X");
(18)            return sb.toString();
(19)    }
(20)
(21)    public static void main(String args[]) {
(22)
(23)            System.setSecurityManager(new RMISecurityManager());
(24)            byte[] aPattern = {(byte) 1011 };
(25)            try {
(26)                    HelloImpl obj = new HelloImpl("xor", aPattern);
(27)                    Naming.rebind("rmi://53.16.71.55:1099/HelloServer", obj);
(28)                    System.out.println("HelloServer bound in registry");
(29)            } catch (Exception e) {
(30)                    System.out.println("HelloImpl err: " + e.getMessage());
(31)                    e.printStackTrace();
(32)            }
(33)    }
(34)}
```

Download des Beispiels

Die Klasse zeigt als einzige offensichtliche Veränderung die Ergänzung des Konstruktors des entfernt zugreifbaren Objekts um die Erzeugung der alternativ verwendeten XOR-Sockets; alle weiteren Aufwände die veränderte Kommunikation abzuwickeln sind für den Programmierer transparent und werden durch das RMI-Framework sowie die Implementierung der XOR-Sockets erbracht.

Die Bestandteile der XOR-Socketimplementierung finden sich hier:

- `XorClientSocketFactory`
- `XorInputStream`
- `XorOutputStream`
- `XorServerSocket`
- `XorServerSocketFactory`
- `XorSocket`

Web-Referenzen 12: Weiterführende Links

- RMI @ SUN
- RMI Tutorial @ SUN
- Ninja RMI eine freie RMI-Implementierung
- JDBC FAQ @ JGuru.com
- G. Reese: *Database Programming with JDBC and Java*. O'Reilly, 1997.
- Verhältnis von X/Open CLI und ODBC

4.3 Representational State Transfer (REST)

Hinweis: Dieses Kapitel ist noch unvollständig

Servlets

Ergebnisse eines Anwendungssystems werden typischerweise zur Laufzeit auf der Basis von Anwendungslogik und in Abhängigkeit der Eingaben berechnet. Dies gilt auch für Web-basierte Anwendungssysteme. Jedoch stellt das HTML-basierte Web konzeptionell zunächst ein statisches Medium, welches vorgefertigte Hypertextseiten auf über das HTTP-Protokoll übermittelte Anforderung ausliefert, dar.

Zur Versöhnung dieses Widerspruchs haben sich zwei Ansätze herausgebildet: Zum einen die vollständige Verlagerung der Berechungslogik und Erbebnispräsentation in den anfragenden Client durch Übertragung eines vollständigen binären Programms. Zum anderen der Aufruf von serverseitig vorgehaltenen Applikationen, die nicht im Adresskontext des Webservers operieren.
Bekanntester Vertreter der ersten Kategorie sind die *Java Applets*, welche eine als *Java Bytecode* bezeichnete portable Binärrepräsentation an den Client übertragen, die dieser durch eine interpretativ arbeitende *virtuelle Maschine* ausführt. Die in der Praxis bedeutsamste Realisierung zum Zugriff auf extern vorliegende Programme stellt das *Common Gateway Interface* (CGI) dar.

Jedoch ergeben sich bei beiden Lösungsalternativen spezifische Probleme. So wird zur vollständigen Verlagerung der Ausführungslogik auf den Client, wie sie für Applets propagiert wird, die Möglichkeit zur Ausführung fremden Codes erzwungen. Dies läßt sich zwar durch geeignete Sicherheitsmechanismen absichern, jedoch bleibt die Notwendigkeit, den Code per Internet zu beziehen und lokal auszuführen.
Der CGI-Ansatz krankt an der mangelnden Ausführungsgeschwindigkeit, welche durch die notwendigen Prozeßkontextwechsel drastisch sinkt. Gleichzeitig unterliegt die auf diesem Wege angebundene Kontrollogik nicht der Verwaltungshoheit des Servers.

Java Servlets stellen konzeptionell eine konfluente Weiterentwicklung der beiden Ansätze dar, die es sich zum Ziel gesetzt hat die angeführten Nachteile zu mildern.
Hierbei handelt es sich primär um Server-seitig ausgeführten Java-Code, der über Internetprotokolle (zumeist HTTP) mit dem Benutzer interagiert. Technisch gesehen stellen Servlets Module eines Web-Servers dar, die dynamisch anstelle der Auslieferung statischer Inhalte (z.B. HTML-Seiten) ausgeführt werden. Zur Ausführung dient eine als *Servlet Container* bezeichnete Softwarekomponente, welche die durch die dort abgelegten Servlets bedienten Schnittstellen der Standard-API bedient.

Bis zur Freigabe der Java-2-Plattform war die Servlet-API Bestandteil des JDK. Mittlerweile sind neben der Servlet-Spezifikation von SUN auch einige Implementierungen in verbreiteten Web-Servern verfügbar. Daher wurde die gesamte Servlet-API aus dem `javax`-Paket entfernt, und steht nunmehr nur noch als Bestandteil diverser Web-Server-Produkte (u.a. J2EE-Servern) zur Verfügung.

Lebenszyklus eines Servlets:

1. Der Server lädt das Servlet
2. Clientanforderungen werden durch das Servlet bedient
3. Der Server entlädt das Servlet (evtl. erst während des Server-Shutdown-Vorganges)

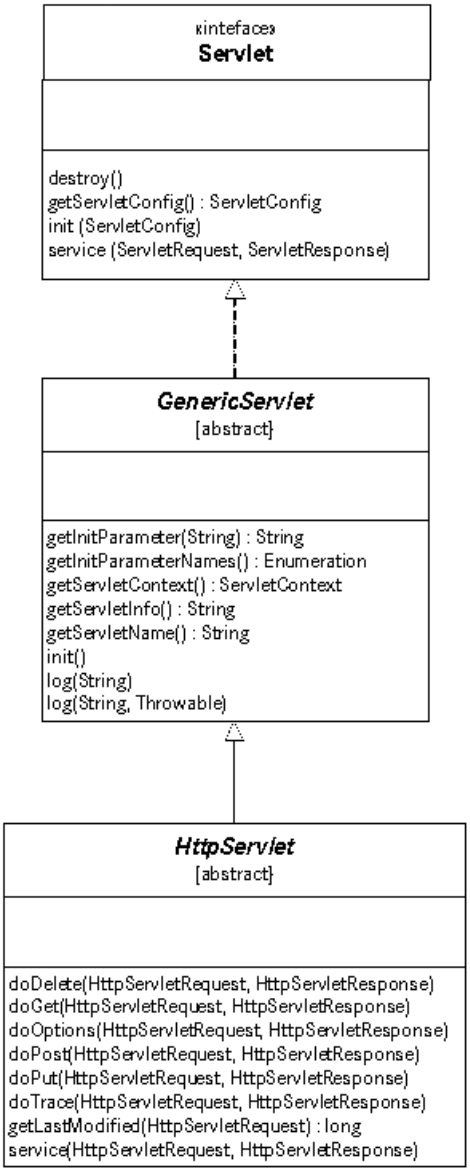
Im Detail korrespondieren diese drei Phasen mit Methoden der Servlet-Schnittstelle:

- `init(ServletConfig)`
  Initialisierung des Servlets nach dem Ladevorgang, und vor dessen Ausführung.
- `service(ServletRequest, ServletResponse)`
  Aufruf durch Servlet-Container (üblicherweise Web-Server) um Client-Anfrage zu bedienen.
  Pro Anfrage wird durch den Container vorgabegemäß ein neuer Thread gestartet. Beim Zugriff auf gemeinsam genutzte Ressourcen (Dateien, Datenbanken, etc.) ist daher zusätzlicher Synchronisationsaufwand notwendig.
- `destroy()`
  Wird vom Servlet-Container bei der Deaktivierung aufgerufen. Zuvor haben alle noch laufenden Threads Gelegenheit ihre Ausführung normal zu beenden. Diese Methode sollte beanspruchte Ressourcen freigeben und ggf. Zustände persistent sichern. Nach Ablauf von `destroy()` erfolgen keine Aufrufe an `service()` mehr durch den Servlet Container.

Ergänzend existieren noch die beiden Methoden `getServletConfig()` zum Zugriff auf die bei der Initialisierung übergebenen Parameter, und `getServletInfo()` zur Rückgabe einer beliebigen Zeichenkette (zumeist für Copyrightinformation o.ä. benutzt).

Die Servlet API gliedert sich in drei Stufen. Die Schnittstelle `Servlet` definiert alle notwendigen Verwaltungsoperationen, die innerhalb eines Servlets durch Methoden zu implementieren sind. Gleichzeitig liefert die diese Operationen umsetzende abstrakte Klasse `GenericServlet` eine erste einfache Basisimplementierung dieser Operationen. Die Spezifikation empfiehlt konkrete protkollspezifische Servletimplementierungen immer als Spezialisierung dieser Klasse zu realisieren. Für das populäre HTTP-Protokoll liefern die Standardimplementierungen mit `HttpServlet` eine solche Implementierung mit. Abbildung 1 stellt die Struktur der Basis API in der Übersicht zusammen.

Abbildung 14: *Struktur der Servlet-API*



(click on image to enlarge!)

Struktur eines Servlets:
Die Schnittstelle `Servlet` definiert alle Methoden, die ein Servlet zwingend implementieren muß. Dies kann durch eigenen Code geschehen, oder durch Erben von einer der vorgegebenen Klassen wie `GenericServlet` oder dem gebräuchlicheren `HttpServlet`.
Durch die Schnittstelle werden bereits Primitive für die beiden möglichen Kommunikationsrichtungen eingeführt: `ServletRequest` (Kommunikation vom Client zum Server) und `ServletResponse` (inverse Kommunikationsrichtung; Server an Client), als Parameter eines durch ein Servlet offerierten services. Beide Primitiven sind wiederum als Schnittstellen definiert, die durch konkrete Implementierungen (wie `HttpServletRequest` bzw. `HttpServletResponse`) umgesetzt werden müssen.

Die Schnittstelle **`ServletRequest`** ermöglicht es dem Servlet auf Parameternamen, Protkolldetails und Rechnernamen der sendenden Maschine zuzugreifen.
Ferner stellt die Schnittstelle den `ServerInputStream` und den `PrintReader` zur Verfügung, über den Daten des Clients (beispielsweise per HTTP POST oder PUT) empfangen werden können.

Die Schnittstelle **`ServletResponse`** erlaubt es dem Servlet Verwaltungsinformation, wie MIME-Typ und Content-Länge, der Antwort zu setzen.
Darüberhinaus stellt es mit dem `ServletOutputStream` einen Kommunikationskanal für binäre

Antwortdaten zur Verfügung. Unicode Text-artige Antworten können per `PrintWriter` übermittelt werden.

Die abstrakte **Klasse `HttpServlet`** stellt die notwendigen Methoden zur Bedienung von Client-Anforderungen bereit. Mindestens eine dieser Methoden ist zu überschreiben, um das gewünschte Verhalten des Servlets zu implementieren.

Die Implementierung des Beispiels 102 zeigt ein einfaches Servlet, welches nach einem Aufruf ein HTML-Dokument erzeugt und zurückliefert.

Beispiel 102: Ein einfaches Servlet

```
(1)import java.io.IOException;
(2)import java.io.PrintWriter;
(3)import java.util.Date;
(4)import javax.servlet.ServletException;
(5)import javax.servlet.http.HttpServlet;
(6)import javax.servlet.http.HttpServletRequest;
(7)import javax.servlet.http.HttpServletResponse;
(8)
(9)public class HelloWorld extends HttpServlet {
(10)    public void doGet(HttpServletRequest request, HttpServletResponse response)
(11)            throws IOException, ServletException {
(12)            response.setContentType("text/html");
(13)            PrintWriter out = response.getWriter();
(14)            out.println("<html>");
(15)            out.println("<head>");
(16)            out.println("<title>Hello World!</title>");
(17)            out.println("</head>");
(18)            out.println("<body>");
(19)            out.println("<h1>Hello World!</h1>");
(20)            out.println("<p>Current Date: "+new Date().toString()+"</p>");
(21)            out.println("</body>");
(22)            out.println("</html>");
(23)    }
(24)}
```

Download des Beispiels

Im Beispiel wird die `goGet`-Methode überschrieben, d.h. das Servlet reagiert auf HTTP GET-Anfragen. Als Antwort (ein wohlgeformtes HTML-Dokument) wird über den `PrintWriter` als Text (MIME-Typ `text/html`) versandt.

Der Servlet Container initialisiert und übergibt jedem Aufruf der Methode `doGet` ein Objekt des Typs `HttpServletRequest` und eines des Typs `HttpServletRespone` (welches im vorhergehenden Beispiel zur Übermittlung HTML-codierter Ergebnisse genutzt wurde.
Der Übergabeparameter des Typs `HttpServletRequest` ermöglicht den Zugriff auf die mit dem Aufruf des Servlets versandten Parameter. Beispiel 103 zeigt den Fall eines HTTP-basierten Servlet, welches die Parameter des `GET`-Aufrufs eingebettet in die URL erhält. Ein möglicher Aufruf enthält daher neben der Lokation des Dienstes auch die notwendigen Aufrufparameter: `http://www.example.com/addService?a=1&b=2`.

Beispiel 103: Ein einfacher Dienst

```
(1)import java.io.IOException;
(2)import java.io.PrintWriter;
(3)
(4)import javax.servlet.http.HttpServlet;
(5)import javax.servlet.http.HttpServletRequest;
(6)import javax.servlet.http.HttpServletResponse;
(7)
(8)public class SimpleService extends HttpServlet {
(9)     public void doGet(HttpServletRequest req, HttpServletResponse resp)
(10)            throws IOException {
(11)            resp.setContentType("text/xml");
(12)            PrintWriter out = resp.getWriter();
(13)            out.println(
(14)                    Integer.parseInt(
(15)                            req.getParameter("a")
(16)                                    + Integer.parseInt(req.getParameter("b"))));
(17)    }
(18)}
```

Download des Beispiels

Neben den bisher genutzten `GET`-Anfragen stehen im Rahmen der Standard-API auch Methoden zur

Behandlung der übrigen HTTP-Methoden zur Verfügung. Beispiel 104 zeigt die Funktionalität zum Zugriff auf Parameter im Rahmen eines DELETE-Aufrufs, auf HTTP-Headerdaten im Rahmen eines GET-Aufrufs sowie die Extraktion von Nutzdaten eines POST-Aufrufs.

Beispiel 104: Verschiedene Servletmethoden

```
(1)import java.io.IOException;
(2)import java.io.PrintWriter;
(3)import java.util.HashMap;
(4)
(5)import javax.servlet.ServletInputStream;
(6)import javax.servlet.http.HttpServlet;
(7)import javax.servlet.http.HttpServletRequest;
(8)import javax.servlet.http.HttpServletResponse;
(9)
(10)public class Methods extends HttpServlet {
(11)    private HashMap hm;
(12)    public void init() {
(13)            hm = new HashMap();
(14)    }
(15)
(16)    public void doDelete(HttpServletRequest req, HttpServletResponse resp) {
(17)            hm.remove(req.getParameter("key"));
(18)    }
(19)
(20)    public void doGet(HttpServletRequest req, HttpServletResponse resp)
(21)            throws IOException {
(22)            resp.setContentType("text/html");
(23)            PrintWriter out = resp.getWriter();
(24)            out.println(hm.get(req.getHeader("key")));
(25)    }
(26)
(27)    public void doPost(HttpServletRequest req, HttpServletResponse resp) {
(28)            byte line[] = new byte[100];
(29)            String key = null;
(30)            String value;
(31)            ServletInputStream sip = null;
(32)            try {
(33)                    sip = req.getInputStream();
(34)            } catch (IOException e) {
(35)                    e.printStackTrace();
(36)            }
(37)            try {
(38)                    sip.readLine(line, 0, 100);
(39)            } catch (IOException e1) {
(40)                    e1.printStackTrace();
(41)            }
(42)            String lineStr = new String(line, 0, line.length);
(43)            key = lineStr.substring(lineStr.indexOf(":") + 1, lineStr.indexOf("&"));
(44)            value =
(45)                    lineStr.substring(lineStr.lastIndexOf(":") + 1, lineStr.length());
(46)
(47)            hm.put(key, value);
(48)    }
(49)}
```

Download des Beispiels

Die Aufrufe erfolgen im HTTP-üblichen Stil:

- **GET**
  ```
  GET /HashMapApp HTTP/1.0
  key:name
  ```
- **POST**
  ```
  POST /HashMapApp HTTP/1.0

  key:name&value:mario
  ```
- **DELETE**
  ```
  DELETE /HashMapApp?key=name
  ```

Prinzipiell ist der Servletgedanke auch auf beliebige andere Protokolle und deren Operationen übertragbar. Jedoch wird er in der Praxis zumeist für HTTP eingesetzt. Die API ist jedoch generell so gestaltet, daß anwenderdefiniert jederzeit zusätzliche potokollspezifische Spezialisierungen der Klasse GenericServlet definiert werden können.

4.4 Web Services

Motivation und Hintergrund

Stand Datenaustausch in der Betrachtung der Metasprache XML bisher nahezu ausschließlich im Zeichen datei- oder strombasierter Integration, so eröffnet das Einsatzgebiet der *Web Services* den Einsatz XML-basierter Sprachen zur Abwicklung Internet-basierter online Kommunikation zwischen Anwendungssystemen.

Dem Begriff *Web Service* war in jüngerer Zeit eine bemerkenswerte Karriere beschieden, so daß es ihm gelang in der Praxis beachtliches Interesse auf sich zu ziehen. Allerdings fehlt bisher eine einheitliche Definition dieses Terminus hinsichtlich Zielsetzung und technischer Inhalte. Vielmehr versuchten und versuchen namhafte Herstelle durch eine Reihe verschiedener Definitionen, die am Markt offen zueinander in Konkurrenz treten, bisherige Techniken mit dem Ansatz der Web Services zu verschmelzen.
Jedoch führt(e) diese Vorgehensweise weder zu einer einheitlichen Begriffsübereinkunft und daher in der Folge auch zu keiner brauchbaren Abgrenzung gegenüber existierenden Techniken --- wie CORBA, RMI oder DCOM --- im Umfeld.
Nachfolgend wird daher eine an die Ergebnisse der Web Service Architecture-Arbeitsgruppe des World Wide Web Konsortiums angelehnte Definition zugrunde gelegt:

Definition 12: Web Service

Ein Web Service ist ein durch eine URI (gemäß RFC 2396) identifiziertes Softwaresystem, dessen öffentliche Schnittstellen und Protokollbindungen durch XML definiert und beschrieben sind. Diese Definitionen können durch andere Softwaresysteme ermittelt und bezogen werden. Auf der Basis der publizierten Schnittstellendefinitionen können diese Systeme dann mit dem Web Service in der durch die Schnittstelle festgelegten Weise interagieren. Diese Interaktion geschieht durch den Austausch XML-basierter Nachrichten, die mittels Internetprotokollen übertragen werden.

Der Begriff des *Service* ist hierbei durchaus in Anlehnung an den klassischen Dienstleistungsbegriff gemeint. Dieser beschreibt Handlungen, die im Auftrag eines Dritten vorgenommen werden, welche kein physisches Gut produzieren, sondern deren Charakter ausschließlich durch die Handlungserbringung selbst definiert ist.

Die Definition enthält ferner bereits Aussagen über die Komponenten einer Web Service basierten Architektur (einer sog. *serviceorientierten Architektur* (SOA)) und deren technischer Realisierung. Grundlegend Eigenschaften einer solchen Architektur ist neben der XML-Basiertheit die Verwendung von Universal Resource Identifkatoren zur eineindeutigen Benennung eines angebotenen Dienstes sowie die Nutzung der bestehenden Internetinfrastruktur zur Datenübertragung. Der Begriff *Internetinfrastruktur* soll hierbei nicht verengt auf die Techniken des WWW, d.h. HTTP auf Basis TCP/IP, verstanden werden, sondern durchaus im Sinne des der Internetprotokollhierarchie gebraucht werden.

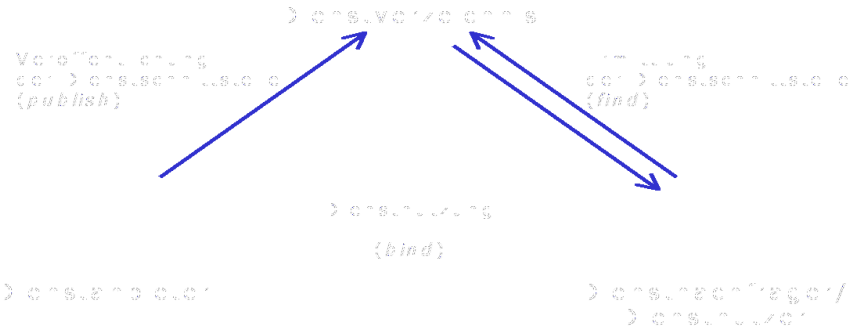
Gleichzeitig postuliert die Definition drei grundlegende Elemente einer serviceorientierten Architektur:

1. Ein XML-basierte Protokoll zur Darstellung und zum Austausch der Daten zwischen Diensten und ihren Nutzern.
2. Eine XML-basierte Beschreibungssprache zur Publikation von Dienstschnittstellen.
3. Einen Dienst zur Verwaltung der publizierten Schnittstellenbeschreibungen.

Diese drei aufeinander aufsetzenden Dienstschichten deuten auch bereits drei typische Rollen innerhalb einer serviceorientierten Architektur an:

1. Dienstanbieter.
   Er übernimmt die physische Bereitstellung des Dienstes, d.h. eine Implementierung und Publikation auf einem über das Internet zugänglichen Server.
   Zusätzlich kann er die Aufrufschnittstelle des Dienstes dokumentieren.
2. Dienstnachfrager/-nutzer.
   Er nutzt den angebotenen Dienst durch Übermittlung XML-codierter Nachrichten über Internetprotokolle.
   Zusätzlich kann er vor der Erstnutzung die Schnittstellenbeschreibung aus einem allgemein zugänglichen Dienstverzeichnis beziehen.
3. Dienstverzeichnis.
   Es verwaltet die durch die Dienstanbieter bereitgestellten kategorisierten Schnittstellenbeschreibungen.

Ausgehend von den Grundrollen und ihren Interaktionsbeziehungen ergeben sich die zwei in Abbildung 23 Abläufe für die Abwicklung einer Web Service-basierten Kommunikation.



Die Abbildung hebt die optionalen Schritte zur Ermittlung der Schnittstellenbeschreibung blau hervor. Liegt diese dem Aufrufwilligen bereits vor oder ist die Schnittstelle ihm bereits bekannt, so kann auf den Bezug der im Dienstverzeichnis abgelegten Beschreibungsdaten verzichtet werden und der Dienstaufruf direkt erfolgen.
Die Schnittstellenbeschreibung nimmt damit keine herausragende Rolle innerhalb der Architekturerstellung ein, wie dies beispielsweise für CORBA, RMI oder DCOM der Fall wäre (dort ist die Schnittstellenbeschreibung Teil des Entwicklungszyklus und muß zum Übersetzungzeitpunkt des Aufrufclients zwingend vorliegen).

Aus der Kombination der Architekturelemente und der Interaktionsszenarien haben sich drei Standards am Markt etabliert, deren Implementierungen die dargestellten Grundaufgaben innerhalb einer serviceorientierten Architektur erfüllen:

- SOAP --- Das Kommunikationsprotokoll zur Kommunikationsabwicklung zwischen Dienstanbieter und -nutzer.
- WSDL (Web Service Description Language) --- Die XML-basierte Schnittstellenbeschreibungssprache.
- UDDI (Universal Service Description, Discovery and Integration) --- Ein Verzeichnisdienst zur Verwaltung von Schnittstellenbeschreibungen, der selbst als Web Service realisiert ist.

Anmerkung: Ursprünglich wurde das Akronym *SOAP* zu *Simple Object Access Protocol* expandiert. Aufgrund der irreführenden Nähe zur objektorientierten Programmierung wurde jedoch im Verlauf des Standardisierungsprozesses beschlossen etablierte Akronym als Namen beizubehalten, jedoch von der weiteren Expansion abzusehen.

SOAP

SOAP, welches als Version 1.0 im Herbst 1999 als Ergebnis der Kooperation zwischen den Firmen *DevelopMentor*, *IBM*, *Microsoft*, *Lotus* und *UserLand Software* vorgestellt wurde markiert weniger den Beginn der Idee der Web Services, als vielmehr einen weiteren evolutionären Entwicklungsschritt.
Die Uridee die zur Abwicklung synchroner entfernter Funktionsaufrufe (*Remote Procedure Calls* (RPC)) zu übermittelnden Über- und Rückgabeparameter durch ein XML-Vokabular auszudrücken findet sich bereits im Protokoll *XML-RPC* verwirklicht.

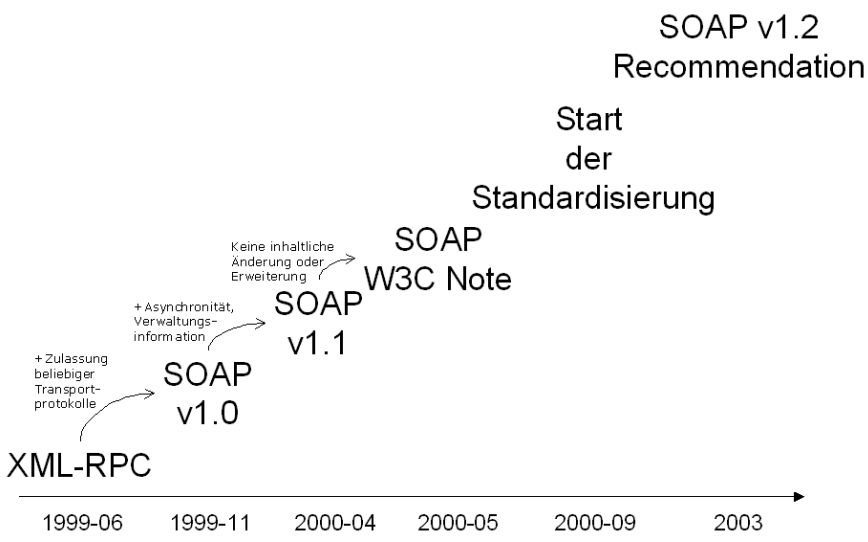
Ausgehend von diesem Ansatz definiert SOAP in der Version 1.0 ein vollständiges Protokoll, welches neben der Übertragung von Nutzdaten auch die Darstellung von dekriptiver Verwaltungsinformation vorsieht. Wie bereits für XML-RPC ist auch bei SOAP v1.0 ausschließlich die Verwendung von HTTP als Transportprotokoll definiert.

Diese Beschränkung wird durch die Anfang 2001 freigegebene SOAP Version 1.1 aufgehoben, welche SOAP als vollständig von der zu tatsächlichen Übertragung gewählten Protokollschicht entkoppelt und damit als eigenständige Protokollabstraktion etabliert. Gleichzeitig führt diese Version die Möglichkeit asynchroner Aufrufe ein.

Um den SOAP-Ansatz einer breiten Interessentenschicht zugänglich werden zu lassen und auch die Sensibilisierung für eine mögliche Standardisierung des Protokolls zu schaffen reichen die beteiligten Partner die Spezifikationsversion 1.1 unverändert als W3C Note ein.

Die im Herbst 2000 begonne Standardisierungsarbeit durch die W3C XML Protocol Arbeitsgruppe übernahm den eingereichten Spezifikationsvorschlag der Version 1.1 weitestgehend und führte, aus Gründen der Interoperabiltität behutsame Korrekturen sowie umfangreiche Erweiterungen ein. Mit der Verabschiedung des endgültigen W3C-Standards ist im Laufe des Jahres 2003 zu rechnen.

Die Abbildung 24 stellt nochmals die einzelnen SOAP-Versionen, ihre Unterschiede und den Vorläufer XML-RPC in der chronologischen Übersicht zusammen.



Aufbau einer SOAP-Nachricht

Jede SOAP-Nachricht, unabhängig davon ob es sich um eine synchron oder asynchron übermittelte Anfrage oder Antwort handelt besteht generell aus zwei Teilen:

1. Dem Rumpfelement (`Body`) zur Aufnahme der zu übermittelnden Nutzdaten.
2. Optional einem oder mehreren Kopfelementen (`Header`) zur Aufnahme von Metainformation.

Das Beispiel 105 zeigt einen vollständigen SOAP-Aufruf.
Er besteht aus einem *Umschlag* (dargestellt durch das Element `Envelope`), der das beschreibende Kopfelement `altertcontrol` sowie die im `Body`-Element zusammengefaßten Nutzdaten enthält.

Alle durch den Standard vorgegebenen Elemente und Attribute sind dem Namensraum `http://www.w3.org/2003/05/soap-envelope` zugeordnet und alle Anwenderdefinierten den entsprechenden eigenen Namensräumen.

Beispiel 105: Ein vollständiger SOAP-Aufruf

```
(1)<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
(2) <env:Header>
(3)  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
(4)   <n:priority>1</n:priority>
(5)   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
(6)  </n:alertcontrol>
(7) </env:Header>
(8) <env:Body>
(9)  <my:message xmlns:my="http://www.mycompany.com">
(10)   <my:text>Hello World!</my:text>
(11)  </my:message>
(12) </env:Body>
(13)</env:Envelope>
```

Die Aufgabe der *SOAP-Header* ist es Verwaltungsinformation aufzunehmen, die nicht ausschließlich durch den Empfänger des SOAP-Aufrufes zu verarbeiten ist und die nicht Teil der Nutzdaten ist.
Beispiele hierfür sind Daten über das Routingverhalten, gesetzte Sperren oder Transaktionen aber auch Zugriffsdaten wie Nutzername und Passwort (diese natürlich verschlüsselt!).

Die Verarbeitung eines Kopfelements ist vorgabegemäß optional, außer das zusätzliche gesetzte Attribut `mustUnderstand` weißt den Wert `1` oder `true` (die beiden durch XML Schema zugelassenen lexikalischen Repräsentationen für logisch *wahr*) auf. In diesem Falle muß ein *SOAP-Knoten* das Kopfelement verarbeiten. Kann die Verarbeitung nicht vorgenommen werden, so muß an den Sender der SOAP-Nachricht eine Fehlermeldung übermittelt werden.
Ein *SOAP-Knoten* ist hierbei jeder Rechner entlang des Nachrichtenpfades zwischen Sender und Empfänger, der das SOAP-Protokoll verarbeiten kann.

Soll die Verarbeitung der SOAP-Kopfelemente auf einen bestimmten Rechner (beispielsweise einen zwischenspeichernden Proxy-Knoten) beschränkt werden, so kann durch das im Standard vorhergesehene Attribut `role` eine eineindeutige Adressierung eines durch URI identifizierten (Zwischen-)Knotens erreicht werden.
Das Beispiel 106 zeigt die Nutzung des `role`- und des `mustUnderstand`-Attributs.

Beispiel 106: Nutzung der SOAP-Kopfelemente

```
(1)<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
(2)     <env:Header>
(3)     <abc:Extension1
(4)             xmlns:abc="http://example.org/2001/06/ext"
(5)             env:mustUnderstand="true"
(6)             role="http://www.example.com/xyz"/>
(7)     <def:Extension2 xmlns:def="http://example.com/stuff"
(8)             env:mustUnderstand="true"
(9)             env:role="http://www.w3.org/2003/05/soap-envelope/role/next"/>
(10)    </env:Header>
(11)    <env:Body>
(12)            ...
(13)    </env:Body>
(14)</env:Envelope>
```

Das Beispiel zeigt die Nutzung zweier Kopfelemente (`Extension1` und `Extension2`) in „eigenen“, d.h. nicht den im Standard vorgesehenen, Namensräumen.
Für beide Kopfelemente ist das Attribut `mustUnderstand` mit `true` belegt, was sie als zwingend zu prozessieren ausweist.
Zusätzlich ist für `Extension2` das Attribut `role` auf `http://www.example.com/xyz` gesetzt. Diese Belegung charakterisiert die durch `mustUnderstand` formulierte verpflichtende Verarbeitung näher. Im vorliegenden Falle muß sie ausschließlich durch den mit der URI `http://www.example.com/xyz` bezeichnenden Zwischenknoten erfolgen, für alle anderen Knoten des Nachrichtenpfades --- einschließlich des endgültigen Empfängers --- kann dieses Kopfelement ignoriert werden. Der dieses Kopfelement verarbeitende SOAP-Knoten darf es nach erfolgreicher Prozessierung aus der SOAP-Nachricht entfernen.
Bei der für das zweite Kopfelement (`Extension2`) gewählten Rolle (`http://www.w3.org/2003/05/soap-envelope/role/next`) handelt es sich um eine durch die SOAP-Spezifikation vordefinierte URI, welche alle Knoten entlang des Nachrichtenpfades einschließlich dem endgültigen Empfänger selbst bezeichnet. In diesem Fall darf das Kopfelement, selbst nach erfolgreicher Verarbeitung durch einen Knoten nicht entfernt werden, da weitere Verarbeitungsschritte durch andere Knoten folgen können. (Sofern es sich nicht um den endgültigen Empfänger handelt, müssen sogar weitere Verarbeitungsschritte folgen.)

Neben der dargestellten Sonderbelegung des Rollenattributes definiert der SOAP-Standard des W3C mit `http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver` eine Attributbelegung, die Kopfelemente kennzeichnet die ausschließlich durch den endgültigen Empfänger einer Nachricht verarbeitet werden dürfen und durch `http://www.w3.org/2003/05/soap-envelope/role/none` Kopfelemente, die durch einen SOAP-Knoten nicht verarbeitet werden dürfen.

Durch die Kopfelemente können Einzelknoten entlang des Vermittlungspfades einer SOAP-Nachricht pauschal oder gezielt zu einer anwenderdefinierten Verarbeitung angeregt werden. Die aktiven Zwischenknoten übernehmen hierbei nicht mehr nur reine vermittelnde Aufgaben auf den tieferliegenden (Transport-)Protokollschichten, sondern werden zu aktiven Elementen der SOAP-Nachrichtenkette. Aufgrund ihrer Stellung im Nachrichtenpfad zwischen Sender und endgültigem Empfänger werden sie auch mit dem Begriff *SOAP Intermediäre* belegt.

Die Übertragung der zwischen Dienstnutzer und Dienst ausgetauschten Daten (d.h. Aufruf- und Rückgabeparameter oder Einwegbotschaft) erfolgt durch das `Body`-Element der SOAP-Nachricht. Beispiel 107 zeigt den erzeugten SOAP-Datenstrom zum Aufruf eines Dienstes der die beiden `int`-Parameter *a* und *b* addiert und das Ergebnis zurückliefert.

Beispiel 107: Nutzung des SOAP-Rumpfelements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<env:Envelope
(3)     xmlns:env="http://www.w3.org/2003/05/soap-envelope"
(4)     xmlns:b="http://www.example.org/AddService>
(5)     <env:Body>
(6)             <b:add env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
(7)             <b:a>2</b:a>
(8)             <b:b>3</b:b>
(9)             </b:add>
(10)    </env:Body>
(11)</env:Envelope>
```

Innerhalb des `Body`-Elements werden die benannten Parameter durch jeweils ein Element, das den Namen des Parameters trägt, dargestellt. Jedes Parameterelement enthält die lexikalische Repräsentation des zu übertragenden Wertes. Der aufgerufene Dienst (im Beispiel: *add*) fungiert als gemeinsames Elternelement der einzelnen Parameterelemente.
Zur Unterscheidung der verschiedenen Vokabularelemente des SOAP-Aufrufes müssen alle anwenderdefinierten Teile des `Body`-Elements einem eigenen, vom SOAP-Vorgabenamensraum

abweichenden, Namensraum zugeordnet sein. Dies geschieht vor dem Hintergrund sowohl der gewünschten Abgrenzung von den durch den Standard vorgegebenen Elementen und Attributen, als auch mit der Zielsetzung zur Validierung der SOAP-Aufrufe eine XML-Schemainstanz einsetzen zu können. Hierzu sind die abweichenden Namensräume zwingend erforderlich, um mithilfe des any-Elements die Validierung der durch den SOAP-Standard strukturell nicht festlegbaren Body-Kindelemente zu ermöglichen.
Die Struktur dieser Kindelemente orientiert sich ausschließlich an der XML-Darstellung der für einen Service notwendigen Parameter und kann daher im allgemeinen Falle nicht durch den Standard vorgegeben werden. Aus diesem Grunde legt die SOAP-Spezifikation lediglich Encodierungsregeln zur Transformation hauptspeicherresidenter Objekte und Strukturen in eine XML-Darstellung fest.
Das Beispiel verwendet diese vordefinierten Encodierungsregeln, welche die Abbildung allgemeiner Objektgraphen in XML-Strukturen gestatten. Neben dieser, an der Belegung des Attributs encodingStyle mit `http://www.w3.org/2003/05/soap-encoding` erkennbaren, Serialisierungsform können durch den Anwender beliebige eigene Regeln zur Gewinnung der XML-Darstellung festgelegt werden. Einzig das im Standard definierte SOAP-Encoding muß jedoch zwingend von allen Implementierungen unterstützt werden.

In ähnlicher Darstellungsform der Anfrage findet sich auch die Übermittlung der durch den aufgerufenen Dienst berechneten Resultate codiert:

Beispiel 108: Nutzung des SOAP-Rumpfelements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<env:Envelope
(3)     xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
(4)     xmlns:b="http://www.example.org/AddService">
(5)     <env:Body>
(6)             <b:result env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">5</b:
result>
(7)     </env:Body>
(8)</env:Envelope>
```

Beispiel 108 zeigt die rückübermittelte Antwort des Addier-Dienstes.
Auch sie folgt den selben Struktur- und Erstellungsprinzipien. Daher weist auch Antwortnachricht die Trennung in (optionale und in diesem Beispiel nicht genutzte) Kopfelemente und nutzdatentragende Rumpfelemente als Kindelemente des Body-Elements auf.
Die Erstellung der XML-Darstellung erfolgt, wie bereits für den Aufruf, gemäß festgelegter bzw. anwenderdefiniert festlegbarer Richtlinien einer Encodierungsdarstellung. Auch in diesem Beispiel wurde die Standardencodierung gewählt, weshalb das encodingStyle-Attribut abermals mit dem Wert `http://www.w3.org/2003/05/soap-encoding` versehen ist.
Zur Abgrenzung von den durch den W3C-Standard vordefinierten Elementen muß auch zur Darstellung der Antwort eines Dienstaufrufs jedes Kindelement des Body-Knotens einen vom SOAP-Standard abweichenden Namensraum besitzen.
Die Organisation der Darstellung des Dienstergebnisses als XML-Dokument ist dem Diensterbringer überlassen. Im Beispiel wird ein Element `result` übertragen.

Quellcode des Web Service
Quellcode eines Web Serviceaufrufes

Fallstudie: Implementierung eines Web Services

Die nachfolgende Fallstudie betrachteten einen einfachen Web Service zur Realisierung der mathematischen Operationen auf dem Körper der komplexen Zahlen.
Die Diskussion wird an der frei verfügbaren SOAP-Implementierung Axis geführt, welche im Rahmen des Apache-Projekts gepflegt wird. Axis stellt eine vollständig in Java realisierte Bibliothek zur Umsetzung von Web Services die als Java-Klassen angeboten werden zur Verfügung.
Zur Ausführung wird eine beliebige Servlet-Umgebung benötigt. Für das Beispiel wurde die ebenfalls kostenfrei verfügbare Jakarta-Tomcat eingesetzt, welche als Ausführungsumgebung der Web Services dient.

Implementierung des Web Service:

Beispiel 109: Beispielwebdienst

```
(1)public class Complex {
(2)     private double re;
(3)     private double im;
(4)
(5)     public Complex(double re, double im) {
(6)             this.re = re;
(7)             this.im = im;
(8)     }
(9)     public Complex() {
(10)            this.re = 0;
(11)            this.im = 0;
(12)    }
(13)
(14)    public void setRe(double re) {
(15)            this.re = re;
(16)    }
(17)    public void setIm(double im) {
(18)            this.im = im;
(19)    }
(20)    public double getIm() {
(21)            return im;
(22)    }
(23)    public double getRe() {
(24)            return re;
(25)    }
(26)    public double modulus(Complex c1) {
(27)            return Math.sqrt(Math.pow(c1.getRe(), 2) + Math.pow(c1.getIm(), 2));
(28)    }
(29)    public Complex add(Complex c1, Complex c2) {
(30)            return new Complex(
(31)                    c1.getRe() + c2.getRe(),
(32)                    c1.getIm() + c2.getIm());
(33)    }
(34)    public Complex mult(Complex c1, Complex c2) {
(35)            return new Complex(
(36)                    c1.getRe() * c2.getRe() - c1.getIm() * c2.getIm(),
(37)                    c1.getRe() * c2.getIm() + c1.getIm() * c2.getRe());
(38)    }
(39)    public Complex div(Complex c1, Complex c2) {
(40)            double div = Math.pow(c2.getRe(), 2) + Math.pow(c2.getIm(), 2);
(41)            return new Complex(
(42)                    (c1.getRe() * c2.getRe() + c1.getIm() * c2.getIm()) / div,
(43)                    (c1.getIm() * c2.getRe() - c1.getRe() * c2.getIm()) / div);
(44)    }
(45)    public Complex pow(Complex c1, int n){
(46)            if (n==0) {
(47)                    return new Complex(1,0);
(48)            }
(49)            Complex result = c1;
(50)            for (int i=1;i<n;i++){
(51)                    result = result.mult(c1, c1);
(52)            }
(53)            return result;
(54)    }
(55)    public String toString() {
(56)            String result = new String();
(57)            result += re;
(58)            if (im < 0) {
(59)                    result += "-";
(60)            } else {
(61)                    result += "+";
(62)            }
(63)            result += "i" + Math.abs(im);
(64)            return result;
(65)    }
(66)}
```

Beispiel 109 zeigt die Implementierung der Klasse `Complex` deren Methoden als Web Service angeboten werden.
Augenscheinlich unterschiedet sich die Implementierung als Web Service nicht von der, die für die statische lokale Bereitstellung leistenden.
Bei der Umsetzung ist jedoch darauf zu achten, auf `this`-Verweise in Methodenrümpfen zu verzichten, da kein Objektkontext durch das SOAP-Protokoll mitversandt wird. Als pragmatisches Hilfsmittel zur

Einhaltung dieser Einschränkung bietet es sich an die als Dienst bereitzustellenden Methoden mit dem Schlüsselwort `static` zu versehen.
Zusätzlich ist auf die Definition eines parameterlosen Vorgabekonstruktors zu achten, da dieser serverseitig zur durch das Framework zur Objekterzeugung herangezogen wird.

Die serverseitige Bereitstellung des Dienstes kann im Rahmen des Axis-Frameworks auf zwei Arten geschehen. Zum einen durch Bereitstellung des Quellcodes aus Beispiel 109 in einem dafür gesondert vorgesehen Serververzeichnis (typischerweise `../tomcat/webapps/axis`). Zusätzlich ist die Datei mit der Extension `jws` (statt dem üblichen `java`) zu versehen. Zum Aufrufzeitpunkt wird der Quellcode durch das Axis-Framework für den Aufrufer transparent übersetzt und zur Ausführung gebracht.
Andererseits sieht das Framework die Möglichkeit der Ablage von vorübersetztem Java-Code vor. Dieser in der Axis-Terminologie als *Web Service Deployment* bezeichnete Vorgang gestattet dem Anwender weiterrechenden Einfluß auf Umstände der Dienstbereitstellung als die zuvor gezeigte Vorgehensweise.

Notwendige Voraussetzung des Deploymentvorganges ist die Bereitstellung einer separaten Konfigurationsdatei, dem sog. *Deployment-Deskriptor* unter zwingend vorgegebenen dem Dateinamen `deploy.wsdd`. Beispiel 110 zeigt eine solche Konfigurationsdatei für den Beispieldienst.

Beispiel 110: Deploymentdeskriptor des Beispieldienstes

```
(1)<deployment
(2)     xmlns="http://xml.apache.org/axis/wsdd/"
(3)     xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
(4)   <service name="ComplexCalc" style="RPC">
(5)     <parameter name="className" value="Complex"/>
(6)      <parameter name="allowedMethods" value="add mult modulus div pow"/>
(7)      <beanMapping
(8)             qname="myNS:Complex"
(9)             xmlns:myNS="urn:Complex"
(10)            languageSpecificType="java:Complex"/>
(11)    </service>
(12)</deployment>
```

Innerhalb der Konfigurationsdatei werden Aussagen über das Interaktionsmuster, welches der Kommunikation mit dem Dienst zugrunde liegt. Im Beispiel wird durch die Belegung des Attributs `style` mit dem Wert `RPC` synchrone Kommunikation im Stile entfernter Methodenaufrufe gewählt.
Zusätzlich kann ein vom Namen des Compilats abweichender Dienstname frei festgelegt werden (im Beispiel `ComplexCalc` als Belegung des Attributs `name` des `service`-Elements).
Innerhalb der Kindelemente des Elements `service` wird die dienstimplementierende Klasse identifiziert (`parameter`-Element dessen `name`-Attribut den Wert `className` trägt, das zugehörige `value`-Attribut desselben Elements enthält dann den vollqualifizierten Klassennamen) sowie die per SOAP zugreifbaren Methoden mit separierenden Leerzeichen gemäß XML-Konvention aufgelistet (im Beispiel `add mult modulus div pow`).
Hierdurch eröffnet die Konfiguration des Webdienstes durch den Deploymentdeskriptor bereits einen Eingriffspunkt, der bei der reinen serverseitigen Ablage des Quellcodes, mit automatischer Freigabe aller öffentlich zugänglichen Methoden, nicht bestand.
Ferner können vermöge des Elements `beanMapping` anwenderdefinierte strukturierte Datentypen, die als Java-Bean-konforme Klasse umgesetzt sind definiert werden. Im Beispiel ist dies die Klasse zur Implementierung der komplexen Zahl.

Die Installation und Konfiguration (Deployment) des Web Service geschieht durch Aufruf des Administrationswerkeuges `java org.apache.axis.client.AdminClient deploy.wsdd`.
Zusätzlich ist die Dienstklasse im Verzeichnis `.../tomcat/webapps/axis/WEB-INF/classes` abzulegen.

Implementierung des Web Service Aufrufers:

Beispiel 111: Aufruf des Beispielwebdiensts

```
(1)import java.rmi.RemoteException;
(2)import javax.xml.namespace.QName;
(3)import javax.xml.rpc.ParameterMode;
(4)import javax.xml.rpc.encoding.XMLType;
(5)import org.apache.axis.client.Call;
(6)import org.apache.axis.client.Service;
(7)import org.apache.axis.encoding.ser.BeanSerializerFactory;
(8)import org.apache.axis.encoding.ser.BeanDeserializerFactory;
(9)
(10)public class callService {
(11)    public static void main(String[] args) {
(12)            Call call = new Call( new Service() );
(13)
(14)            QName qn = new QName("urn:Complex","Complex");
(15)
(16)            call.setTargetEndpointAddress("http://10.0.0.1:8080/axis/services/
```

```
ComplexCalc");
(17)            call.setOperationName("pow");
(18)            call.registerTypeMapping(Complex.class, qn,
(19)                                     new BeanSerializerFactory(Complex.class, qn),
(20)                                     new BeanDeserializerFactory(Complex.class, qn));
(21)
(22)            call.addParameter("c1", qn, ParameterMode.IN);
(23)            call.addParameter("n", XMLType.XSD_INT, ParameterMode.IN);
(24)            Object[] param = new Object[2];
(25)            param[0] = new Complex(2,3);
(26)            param[1] = new Integer(3);
(27)
(28)            call.setReturnType( qn );
(29)
(30)            try {
(31)                    Complex result = (Complex) call.invoke(param);
(32)                    System.out.println("result=" + result);
(33)            } catch (RemoteException re) {
(34)                    re.printStackTrace();
(35)            }
(36)    }
(37)}
```

Beispiel 111 zeigt die Implementierung eines Aufrufs des Beispielwebdienstes.
Zentrales Objekt des Aufrufs eines Web Service ist die Klasse `Call`. Sie kapselt die gesamten zur Kommunikation notwendigen Daten und übernimmt den synchronen Aufruf des entfernten Dienstes. Zur Verwaltung von Verbindungsdaten, die für verschieden Einzelaufrufe desselben Dienstes gleichermaßen Verwendung finden können diese zusätzlich durch einen `Service` repräsentiert werden. Im Beispiel wird hierauf jedoch aus Übersichtlichkeitsgründen verzichtet und alle alle Einstellungen direkt für das `Call`-Objekt vorgenommen.
Zu diesen Einstellungen zählt die dargestellte Angabe des *Service Endpunktes*, derjenigen Adresse, die konform zum gewählten Übertragungsprotkoll (im Beispiel ist dies HTTP) die physische Übergabestelle des SOAP-Aufrufs an den Web Service identifiziert.
Daher ist bei Änderung der Dienstbereitstellung, etwa durch Verlagerung auf einen anderen Server oder auch die sich aus dem oben gezeigten geänderten Deployment ergebende Endpunktadresse, lediglich der Parameter der Methode `setTargetEndpointAddress` geeignet anzupassen.

Zusätzlich zur Angabe der aufzurufenden Operation innerhalb des angesprochenen Dienstes, durch die Methode `setOperationName` erfolgt durch `addParameter` die Definition der Übergabeparameter des Dienstes.
Hierbei muß jeder Übergabeparameter in Name, Typ und Übergabeart spezifiziert werden. Im Beispiel sind dies die beiden Parameter `c1` und `n`, die als Eingabe (d.h. nur zum lesenden Zugriff) der Methode `pow` dienen.
Diese Methode implementiert die Potenzierung komplexer Zahlen durch fortwährende Multiplikation. Ihre Signatur erfordert daher die Übergabe der komplexen Basis sowie des ganzzahligen Exponenten.

Zur Übertragung der speicherresidenten Wertdarstellung ordnet das Axis-Framework den Java-Primtivdatentypen und einigen ausgewählten als Klasse realisierten Datentypen eindeutige Abbildungen in die in XML-Schema definierten Datentypen zu.
Die Reihenfolge der Aufrufe der `addParameter`-Methode muß der Auftrittsreihenfolge in der Signatur des aufzurufenden Dienstes entsprechen, um die Kompatibilität zur Programmiersprachen, die ausschließlich über Stellungsparameter verfügen zu wahren.
Tabelle 27 stellt die durch das Axis-Framework Typäquivalenzen zusammen:

Tabelle 27: Abbildung der Java-Datentypen auf XML-Schema

Java	XML-Schema	Bemerkung
byte[]	base64Binary hexBinary	Beide Darstellungsweisen sind gleichwertig möglich.
boolean	boolean	
byte	byte	
java.util.Calendar	dateTime	
java.math.BigDecimal	decimal	
double	double	
float	float	
int	int	
java.math.BigInteger	integer	
long	long	
javax.xml.namespace.QName	QName	Diese Javaklasse ist Bestandteil der API-Erweiterung für XML, die separat bezogen werden muß.

short	short
java.lang.String	string

Bei der zu übergebenden komplexen Zahl handelt es sich um eine anwenderdefinierte Klasse, die einen neuen Java-Typ etabliert für den (naturgemäß) keine Entsprechung in XML-Schema zur Verfügung steht. Aus diesem Grund muß auf Seiten des Aufrufers und des aufgerufenen Dienstes eine eigene Abbildungsvorschrift für die Erzeugung der SOAP-Darstellung bereitgestellt werden.

Durch die Axis-Serviceausführungsumgebung kann automatisiert eine solche Abbildung erzeugt werden, sofern der anwenderdefinierte Typ (d.h. die implementierende Klasse) gemäß der Java-Beans-Programmierkonvention umgesetzt ist. Diese ist eingehalten, sobald für jedes datenhaltende Attribut eine get- und set-Methode umgesetzt wird. Hintergrund dieses Vorgehens ist die Möglichkeit der Abfrage aller objektinternen Datenfelder durch Nutzung der Java-Reflection-API.
Die hierfür notwendige Zuordnung zwischen neuem Java- und XML-Typ geschieht im durch Beispiel 110 dargestellten Deploymentdeskriptor ab Zeile sieben. Dort gibt das Attribut `qname` den Namen des XML-Typs und `languageSpecificType` den Namen der implementierenden Java-Klasse (`Complex`) an.

Entsprechend vollzieht der Dienstaufrufer die Typabbildung im Code nach. Hierzu muß ihm der gewählte Name des neuen XML-Typen bekannt sein (im Beispiel: `Complex` im Namensraum `urn:Complex`). Durch den Aufruf der Methode `registerTypeMapping` gibt er dem Framework die Implementierungen der beiden Abbildungsrichtungen (Java-Hauptspeicherobjekte in SOAP bzw. SOAP zu Hauptspeicherobjekten) bekannt. Das Beispiel verwendet hierzu die im Axis-Framework mitgelieferten Klassen `BeanSerializerFactory` und `BeanDeserializerFactory` welche das Gewünschte auf Basis einer als Java-Bean codierten Klasse leisten.

Die Festlegung der tatsächlichen Übergabewerte erfolgt im Beispiel durch Erzeugung eines Arrays zur Aufnahme von `Object`-typisierten Elementen in die die zuvor hinsichtlich ihres Typs definierten Aktualparameter eingefügt werden.
Aufgrund der Einschränkung der Programmiersprache Java (bis zur Version 1.4), die Ausprägungen von Primitivtypen nicht als Objekte behandeln kann, erfolgt im Beispiel die Kapselung des `int`-Wertes für `n` durch ein Objekt der Klasse `Integer`.

Für die Definition des Typs des erwarteten Rückgabewertes gelten dieselben Gesetztmäßigkeiten hinsichtlich Bekanntmachung und Zugriffsabwicklung. Die Definition des erwarteten Rückgabetyps erfolgt durch Aufruf der Methode `setReturnType`.

Der synchrone Aufruf des entfernten Dienstes wird dann durch die Methode `invoke` des erzeugten und entsprechend parametrisierten `Call`-Objektes vollzogen.
Dieser Aufruf ist blockierend und läßt den Aufrufer bis zum Eintreffen des berechneten Ergebnisses warten.

Die Rückgabe wird, trotz der Definition des spezielleren Rückgabetyps `Complex` generell als `Object`-Ausprägung geliefert um eine strikt typprüfbare Signatur der Methode `invoke` zu erhalten. Daher muß das durch den Web Service gelieferte Resultat geeignet, d.h. konform zur durch `setReturnType` getroffenen Festlegung, typgewandelt werden.

Beispiel 112 zeigt den SOAP-Datenverkehr beim Aufruf des Dienstes zur Berechnung der Potenz einer komplexen Zahl. Aufgrund des derzeitigen Entwicklungsstandes des Axis-Frameworks weicht der Leitungsmittschnitt von der zu Eingang dieses Kapitels vorgestellten SOAP-Struktur des W3C-Standards ab. Gegenwärtig (d.h. zum Zeitpunkt der Verfügbarkeit der Version 1.1) unterstützt das Framework nur eine Untermenge des neuen Standards und codiert die XML-Nachrichten noch nach dem Stand der Vorgängerspezifikation.

Beispiel 112: SOAP-Nachricht zum Aufruf des Beispieldienstes

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<soapenv:Envelope
(3)     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
(4)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(5)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
(6)     <soapenv:Body>
(7)             <pow soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
(8)             <c1 href="#id0"/>
(9)             <n xsi:type="xsd:int">3</n>
(10)            </pow>
(11)            <multiRef
(12)                    id="id0"
(13)                    soapenc:root="0"
(14)                    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
(15)                    xsi:type="ns1:Complex"
(16)                    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
(17)                    xmlns:ns1="urn:Complex">
(18)            <im xsi:type="xsd:double">3.0</im>
(19)            <re xsi:type="xsd:double">2.0</re>
```

```
(20)            </multiRef>
(21)    </soapenv:Body>
(22)</soapenv:Envelope>
```

Im Code des Beispiels gut zu sehen ist die Darstellung der beiden Übergabeparameter `c1` und `n` für die Basis der Potenzierung und den ganzzahligen Exponenten. Für `n` wird die in Tabelle 112 Abbildung in den äquivalenten Typen `int` aus XML-Schema durchgeführt. Zusätzlich überträgt die durch Axis gewählte Codierung die Typisierung explizit mit (Attribut `xsi:type="xsd:int"`) wodurch die Typabbildung offenbar wird.
Für den nicht direkt nach XML-Schema transformierbaren strukturierten (Java-)Datentypen `Complex` wird durch den Serialisierungsmechanismus ein eigenes mit `multiRef` bezeichnetes und durch das Attribut `id` identifiziertes Element erzeugt. Dieses Vorgehen entspricht der durch die SOAP-Spezifikation vorgesehenen Serialisierungs allgemeiner Graphen, welche die speicherresidenten Datenobjekte als Knoten interpretiert und hierfür wiederverwendbare (der Name des Elements deutet dies an) Elemente erzeugt. Die tatsächliche Wiederverwendung innerhalb des SOAP-Stromes findet durch Mehrfachnennung des im `id`-Attribut festgelegten identifizierenden Wertes im `href`-Attribut eines Verweiselementes statt.
Im Beispiel wird die Berechnungsbasis als ein solches `multiRef`-Element ausgedrückt, welches die Wertbelegungen der durch get-Methoden zugänglichen Javafelder (`im` und `re`) enthält.
Das den Übergabeparameter `c1` repräsentierende XML-Element enthält daher lediglich im `href`-Attribut einen Verweis auf die `multiRef`-Struktur.

Beispiel 113 zeigt den für die Übermittlung des Dienstergebnisses übertragenen SOAP-Datenverkehr. Auch er weicht, aufgrund der Konformität zur älteren SOAP-Spezifikationsversion, geringfügig vom aktuellen Standard ab.

Beispiel 113: SOAP-Nachricht zur Übermittlung des Berechnungsergebnisses des Beispieldienstes

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<soapenv:Envelope
(3)     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
(4)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(5)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
(6)     <soapenv:Body>
(7)             <powResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/">
(8)     <powReturn href="#id0"/>
(9)     </powResponse>
(10)    <multiRef
(11)            id="id0"
(12)            soapenc:root="0"
(13)            soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
(14)            xsi:type="ns1:Complex"
(15)            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
(16)            xmlns:ns1="urn:Complex">
(17)    <im xsi:type="xsd:double">12.0</im>
(18)    <re xsi:type="xsd:double">-5.0</re>
(19)    </multiRef>
(20)    </soapenv:Body>
(21)</soapenv:Envelope>
```

Wie bereits beim Aufruf realisiert wird die XML-Darstellung des strukturierten Datentyps `Complex`, der hier als Rückgabetyp dient, durch Nutzung der allgemeinen Graphenserialisierung erzeugt.

WSDL

Die Grundidee der Web Services fordert nicht zwingend die Darstellung der Dienstschnittstellen für Dritte, wie dies beispielsweise Verteilungsarchitekturen wie CORBA und DCOM zwingend als Teil des Entwicklungszyklus vorgeben. Dennoch erweist es sich auch für Web Services sinnvoll die Schnittstellenbeschreibungen in einem maschinenlesbaren Format bereitzustellen um Werkzeugen die automatisierte Verarbeitung zu ermöglichen. Hierunter fallen beispielsweise Entwicklungsumgebungen, die aus Schnittstellenbeschreibungen erste Rohgerüste für die Abwicklung der Dienstaufrufe generieren können. Zusätzlich stellen formalisiert dokumentierte Schnittstellen einen wichtigen Bestandteil der Dokumentation eines Softwaresystems dar.
Zur Schnittstellendokumentation von Web Services etabliert sich gegenwärtig der Standard der *Web Service Description Language*, der durch eine Arbeitsgruppe des World Wide Web Konsortiums vorangetrieben wird.

Eine WSDL-Beschreibung selbst ist eine XML-Datei, WSDL mithin als XML-Schema-basiertes XML-Vokabular realisiert und zerfällt in sechs Teile:

1. **Service**: Zur allgemeinen Beschreibung des angebotenen Dienstes
2. **Operations**: Zur Beschreibung der angebotenen Operationen
3. **Messages**: Zur Beschreibung Signatur der einzelnen Nachrichten (Aufrufe und Antworten)
4. **PortTypes**: Zur Verknüpfung von *Operations* und *Messages*
5. **Bindings**: Zur Verknüpfung einer *Operation* mit einem Transportprotkoll
6. **Types**: Zur Definition anwenderdefinierter Übergabe- und Rückgabetypen

Die Angaben zur Struktur der anwenderdefinierten Typen ist optional und muß nur im Falle der Existenz solcher Typen erfolgen.

Das Beispiel 114 zeigt die vollständige WSDL-Dienstbeschreibung des aus der Fallstudie bekannten Dienstes:

Beispiel 114: WSDL-Beschreibung des Beispieldienstes

```
(1)<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.
apache.org/xml-soap" xmlns:impl="http://10.0.0.1:8080/axis/services/Complex" xmlns:
intf="http://10.0.0.1:8080/axis/services/Complex" xmlns:soapenc="http://schemas.xmlsoap.
org/soap/encoding/" xmlns:tns1="urn:Complex" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/
XMLSchema" targetNamespace="http://10.0.0.1:8080/axis/services/Complex">
(2)     <wsdl:types>
(3)             <schema targetNamespace="urn:Complex" xmlns="http://www.w3.org/2001/
XMLSchema">
(4)                     <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
(5)                     <complexType name="Complex">
(6)                             <sequence>
(7)                                     <element name="im" type="xsd:double"/>
(8)                                     <element name="re" type="xsd:double"/>
(9)                             </sequence>
(10)                    </complexType>
(11)            </schema>
(12)    </wsdl:types>
(13)    <wsdl:message name="addResponse">
(14)            <wsdl:part name="addReturn" type="tns1:Complex"/>
(15)    </wsdl:message>
(16)    <wsdl:message name="divRequest">
(17)            <wsdl:part name="c1" type="tns1:Complex"/>
(18)            <wsdl:part name="c2" type="tns1:Complex"/>
(19)    </wsdl:message>
(20)    <wsdl:message name="modulusRequest">
(21)            <wsdl:part name="c1" type="tns1:Complex"/>
(22)    </wsdl:message>
(23)    <wsdl:message name="divResponse">
(24)            <wsdl:part name="divReturn" type="tns1:Complex"/>
(25)    </wsdl:message>
(26)    <wsdl:message name="multResponse">
(27)            <wsdl:part name="multReturn" type="tns1:Complex"/>
(28)    </wsdl:message>
(29)    <wsdl:message name="powResponse">
(30)            <wsdl:part name="powReturn" type="tns1:Complex"/>
(31)    </wsdl:message>
(32)    <wsdl:message name="addRequest">
(33)            <wsdl:part name="c1" type="tns1:Complex"/>
(34)            <wsdl:part name="c2" type="tns1:Complex"/>
(35)    </wsdl:message>
(36)    <wsdl:message name="modulusResponse">
(37)            <wsdl:part name="modulusReturn" type="xsd:double"/>
(38)    </wsdl:message>
(39)    <wsdl:message name="powRequest">
(40)            <wsdl:part name="c1" type="tns1:Complex"/>
(41)            <wsdl:part name="n" type="xsd:int"/>
(42)    </wsdl:message>
(43)    <wsdl:message name="multRequest">
(44)            <wsdl:part name="c1" type="tns1:Complex"/>
(45)            <wsdl:part name="c2" type="tns1:Complex"/>
(46)    </wsdl:message>
(47)    <wsdl:portType name="Complex">
(48)            <wsdl:operation name="pow" parameterOrder="c1 n">
(49)                    <wsdl:input name="powRequest" message="impl:powRequest"/>
(50)                    <wsdl:output name="powResponse" message="impl:powResponse"/>
(51)            </wsdl:operation>
(52)            <wsdl:operation name="add" parameterOrder="c1 c2">
(53)                    <wsdl:input name="addRequest" message="impl:addRequest"/>
(54)                    <wsdl:output name="addResponse" message="impl:addResponse"/>
(55)            </wsdl:operation>
(56)            <wsdl:operation name="mult" parameterOrder="c1 c2">
```

```
(57)                    <wsdl:input name="multRequest" message="impl:multRequest"/>
(58)                    <wsdl:output name="multResponse" message="impl:multResponse"/>
(59)            </wsdl:operation>
(60)            <wsdl:operation name="modulus" parameterOrder="c1">
(61)                    <wsdl:input name="modulusRequest" message="impl:modulusRequest"/>
(62)                    <wsdl:output name="modulusResponse" message="impl:modulusResponse"/
>
(63)            </wsdl:operation>
(64)            <wsdl:operation name="div" parameterOrder="c1 c2">
(65)                    <wsdl:input name="divRequest" message="impl:divRequest"/>
(66)                    <wsdl:output name="divResponse" message="impl:divResponse"/>
(67)            </wsdl:operation>
(68)    </wsdl:portType>
(69)    <wsdl:binding name="ComplexSoapBinding" type="impl:Complex">
(70)            <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/
http"/>
(71)            <wsdl:operation name="pow">
(72)                    <wsdlsoap:operation/>
(73)                    <wsdl:input>
(74)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(75)                    </wsdl:input>
(76)                    <wsdl:output>
(77)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(78)                    </wsdl:output>
(79)            </wsdl:operation>
(80)            <wsdl:operation name="add">
(81)                    <wsdlsoap:operation/>
(82)                    <wsdl:input>
(83)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(84)                    </wsdl:input>
(85)                    <wsdl:output>
(86)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(87)                    </wsdl:output>
(88)            </wsdl:operation>
(89)            <wsdl:operation name="mult">
(90)                    <wsdlsoap:operation/>
(91)                    <wsdl:input>
(92)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(93)                    </wsdl:input>
(94)                    <wsdl:output>
(95)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(96)                    </wsdl:output>
(97)            </wsdl:operation>
(98)            <wsdl:operation name="modulus">
(99)                    <wsdlsoap:operation/>
(100)                   <wsdl:input>
(101)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(102)                   </wsdl:input>
(103)                   <wsdl:output>
(104)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(105)                   </wsdl:output>
(106)           </wsdl:operation>
(107)           <wsdl:operation name="div">
(108)                   <wsdlsoap:operation/>
(109)                   <wsdl:input>
(110)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(111)                   </wsdl:input>
(112)                   <wsdl:output>
(113)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.
xmlsoap.org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(114)                   </wsdl:output>
(115)           </wsdl:operation>
(116)   </wsdl:binding>
(117)   <wsdl:service name="ComplexService">
(118)           <wsdl:port name="Complex" binding="impl:ComplexSoapBinding">
(119)                   <wsdlsoap:address location="http://10.0.0.1:8080/axis/services/
Complex"/>
```

```
(120)          </wsdl:port>
(121)   </wsdl:service>
(122)</wsdl:definitions>
```

Das Beispiel spiegelt im **Service**-Element die bereits durch den Deploymentdeskriptor getroffenen Festlegungen wieder und macht sie so dem (potentiellen) Dienstnutzer zugänglich. So enthält das Attribut `name` mit dem Wert `ComplexService` den gewählten Namen des Dienstes. Zusätzlich der als Kindelement realisierte `address`-Eintrag unter `location` die Adresse des SOAP-Endpunktes wieder welche SOAP-Botschaften zum Dienstaufruf entgegennimmt.
Da derselbe Dienst durch verschiedene Endpunkte zur Verfügung gestellt werden kann, ist das mehrfache Auftreten von `address`-Element zugelassen. Aus Gründen der Eindeutigkeit und Zuordnung des Endpunktes zum jeweils unterstützten Transportprotokoll ist jedes `address`-Element durch ein `port`-Element umschlossen welches auf das definierte Protokoll verweist.

Jedes **Operation**-Element definiert die zum Aufruf zu übermittelnden Parameter hinsichtlich Reihenfolge und referenziert die eine Interaktion konstituierenden Interaktionsschritte.
Im Beispiel legt die Operation `pow` (die Benennung wird durch das `name`-Attribut festgelegt) fest, daß die beiden Parameter `c1` und `n` benötigt werden. Zusätzlich erfolgt durch das Element `input` eine Referenz auf die Eingabenachricht bzw. `output` auf die Ausgabenachricht die zur Initiierung der Operation bzw. nach deren Ende versandt werden.

Innerhalb des **Message**-Elements werden die Übergabe Parameter inhaltlich hinsichtlich Typ und Name ausdefiniert.
Für die Nachricht `powRequest` (Aufruf der Methode `pow`) werden daher `c1` vom Typ eigendefinierten Typ `Complex` sowie `n` vom Standardtyp `int` genannt.

Durch das WSDL-Element `PortType` erfolgt die Aggregation der zuvor definierten Operationen, deren Nachrichten, an den in der Service-Definition genannten Endpunkt.

Die technischen Details wie gewähltes Encodierungsschema und genutzter Namensraum werden im Rahmen des **Binding**-Elements für jeden Nichtstandardtypen --- im Beispiel ist dies `Complex` --- definiert.

Die zur Erzeugung der über die Netzwerkleitung zu transportierenden XML-Darstellung dieser Nichtstandardtypen wird durch ein XML-Schemafragment beschrieben, welches als Kindelement des WSDL-Elements **Types** abgelegt ist.

UDDI

Zwar bietet WSDL eine wertvolle Möglichkeit zur Beschreibung der technischen Schnittstellencharakteristika eines Web Service, jedoch bleiben andere Fragestellungen --- etwa die nach der Natur und menschenlesbaren Beschreibung eines Dienstes oder seinen Nutzungs- und Abrechnungsbedingungen --- durch diesen Standard vollkommen offen. Überdies enthält das offizielle Spezifikationsdokument keinerlei Aussagen über einen präferierten oder auch nur sinnvollen Bereitstellungsort der WSDL-Beschreibungen.
Einen Ansatz zur Antwort auf diese Fragestellungen versucht der im Rahmen des OASIS-Standardisierungsprozeß vorangetriebene Verzeichnisdienst *Universal Service Description, Discovery and Integration* zu liefern.

Dieses, selbst als SOAP-basierter Web-Dienst angebotene, Verzeichnisdienst stellt eine Verwaltungsstruktur zur Ablage von WSDL-Beschreibungen und anderer dienstbezogener deskriptiver Metadaten bereit die durch eine definierte Schnittstelle abgefragt werden kann.

Die Grundprimitive des UDDI-Dienstes sind:

- **Business Entity**: Der Anbieter eines Web Service
- **Business Service**: Der angebotene Dienst
- **tModel**: Die Beschreibung des Dienstes
- **BindingTemplate**: Verknüpfung von tModel und Business Service

Ziel der **Business Entity**-Struktur ist es telephonbuchartig beschreibende Metadaten zum Dienstanbieter, wie Firmenname und Erreichbarkeitsdaten in einer alphabetisch sortierten Struktur anzubieten.

Jedem Business Entity-Eintrag können mehrere **Business Services** zugeordnet sein, welche die angebotenen Web Services repräsentieren.

Jedes **tModel** (Abkürzung für *technical model*) kann eine WSDL-Beschreibung oder beliebige durch den Anwender festlegbare dienstbezogene deskriptive Inhalte aufnehmen.
Insbesondere kann ein tModel auch Kategorisierungen eines Dienstes, etwa die Zuordnung zu einer Dienstfamilie, die Herstellung eines bestimmten Typ-Kontexts wie Erbringungsort des Dienstes, aufnehmen.

Alle tModels eines Dienstes werden mit diesem durch **BindingTemplate**-Elemente verbunden.

Die Interaktion mit einem UDDI-Verzeichnisdienst kann SOAP-basiert oder durch manuelle Erfassung der abzulegenden Daten über eine Webseite erfolgen.
Ziel dieser Interaktion ist zumeist einer der global angebotenen UDDI-Verzeichnisdienste, die gegenwärtig innerhalb eines Tages für die vollständige Inhaltsreplizierung sorgen.

Aufgrund immernoch offener Sicherheitsfragestellungen und der als ungelöst anzusehenden Abrechnungsproblematik liegen jedoch in den aktuell zugänglichen UDDI-Diensten kaum kommerzielle Dienstangebote vor, sondern überwiegend Testeinträge.

Web-Referenzen 13: Weiterführende Links

- SOAP-Standard des W3C
  - Primer
  - Messaging Framework
  - Adjuncts
- Spezifikation der Web Service Description Language
- UDDI-Spezifikation
- Axis-Framework Eine Open-Source SOAP-Implementierung
- Aktuelle Informationen rund um Web Services

---

5 Präsentationsaspekte

TODO(Hinweis: Dieses Kapitel fehlt noch)

5.1 XHTML und XForms

TODO(Hinweis: Dieses Kapitel fehlt noch)

5.2 Java Server Pages (JSP)

Beim Einsatz von Servlets ist oftmals das Verhältnis zwischen berechneten Anteilen und statisch produzierten Ausgaben deutlich zu Gunsten der statischen Anteile verschoben. Gleichzeitig ist jedoch die Ausgabe statischen HTML- oder XML-Codes vergleichsweise mühsam und schreibaufwendig, wie Beispiel 102 zeigt.
Zur Abhilfe dies vermeidbaren Aufwende und damit zur Unterstützung von Applikationstypen in denen die dynamisch festzulegenden Anteile einer Ausgabe deutlich gegenüber den statischen zurückstehen sieht Sammlung der J2EE-Applikationstypen die *Java Server Pages* (JSP) vor.

Konzeptionell stellt eine Server Page eine --- gegenüber klassischem HTML/XML nahezu unveränderte --- Ausgabeseite dar, in die Anweisungen eingestreut werden können, welche zur Auslieferungszeit an den Client durch ihr Berechnungsergebnis ersetzt werden.
Technisch ist der Ansatz auf Basis der existierenden Servlet-API realisiert und wird serverseitig auch transparent in ein solches übersetzt. Als einzige Einschränkung gegenüber der Mächtigkeit des Servletansatzes ist jedoch die Beschränkung auf die Verarbeitung von HTTP-POST-Anfragen zu sehen.

Beispiel 115 zeigt die Reformulierung der Servlet-basierten Umsetzung aus Beispiel 102 als JSP:

Beispiel 115: Einfache JSP

```
(1)<%@page import="java.util.Date"%>
(2)
(3)<html>
(4)     <head>
(5)             <title>Hello World!</title>
(6)     </head>
(7)     <body>
(8)             <h1>Hello World!</h1>
(9)             <p>Current date: <%= (new Date()).toString() %></p>
(10)    </body>
(11)</html>
```

Download des Beispiels

Augenfälligste Änderung gegenüber der Fassung als Servlet ist die direkte Darstellung der HTML-Ausgabeelemente. Die aktiven Inhalte werden durch Elemente, die mit dem Zeichen „%=“ beginnen und

mit „%“ enden eingeschlossen. Dazwischen befindet sich übersetzbarer Java-Quellcode.
Etwaige benötige Bibliotheken werden im Java-üblichen import-Stil angegeben und zu Beginn der JSP-Seite eingeführt.

Zu Beginn einer JSP-Seite können im Rahmen der page-Deklaration verschiedene seitenglobale Vereinbarungen getroffen werden:

- `language`
  Wurde ursprünglich für eine (immernoch ausstehende) Unterstützung des JSP-Mechanismus durch verschiedene Programmiersprachen eingeführt. Vorgabegemäß hat das Attribut den Wert „Java“. Andere Werte sind gegenwärtig nicht definiert.
- `extends`
  Erlaubt es die aus der JSP-Seite entstehende Servletklasse explizit als Spezialisierung der angegebenen Klasse zu definieren.
- `import`
  Kommaseparierte Liste der verwendeten Bibliotheken.
- `session`
  Legt fest, ob die JSP-Seite im Rahmen des HTTP-Sessionhandlings zustandsbehaftet verarbeitet wird.
- `buffer`
  Erlaubt die Festlegung der innerhalb des übersetzten Servelts verwendeten Ausgabemechanismus. Vorgabegemäß und bei Angabe von none ist dies `PrintWriter`.
- `autoFlush`
  Legt fest, daß gefüllte Ausgabepuffer automatisch an den anfragenden Client übermittelt werden.
- `isThreadSafe`
  Weist dieses Attribut den Wert `true` auf, dann werden durch die Ausführungsumgebungen mehrere Anfragen zur selben Zeit an die JSP (bzw. das entstehende Servlet) übergeben.
- `info`
  Möglichkeit zur Ablage informaler deskriptiver Information.
- `errorPage`
  Referenz auf eine andere JSP, die zur Verarbeitung von Ausnahmeereignissen herangezogen wird.
- `isErrorPage`
  Kennzeichent ob eine JSP-Seite zur Fehlerbehandlung einer anderen Seite herangezogen werden kann und soll.

Die nähere Analyse des Quellecodes von Beispiel 115 sowie die Semantik der vorgestellten Attribute offenbart zwei unterschiedliche Codierungsformen für JSPs. Zunächst die durch <@ eingeleiteten vordefinierten **Direktiven**, welche im Rahmen des Serveltübersetzungsprozesses in Java-Code transformiert werden. Diese Codeabbildung wird durch eine sog. *Tag Library*, einer Sammlung von Javacodefragementen und -transformationsvorschriften zur dynamischen Erzeugung der Ausgabe, gesteuert.
Zum zweiten gestattet der JSP-Mechanismus die direkte Darstellung von Java-Quellcode, der zur Laufzeit direkt in ausführbaren Code umgesetzt wird im Rahmen der sog. **Scriptlets**. Diese durch <% eingeführten Codeanteile werden unverädert in das entstehende Servlet einkopiert.
Zusätzlich stellt das Beispiel, durch <= eingeleitete, Ausdrücke (**Expressions**) dar, deren Resultatwert durch die Laufzeitumgebung in die an den Client übermittelte Seite eingefügt wird.
Im Beispiel nicht dargestellt sind Variablen- oder Konstantendeklarationen, welche auf die Einleitungssequenz <%! folgen.

Tag Libraries

Zur Strukturierung des Entwurfs und zur Wiederverwendung bereits einmal eingesetzter Codesequenzen für JSPs existiert mit dem Mechanismus der Tag Libraries die Möglichkeit anwenderdefiniert den Sprachumfang der Server Pages zu erweitern.
Einige häufig gebrauchte Aktionen sind bereits in der im Standardumfang enthaltenen Bibliothek versammelt:

- `useBean`
  Zugriff auf eine Enterprise Java Bean.
- `setProperty`
  Setzen einer Java-Property. Wird zumeist in Verbindung mit useBean verwendet um Bean-Eigenschaften festzulegen.
- `getProperty`
  Auslesen einer Java-Property.
- `param`
  Dient zur Parameterübergabe an andere Direktiven.
- `include`
  Inklusion anderer Datenquellen.
- `forward`
  Weiterleitung des Aufrufes der die JSP-Seite erreichte an andere Ressource.
- `plugin`
  Einschluß eines Java-Applets in die erzeugte Ausgabe.

Beispiel 116 zeigt mit der `include`-Direktive die vordefinierte JSP-Aktion zur serverseitigen Inklusion

existierender Dateien.

Beispiel 116: Importmechanismus für Server Pages

```
(1)<html>
(2)     <head>
(3)             <title>J. W. von Goethe: Faust I</title>
(4)     </head>
(5)     <body>
(6)             <h1>FAUST: EINE TRAGÖDIE</h1>
(7)             <h2>Zueignung.</h2>
(8)             <%@ include file="Zueignung.html" %>
(9)
(10)            <h2>Vorspiel auf dem Theater.</h2>
(11)            <%@ include file="Vorspiel.html" %>
(12)
(13)            <h2>Prolog im Himmel.</h2>
(14)            <%@ include file="Prolog.html" %>
(15)    </body>
(16)</html>
```

Download des Beispiels

Die Erzeugung eigener Tag Libraries geschieht durch Erstellung einer Java-Klasse, welche die Schnittstelle `javax.servlet.jsp.tagext.Tag` implementiert. Zumeist wird jedoch die Klasse `TagSupport` spezialisiert, die einige zusätzliche Methoden definiert, welche die Erstellung vereinfachen.

Beispiel 117 zeigt die Umsetzung des anwenderdefinierten Tags `hello` und Beispiel 118 seine Anwendung.

Beispiel 117: Erstellung einer Tag Library

```
(1)import java.io.IOException;
(2)import java.util.Date;
(3)
(4)import javax.servlet.jsp.JspTagException;
(5)import javax.servlet.jsp.tagext.TagSupport;
(6)
(7)public class HelloTag extends TagSupport {
(8)     public int doStartTag() throws JspTagException {
(9)             return EVAL_BODY_INCLUDE;
(10)    }
(11)
(12)    public int doEndTag() throws JspTagException {
(13)            try {
(14)                    pageContext.getOut().write("Hello World!");
(15)                    pageContext.getOut().write(
(16)                            "current date: " + new Date().toString());
(17)            } catch (IOException ioe) {
(18)                    ioe.printStackTrace();
(19)            }
(20)            return EVAL_PAGE;
(21)    }
(22)
(23)}
```

Download des Beispiels

Das Beispiel definiert Methoden für die beiden durch die Schnittstelle vorgegebenen Operationen `doStartTag` und `doEndTag`, die bei der Verarbeitung der entsprechenden Tag-Anteile aufgerufen werden. Der Rückgabewert der Methoden legt fest, daß die Inhalte des anwenderdefinierten Elements in der JSP-Seite in den Ausgabestrom übernommen werden (`EVAL_BODY_INCLUDE`) bzw. das mit der Verarbeitung der Seite fortgefahren werden soll (`EVAL_PAGE`).

Beispiel 118: Nutzung eines anwenderdefinierten Tags

```
(1)<%@ taglib uri="hello" prefix="example" %>
(2)<html>
(3)     <head>
(4)             <title>Hello World!</title>
(5)     </head>
(6)     <body>
(7)             <p>Statische Ausgabe ...</p>
(8)             <example:hello>
(9)             </example:hello>
(10)    </body>
(11)</html>
```

Download des Beispiels

Abschließende Bemerkung:
Der JSP-Mechanismus eignet sich sehr gut zur schnellen Erstellung leistungsfähiger Anwendungen einer gewissen Komplexität jedoch wirkt sich die physische Kopplung des Kontrollflusses an die erzeugte Ausgabe durch die Ablage in derselben Datei negativ auf die Skalierbarkeit hinsichtlich der betrachteten Problemgröße aus. Durch die mangelnde zentralisierte Verwaltbarkeit lassen sich komplexe Anwendungen oft nur noch schwer überschauen und testen.
Zusätzlich wirkt sich die erst zum Aufrufzeitpunkt serverseitig vorgenommene Übersetzung in ein Servlet nicht nur negativ auf die Laufzeit aus, sondern erschwert zusätzlich die Fehlersuche, da Syntaxfehler erst beim Aufruf entdeckt werden können.

5.3 Java Server Faces (JSF)

TODO(Hinweis: Dieses Kapitel fehlt noch)

5.4 XSL Transformations (XSLT)

Die prominenteste Verwendung der XPath-Pfadausdrücke und eine der in jüngerer Zeit am weitesten beachteten XML-Vokabulare dürfte XSLT -- die Sprache der *XSL Transformations* -- sein. Sie wurde im Verlauf der Standardisierung der Stylesheetsprache XSL von dieser abgetrennt und seither durch eine eigene W3C-Arbeitsgruppe vorangetrieben.
In einem Satz zusammengefaßt stellt sie eine Turing-vollständige funktionale Programmiersprache zur Transformation wohlgeformter XML-Dokumente in beliebige Unicode-Streams -- und damit im speziellen wiederum in XML-Dokumente -- dar.
Der Begriff *Transformationen* bezeichnet hierbei die Selektion einzelner Bestandteile des Quelldokuments, deren Umordnung sowie die Ableitung neuer Inhalte aus den bereits bestehenden.
Derzeit aktuell ist die Recommendation zur Version 1.0 von Herbst 1999. Intern arbeitet das W3C bereits an der Nachfolgerversion XSLT v2.0, welche auch die absehbaren Entwicklungen des Umfeldes, wie XPath v2.0 oder XML-Schema, berücksichtigen wird.

Jedes XSLT-Stylesheet ist ein gültiges XML-Dokument, in dem alle Elemente der Sprache XSLT im Namensraum `http://www.w3.org/1999/XSL/Transform` plaziert sind.
Üblicherweise wird der Namensraum an das Präfix `xsl` gebunden, welches allen XSLT-Sprachelementen explizit vorangestellt wird.
Das Wuzelelement eines XSLT-Dokuments bildet der Knoten `stylesheet`. Alternativ kann auch `transform` angegeben werden. Zusätzlich verfügt jedes Stylesheet über ein Versionsattribut zur Bezeichnung der verwendeten XSLT-Version.
Das Beispiel zeigt ein minimales Stylesheet

Beispiel 119: Ein minimales Stylesheet

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

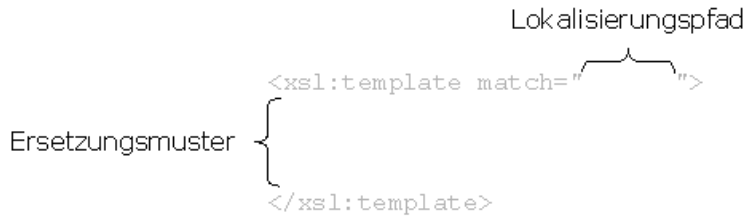
Download des Beispiels

Angewendet auf das Beispieldokument der Projektverwaltung liefert es folgende Ausgabe:

```
(1)<?xml version="1.0" encoding="utf-8"?>
(2)Hans Hinterhuber
(3)Franz Xaver Obermüller
(4)IT-Kompetenz verschiedene Betriebssysteme und professionelle Programmierung
verschiedener Programmiersprachen
(5)Entwickler von 1988-1990 Projektleiterfunktion von 1990-93 im X42-Projekt in Abteilung
AB&amp;amp;C
(6)Fritz Meier
```

Dies mag zunächst verwundern, definiert doch das Beispiel-Stylesheet keinerlei Transformationsregeln oder Ausgabefunktionen.
Das Ergebnis des minimal-Beispiels wird durch die eingebauten Vorgaberegeln erzeugt. Diese geben, sofern nicht anders angegeben alle Character Information Items unverändert aus.
Durch Überschreiben dieser Vorgaben und die Definition eigener Regeln lassen sich komplexe Transformationen auf dem Eingabedokument verwirklichen.

Transformationsschablonen



Die Graphik zeigt den Aufbau einer Transformationsschablone. Ihre beiden Hauptbestandteile sind der *Lokalisierungspfad* und das *Ersetzungsmuster*.
Der Lokalisierungspfad (in der Spezifikation als *pattern* bezeichnet) liefert eine Knotenmenge. Als Syntax wird eine eingeschränkte Variante der Lokatorsprache XPath verwendet.
Augenfälligster Unterschied zur bisherigen Notation ist die Optionalität der descendant-Achse (zumeist zu // verkürzt) zur hierarchiebenenunabhängigen Lokalisierung eines Knotens.
Im Rumpf des Musters legt das Ersetzungsmuster diejenige Zeichenfolge fest, die statt jedem Element der lokalisierten Knotenmenge ausgegeben werden soll.
Das nachfolgende Transformationssheet liefert angewandt auf das Projektverwaltungsbeispiel dreimal die Ausgabe `Person gefunden!`; für jedes Auftreten des Knotens `Person` in der Eingabe.

Beispiel 120: Eine einfache Transformation

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)     <xsl:template match="Person">
(6)             <xsl:text>Person gefunden!</xsl:text>
(7)     </xsl:template>
(8)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Innerhalb des Ersetzungsmusters können im Allgemeinen beliebige Textsequenzen angegeben werden. Insbesondere ist die Verwendung wohlgeformter XML-Fragmente zugelassen.
Textsequenzen werden hierbei durch direktes Anschreiben, oder umschlossen durch das Element `xsl:text` definiert.
Anmerkung: Die Anforderungen an die Wohlgeformtheit sind hierbei auf die korrekte Terminierung der Elemente (damit einhergehend ihre korrekte Schachtelung), sowie die Quotierung der Attribute beschränkt.
Im folgenden Beispiel wird jedes Element des Typs `Person` durch ein leeres `Mitarbeiter`-Element ersetzt. Das Beispiel erklärt auch die übliche Anwendungspraxis, alle XSLT-Elemente durch Namensraumpräfix zu qualifizieren, statt der -- weniger schreibaufwendigen -- Überschreibung des Vorgabenamensraumes. Würde der Vorgabenamensraum mit der Namensraum-URI der XSL-Transformations belegt, so befände sich auch jedes XML-Element und -Attribut innerhalb des Ersetzungsmusters in diesem Namensraum. Als Konsequenz würde der XSLT-Prozessor die Transformation wegen des Auftretens ungültiger (d.h. nicht in der XSLT-Sprache enthaltener) Elemente ablehnen. Bei Redefinition des Vorgabenamensraumes müßte daher für alle Elemente, die nicht Bestandteil von XSLT sind, eine explizite Namensraumdefinition im Element erfolgen, wodurch die Lesbarkeit stark herabgesetzt würde.

Beispiel 121: Erzeugung einer XML-Ausgabe

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter/>
(7)</xsl:template>
(8)
(9)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Erwartungsgemäß liefert das Beispiel dreifach das leere Element `Mitarbeiter` anstelle der `Personen`-Elemente der Eingabe.
Die bisher genutzte Form der Ersetzung ist jedoch für die praktische Anwendung in nur äußerst wenigen Fällen geeignet, da sie keine Übernahme von Daten der Eingabedatei in die Ausgabe erlaubt.
Dieses Manko wird durch das XSLT-Sprachelement `value-of` behoben.

Das Element `value-of` übernimmt als Teil des Ersetzungsmusters frei selektierbare Text-artige Informationsknoten aus dem Quelldokument. Die Lokalisierung der zu übernehmenden Knoten erfolgt

durch XPath-Syntax innerhalb des select-Attributs.
Im folgenden Beispiel wird der Inhalt der Personen-Knoten in den neuen Knotentyp Mitarbeiter übernommen.

Beispiel 122: Übernahme bestehender Information aus dem Quelldokument

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter>
(7)             <xsl:value-of select="self::*"/>
(8)     </Mitarbeiter>
(9)</xsl:template>
(10)
(11)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Das Resultat liefert jedoch nicht das Quelldokument, unter Umbenennung der Personen-Knoten in Mitarbeiter, sondern die dargestellte Textsequenz.

```
(1)<?xml version="1.0" encoding="utf-8"?>
(2)<Mitarbeiter>Hans Hinterhuber</Mitarbeiter>
(3)<Mitarbeiter>Franz Xaver Obermüller IT-Kompetenzverschiedene Betriebssysteme und
professionelle
(4)Programmierung verschiedener Programmiersprachen Entwickler von 1988-1990
Projektleiterfunktion
(5)von 1990-93 im X42-Projekt in Abteilung AB&amp;amp;C</Mitarbeiter>
(6)<Mitarbeiter>Fritz Meier</Mitarbeiter>
```

Die Lösung liegt in der Definition des value-of-Elements. Es konvertiert alle durch den im select-Attribut bezeichneten XPath lokalisierten Knoten in ihre Textrepräsentation. Im vorliegenden Beispiel ist dies der Inhalt der Knoten des Typs Person, der durch den Elementinhalt gebildet wird. Der Elementinhalt wird hierbei durch alle Kindknoten und deren Attribute gebildet.

Zur unveränderten Übernahme eines vollständigen Elements einschließlich der Auszeichnungssymbole wird das XSLT-Element copy-of angeboten.
Das nachfolgende Beispiel modifiziert das vorhergend diskutierte Stylesheet dergestalt, daß für alle Personen-Knoten zunächst ein öffnender Mitarbeiter-Tag gesetzt wird. Im Rumpf des so begonnenen Elements werden durch das copy-of-Element alle Kindknoten der aktuellen Knotenmenge unverändert kopiert.
Als zusätzliche Veränderung gegenüber dem vorigen Beispiel fällt die Nutzung der child-Achse im select-Attribut des copy-of-Elements auf. Dies ist notwendig, da durch den Lokalisierungspfad der Schablone alle Personen-Knoten zu einer Ergebnisknotenmenge zusammengefaßt werden. Die Anwendung der self-Achse innerhalb des copy-of-Elements würde daher auch die Personen-Knoten selbst übernehmen.
Zusammenfassend läßt sich die Funktionalität des Beispiels mit *Umbenennung aller Elemente des Types Person in Mitarbeiter* wiedergeben.

Beispiel 123: Kopieren vollständiger Elementknoten

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="//Person">
(6)     <Mitarbeiter>
(7)             <xsl:copy-of select="child::*"/>
(8)     </Mitarbeiter>
(9)</xsl:template>
(10)
(11)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Die vorgestellte Transformationsvorschrift ist hinreichend flexibel für Knoten des Typs Person auf beliebiger Hierarchiestufe des Eingabedokuments. Jedoch versagt sie bereits beim Versuch einen weiteren Transformationsschritt einzubauen, der auf einem der unverändert kopierten Kindelemente von Person operiert.
Das folgende Stylesheet stellt einen flexibleren Ansatz zur Umbenennung von einzelnen Knoten vor.

Gegenüber der vorhergehenden Fassung ist der Umfang um zwei weitere `template`-Elemente erweitert. Eines, das für alle `Qualifikationsprofil`-Knoten angewendet wird und eines für alle anderen Knoten. Der dort eingesetzte Lokalisierungspfad liefert als Ergebnismenge die Vereinigung aller Attributknoten (`attribute::*`) mit allen Knoten außer dem Wurzelknoten (`node()`-Funktion). Im Rumpf des Elements wird das copy-Element zur Kopie jedes Elements verwendet. Anders als das bisher herangezogene `copy-of`-Element übernimmt diese Variante ausschließlich das aktuelle Element in das Ergebnisdokument und läßt eventuell vorhandene Kindelemente unberücksichtigt.
Das `template`-Element mit dem Lokalisierungspfad `Qualifikationsprofil` verfügt über kein Ersetzungsmuster. Es bewirkt damit die Unterdrückung des Teilbaumes unterhalb aller Knoten des Typs `Qualifikationsprofil`.

Die korrekte Anwendung der verschiedenen Schablonen wird durch den ausführenden XSLT-Prozessor gewährleistet, er ermittelt anhand der Lokalisierungspfade das best-passendste Template und bringt es zur Ausführung. Wenn, wie im vorliegenden Beispiel, mehrere Pfadausdrücke einen Knoten des Eingabedokuments lokalisieren, so wird das am weitesten spezifizierte Muster ausgewählt. Im untenstehenden Beispiel gilt dies für alle Elemente des Typs `Person`. Jeder dieser Knoten ist sowohl durch den XPath-Ausdruck der ersten Schablone als auch durch den allgemeineren Ausdruck `node()` zugänglich. Der explizite Pfad `Person` des ersten Lokalisierungsmusters ist jedoch gegenüber der Ergebnismenge der node-Funktion (deutlich) spezifischer.

Da jeder Knoten, außer denen des Typs `Person` und `Qualifikationsprofil`, unverändert in den Ausgabestrom übernommen werden soll, wird im Beispiel wieder ein copy-Element eingesetzt. Im Unterschied zum bisher verwendeten `copy-of` jedoch mit der Einschränkung, daß copy nur den aktuellen Knoten kopiert und eventuell existierende Kindknoten unberücksichtigt läßt. Dieser Vorgang wird auch als *shallow copy* bezeichnet.

Steuerung der Transformationsreihenfolge durch `apply-templates`-Elemente

Standardmäßig durchläuft ein XSLT-Prozessor den aus dem Eingabedokument erzeugten Baum ausgehend vom Wurzelknoten in Preorder Reihenfolge. Während des Traversierungsvorganges werden die zum jeweiligen Knoten „passenden“ Schablonen ausgewertet. „Passend“ deutet hierbei auf das Enthaltensein des besuchten Knotens in der Ergebnismenge eines XPath-Ausdrucks innerhalb eines `match`-Attributes hin. Jedoch ist auch die anwenderdefinierte Beeinflussung der vorgegebenen Abarbeitungsreihenfolge möglich. Hierzu enthält das Beispiel zwei `apply-templates`-Elemente. Diese lösen einen Rekursionsschritt aus, dergestalt, daß an jeder Stelle, an der sich ein `apply-templates`-Aufruf findet, der Prozessor versucht, weitere passende Schablonen anhand der angegebenen Lokalisierungspfade zu ermitteln. Diese können an der gegebenen Stelle ausgewertet werden. Der Vorgang entspricht damit der Substitution in funktionalen Programmiersprachen.
Im vorliegenden Fall wird nach Ausgabe des öffnenden Tags `Mitarbeiter` -- nachdem ein `Personen`-Knoten im Eingabedokument ermittelt wurde -- nach weiteren Knoten im Eingabedokument gesucht, zu denen Lokalisierungspfade in der XSLT-Transformationsvorschrift existieren. Dies ist für alle Attribute und Kindknoten von `Person` der Fall, da sie durch den Lokalisierungspfad `attribute::*|node()` zugänglich sind. So wird innerhalb des neu erzeugten Elements `Mitarbeiter` des Ausgabestroms das Ersetzungsmuster ausgeführt, das die Elemente und Attribute mit ihren Inhalten unverändert übernimmt. Als Besonderheit nutzt das `apply-templates`-Element im „allgemeinen“ (dritten) `template` das Attribut `select`. Es erlaubt die anwenderdefinierte Steuerung der Menge, innerhalb der nach weiteren *passenden* Lokalisierungspfaden gesucht werden soll. Standardmäßig ist diese Menge mit allen dem aktuellen Element nachfolgenden (`following`-Achse) Elementknoten gefüllt. Im vorliegenden Beispiel wird sie durch den XPath des Attributwertes um alle Attributknoten erweitert.

Beispiel 124: Flexible Umbenennung und Löschung von Elementen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter>
(7)             <xsl:apply-templates/>
(8)     </Mitarbeiter>
(9)</xsl:template>
(10)
(11)<xsl:template match="Qualifikationsprofil"/>
(12)
(13)<xsl:template match="attribute::*|node()">
(14)    <xsl:copy>
(15)            <xsl:apply-templates select="attribute::*|node()"/>
(16)    </xsl:copy>
(17)</xsl:template>
(18)
(19)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Übung 3: Verfolgung der Aufrufreihenfolge
Lassen Sie sich mit von einem XSLT-Prozessor die Aufrufreihenfolge der einzelnen Templates ausgeben.
Beispielsweise mit dem Prozessor Xalan aus dem Apache-Projekt.

Benannte Ersetzungsschablonen und Parameterübergabe

Neben den Lokationspfad-gesteuerten Schablonen kann der Anwender auch benannte Schablonen, ohne `match`-Attribut, definieren. Diese werden während der Abarbeitung des Eingabebaumes nicht berücksichtigt, sondern müssen manuell durch `call-template` aufgerufen werden.
Konzeptionell entsprechen sie Funktionsaufrufen herkömmlicher Programmiersprachen.
(In Spezifikation nachschlagen).
Die Definition erfolgt analog den bisher genutzten Schablonen, mit der Ausnahme, daß statt dem `match`-Attribut ein eindeutiger Name (Attribute `name`) angegeben wird.

Als neuer Freiheitsgrad beim nun notwendigen manuellen Aufruf tritt die Möglichkeit der Parameterübergabe hinzu. Als Parameter können beliebige Dokumentbestandteile als Knotenmenge, Ergebnisse von Funktionsausdrücken oder Konstanten übergeben werden.
Eine Parameterrückgabe ist nicht möglich, sie wird durch den Anteil der Schablone an der Ausgabe realisiert.
Die Aufrufsyntax lautet:

```
(1)<xsl:call-template name="QName">
(2)     <!-- Content: xsl:with-param* -->
(3)</xsl:call-template>
(4)<xsl:with-param name="QName" select="eingeschränkter XPath-Ausdruck">
(5)     <!-- Content: template -->
(6)</xsl:with-param>
```

Anmerkung: Wie in allen funktionalen Sprachen kommt den Funktionen dieses Typs besondere Bedeutung, als Ausgangspunkt rekursiver Aufrufe, zu.

Die Vorgabe-Transformationsregeln

Das einführende Beispiel dieses Kapitels griff bereits auf die in den XSLT-Prozessor „eingebauten“ Standard-Transformationsvorschriften (*built-in templates*) zurück.
So lautet die Definition, welche für alle Text- und Attributknoten der Eingabe den Inhalt in die Ausgabe kopiert:

```
(1)<xsl:template match="text()|@*">
(2)     <xsl:value-of select="."/>
(3)</xsl:template>
```

Ferner existiert ein `template` zur rekursiven Abarbeitung des Eingabebaumes, welches immer dann Anwendung findet, wenn sich keine spezialisiertere Transformationsvorschrift findet.

```
(1)<xsl:template match="*|/">
(2)     <xsl:apply-templates/>
(3)</xsl:template>
```

Ergänzt wird diese Zusammenstellung durch eine Schablone zur Eliminierung von Namensraum- und Kommentarknoten.

Elemente der Ablaufsteuerung

Ähnlich zu herkömmlichen Programmiersprachen bietet auch XSLT Sprachmittel zur Selektion und bedingten Verarbeitung, abhängig von den Eingabedaten.
So erlaubt das `if`-Element die Bearbeitung der umschlossenen Elemente nur, wenn die im `test`-Attribut formulierte Boole'sche Bedingung wahr ist.
(In Spezifikation nachschlagen)
Die Syntax lautet:

```
<xsl:if test="Boole'scher Ausdruck">   <!-- Content: template --> </xsl:if>
```

Das Beispiel zeigt die Nutzung des `if`-Elements. Verfügt ein Knoten des Typs `Person` über mehr als einen Kindknoten des Typs `Vorname` so wird der Inhalt des `if`-Elements abgearbeitet, der durch das XSLT-Element `message` während des Transformationsvorganges eine Bildschirmausgabe erzeugt.
Diese gibt zunächst den Inhalt des `Nachnamen` Knotens gefolgt von einem statischen Text aus.

Angewandt auf das Beispieldokument liefert es auf Kommandozeile die Ausgabe: `Obermüller hat mehr als einen Vornamen!`. Der Inhalt des Eingabedokuments wird unverändert kopiert.

Beispiel 125: Bedingte Verarbeitung durch Verwendung des if-Elements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter>
(7)             <xsl:if test="count(Vorname)&gt;1">
(8)                     <xsl:message><xsl:value-of select="Nachname"/> hat mehr als einen
Vornamen!</xsl:message>
(9)             </xsl:if>
(10)            <xsl:apply-templates/>
(11)    </Mitarbeiter>
(12)</xsl:template>
(13)
(14)<xsl:template match="Qualifikationsprofil"/>
(15)
(16)<xsl:template match="attribute::*|node()">
(17)    <xsl:copy>
(18)            <xsl:apply-templates select="attribute::*|node()"/>
(19)    </xsl:copy>
(20)</xsl:template>
(21)
(22)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

In Erweiterung der simplen Selektion bildet choose die Möglichkeiten einer Mehrfachselektion, oder einer simplen if-then-else-Struktur ab.
(In Spezifikation nachschlagen)
Die Syntax lautet:

```
(1)<xsl:choose>
(2)     <!-- Content: (xsl:when+, xsl:otherwise?) -->
(3)</xsl:choose>
(4)<xsl:when test="Boole'scher Ausdruck">
(5)     <!-- Content: template -->
(6)</xsl:when>
(7)<xsl:otherwise>
(8)     <!-- Content: template -->
(9)</xsl:otherwise>
```

Das Beispiel zeigt die Transformation des Beispieldokuments in eine XHTML-Ausgabe.
Hierbei wird zunächst der XHTML-Dokumentrahmen bei Auftreten des Dokumentknotens (Suchmuster `/`) erzeugt. Nach dem öffnenden XHTML-Rumpfelement body werden innerhalb des Quelldokuments wahlfrei weitere, auf eines der angegebenen Suchmuster passende, Knoten gesucht (`apply-templates`-Aufruf).
Bei Auftreten des Knotens `ProjektVerwaltung` wird -- nach einigen Überschriftszeilen -- der Kopf einer XHTML-Tabelle, bestehend aus den Elementen `table` und der Kopfzeile (eingeschlossen durch `tr`), erzeugt. Den Rumpf der Tabelle schreibt ein anderes Template. Dieses wird jedoch nicht direkt aufgerufen, sondern der Prozeß der Ermittlung neuer „passender“ Knoten neu initiiert; jedoch diesmal, durch das `select`-Attribut des `apply-templates`-Elements, nur auf Knoten des Typs `Person` beschränkt.
Die Ersetzungsregel für `Person` speichert zunächst den Inhalt des Attributs `PersID` in einer Variable. Anschließend werden die Tabellenelemente der in der Ersetzungsregel für `ProjektVerwaltung` geöffneten Tabelle geschrieben. Hierzu werden nacheinander die Ersetzungsmuster für Knoten des Typs `Vorname` bzw. `Nachname`, die Kindknoten des aktuellen Knotens sind (die `PersID` des aktuellen `Personen`-Knotens findet sich in der zuvor belegten Variable), aktiviert.
Ein zweites Tabellenelement enthält einen XHTML-Hyperlink zu den durch den Mitarbeiter bearbeiteten Projekten. Als Sprungziel wird hierbei der Inhalt des Attributs `mitarbeitInProjekt` eingetragen.
Das Ersetzungsmuster `Vorname` übernimmt durch `value-of` die in einen Zeichenkettenwert gewandelten Inhalte des Elements. Zusätzlich existiert ein zweites Ersetzungsmuster für Knoten des Typs `Vorname`, das jedoch durch ein Prädikat nur auf das zweite und alle folgenden Elemente dieses Typs angewendet wird. Es gibt vor der Übernahme des Elementinhaltes ein Leerzeichen aus.
Das Element `Nachname` bettet den Elementinhalt in eine benannte Sprungreferenz ein. Zur Erzeugung der dokumentweiten eindeutigen Benennung wird die Funktion `generate-id` herangezogen, die für jedes Element des Information Sets des Eingabedokuments einen eindeutigen Bezeichner liefert.
Nach Abschluß der Tabellengenerierung im durch den Lokalisierungspfad `ProjektVerwaltung` gekennzeichneten Template werden durch den Aufruf des benannten Ersetzungsmusters `wasteSpace` 25 Leerzeilen, jeweils gefüllt mit der Zeichenkette „...“, erzeugt. Als Besonderheit ist in diesem Muster sehr deutlich die Nutzung der Rekursion zur Modellierung einer Schleife zu sehen.

Zum Abschluß wird nochmals eine Tabelle, nun mit den Mitarbeitern und ihren Projekten, erzeugt.

Beispiel 126: Erzeugung eines XHTML-Reports

```
(1)<?xml version="1.0"?>
(2)<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(3)
(4)<xsl:output method="xml" encoding="ISO-8859-1" omit-xml-declaration="no"  indent="yes"
(5)             xmlns="http://www.w3.org/1999/xhtml" />
(6)
(7)<xsl:template match="/">
(8)     <html>
(9)             <head>
(10)                    <title>XML-Vorlesung Sommersemester 2001 -- Projektverwaltung</
title>
(11)            </head>
(12)            <body>
(13)                    <xsl:apply-templates/>
(14)            </body>
(15)    </html>
(16)</xsl:template>
(17)
(18)<xsl:template match="ProjektVerwaltung">
(19)    <center><h1>Projektverwaltung</h1></center>
(20)
(21)    <h2>Projekte</h2>
(22)    <table border="1">
(23)            <tr>
(24)                    <td><b>Projektnummer</b></td>
(25)                    <td><b>Projektleiter</b></td>
(26)            </tr>
(27)            <xsl:apply-templates select="Projekt"/>
(28)    </table>
(29)
(30)    <pre>
(31)            <xsl:call-template name="wasteSpace">
(32)                    <xsl:with-param name="maxLines">25</xsl:with-param>
(33)                    <xsl:with-param name="curLines">0</xsl:with-param>
(34)            </xsl:call-template>
(35)    </pre>
(36)
(37)    <h2>Mitarbeiter</h2>
(38)    <table border="1">
(39)            <tr>
(40)                    <td><b>Name</b></td>
(41)                    <td><b>Projekt</b></td>
(42)            </tr>
(43)            <xsl:apply-templates select="Person"/>
(44)    </table>
(45)</xsl:template>
(46)
(47)    <xsl:template name="wasteSpace">
(48)            <xsl:param name="maxLines"/>
(49)            <xsl:param name="curLines"/>
(50)            <xsl:if test="number($curLines) &lt; number($maxLines)">
(51)<xsl:text>...
(52)</xsl:text>
(53)                    <xsl:call-template name="wasteSpace">
(54)                            <xsl:with-param name="maxLines"><xsl:value-of
select="$maxLines"/></xsl:with-param>
(55)                            <xsl:with-param name="curLines"><xsl:value-of
select="$curLines + 1"/></xsl:with-param>
(56)                    </xsl:call-template>
(57)            </xsl:if>
(58)    </xsl:template>
(59)
(60)<xsl:template match="Projekt">
(61)    <xsl:variable name="prjLeiter" select="@Projektleiter"/>
(62)    <tr>
(63)            <td>
(64)                    <xsl:element name="a">
(65)                            <xsl:attribute name="name"><xsl:value-of select="@ID"/></
xsl:attribute>
(66)                            <xsl:value-of select="@ID"/>
(67)                    </xsl:element>
(68)            </td>
(69)            <td>
```

```
(70)                    <xsl:element name="a">
(71)                            <xsl:attribute name="href">#<xsl:value-of select="generate-
id(//Person[@PersID=$prjLeiter]/Nachname)"/></xsl:attribute>
(72)                            <xsl:value-of select="//Person[@PersID=$prjLeiter]/
Nachname"/>
(73)                    </xsl:element>
(74)            </td>
(75)    </tr>
(76)</xsl:template>
(77)
(78)<xsl:template match="Person">
(79)    <xsl:variable name="persNr" select="@PersID"/>
(80)    <tr>
(81)            <td><xsl:apply-templates select="Vorname[parent::Person/@PersID=$persNr]"/
> 
(82)                     <xsl:apply-templates select="Nachname[parent::Person/@PersID=
$persNr]"/></td>
(83)            <td>
(84)                    <xsl:element name="a">
(85)                            <xsl:attribute name="href">#<xsl:value-of
select="@mitarbeitInProjekt"/></xsl:attribute>
(86)                            <xsl:value-of select="@mitarbeitInProjekt"/>
(87)                    </xsl:element>
(88)            </td>
(89)    </tr>
(90)</xsl:template>
(91)
(92)<xsl:template match="Vorname">
(93)    <xsl:value-of select="."/>
(94)</xsl:template>
(95)
(96)<xsl:template match="Vorname[position() &gt; 1]">
(97)    <xsl:text> </xsl:text><xsl:value-of select="."/>
(98)</xsl:template>
(99)
(100)<xsl:template match="Nachname">
(101)   <xsl:element name="a">
(102)           <xsl:attribute name="name"><xsl:value-of select="generate-id()"/></xsl:
attribute>
(103)           <xsl:value-of select="."/>
(104)   </xsl:element>
(105)</xsl:template>
(106)
(107)<xsl:template match="text()"/>
(108)
(109)</xsl:stylesheet>
```

Download des Beispiels
Download der Ergebnisdatei

Die nachfolgende XSLT-Transformation ermittelt zu jedem beliebigen Element- und Attributknoten (Muster: * bzw. @*) die zugehörige Namensraum-URI (Funktion namespace-uri) und gibt sie gemeinsam mit dem Namen des Knotens (Funktion name) in einer XML-formatierten Ausgabe aus.
Zur Neuselektion aller weiteren Element- und Attributknoten wird apply-templates mit dem Musterausdruck @*|node() ausgeführt, um in die Prüfung nach weiteren passenden Knoten (node()) auch explizit alle beliebigen Attribute (@*) einzubeziehen. Standardmäßig ermittelt apply-templates weitere passende Knoten nur innerhalb der Elemente eines Dokuments.

Beispiel 127: Ausgabe Namensräume jedes Elements und Attributs eines beliebigen XML-Dokuments

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:output method="xml" encoding="ISO-8859-1" omit-xml-declaration="no"  indent="yes" /
>
(6)
(7)<xsl:template match="*">
(8)     <xsl:element name="element">
(9)             <xsl:attribute name="name"><xsl:value-of select="name()"/></xsl:attribute>
(10)            <xsl:attribute name="namespace"><xsl:value-of select="namespace-uri()"/></
xsl:attribute>
(11)       <xsl:apply-templates select="@*|node()"/>
(12)    </xsl:element>
(13)</xsl:template>
```

```
(14)
(15)<xsl:template match="@*">
(16)    <xsl:element name="attribute">
(17)            <xsl:attribute name="name"><xsl:value-of select="name()"/></xsl:attribute>
(18)            <xsl:attribute name="namespace"><xsl:value-of select="namespace-uri()"/></
xsl:attribute>
(19)       <xsl:apply-templates select="@*|node()"/>
(20)    </xsl:element>
(21)</xsl:template>
(22)
(23)<xsl:template match="text()"/>
(24)
(25)</xsl:transform>
```

Download des Beispiels

Web-Referenzen 14: Weiterführende Links

- XSLT v1.0 Spezifikation
- XSLT @ jeckle.de
- Holman, K. G.: *What is XSLT?*
- Saxon -- ein freier XSLT-Prozessor
- Apache Xalan -- ein Open Source XSLT-Prozessor
- Literatur zum Thema

Übung 4: Textuelle Aufbereitung der Baumstruktur eines XML-Dokuments

Schreiben Sie eine XSLT-Transformation, die es erlaubt die Elementstruktur beliebiger wohlgeformter XML-Dokumente in Form eines „ASCII-Baums“ (siehe Beispiel) am Bildschirm auszugeben.
Beispiel:
Eingabe:

```
(1)<root>
(2)     <childX>
(3)             <childY1/>
(4)             <childY2/>
(5)     </childX>
(6)     <childX2/>
(7)</root>
```

Ausgabe:

```
(1)root
(2)+--childX
(3)     +--childY1
(4)     +--childY2
(5)+--childX2
```

---

6 Sicherheitsaspekte

6.1 Schlüsselaustausch

TODO(Hinweis: Dieses Kapitel fehlt noch)

6.2 Leitungssicherheit

TODO(Hinweis: Dieses Kapitel fehlt noch)

6.3 Digitale Signatur

TODO(Hinweis: Dieses Kapitel fehlt noch)

6.4 Verschlüsselung

TODO(Hinweis: Dieses Kapitel fehlt noch)

Definitionsverzeichnis

Schlagwortverzeichnis

Abbildungsverzeichnis

Verzeichnis der Beispiele

Service provided by Mario Jeckle
Generated: 2004-05-25T05:38:09+01:00

Feedback SiteMap
This page's original location: http://www.jeckle.de/vorlesung/eBusinessEng/script.html
RDF description for this page

Vorlesungen

Zugang zum Scriptum
Empfohlene Literatur
FAQs zur Vorlesung
Alte Klausuren
Evaluierung
Noten

Mailingliste zur Vorlesung
Vorlesungsinhalt
Lernziel
Links
Download des Scriptums
Semesteraufgaben des vergangenen Semesters

Aktuelles XML

Organisatorisches

Vorlesungszeiten

Die Vorlesung findet montags von 14 bis 17.15 Uhr im Raum C001 statt.

Die schriftliche Abschlußklausur findet am 2004-07-09 um 9.30 Uhr im Raum C001 statt.

Notenbildung

Die Notenbildung erfolgt hälftig durch die Bewertung des vorlesungsbegleitenden Praktikums und der schriftlichen Abschlußklausur am Semesterende.

Abstract

Die Vorlesung führt in die grundlegenden Techniken moderner e-Business

Anwendungen, insbesondere unter Berücksichtigung der zugrundeliegenden Integrationsaspekte ein. Hierzu wird zunächst eine Begriffsbestimmung vorgenommen. Ausgehend von e-Business-Anwendungsfällen wird ein Architektur zur Klassifizierung verschiedener technischer Realisierungskomponenten eingeführt.
Entlang dessen Architekturkomponenten werden verschiedene derzeit in der Praxis zur eingesetzte Realisierungstechniken vorgestellt. Im Vordergrund der Veranstaltungen stehen hierbei sowohl Java- als auch XML-basierte Techniken, die gleichermaßen als Schwerpunkt behandelt werden.
Besonderes Augenmerk wird zusätzlich verschiedenen Kommunikationsaspekten zuteil. Dabei werden neben der eigentlichen Datenübertragung auch Sicherheitsgesichtspunkte diskutiert.

Inhaltsübersicht

Lernziel

Der Student soll nach der Veranstaltung ...

- die zentralen Grundkonzepte moderner e-Business Applikationen beherrschen
- in der Lage sein aktuelle Techniken zur Realisierung von e-Business

Applikationen einzusetzen
- verschiedene Technikalternativen abwägen und bewerten können

Links

Allgemeines

- J2EE v1.4 Tutorial

Web Services vs. REST

- Building Web Services the REST Way
- eBay adds SOAP
- Interview mit Jeff Barr über die Amazon.com-Web-Services
- REST, SOAP, and the future
- REST (Dissertation von R. Fielding)
- Roots of the REST/SOAP Debate
- W3C TAG: When to use GET
- What is REST?

JDBC

- Connector/J -- JDBC-Treiber für MySQL
- JDBC @ SUN
- JDBC FAQ
- JDBC-Spezifikation

JSF

- Spezifikation und Referenzimplementierung

Download des Scriptums und der darin enthaltenen Beispiele

- PDF (tief verlinkt)
- PDF
- XML
- Beispiele (tgz)

Semesteraufgaben des vergangenen Semesters

Begleitend zur Vorlesung wurden über das Semester die beiden untenstehenden Aufgaben von je einer Gruppe bearbeitet.

- Mozilla als RSS-Reader
- XML-basierte Suchmaschine

Mailingliste

Steht im Wintra-System zur Verfügung.

Verwendete Symbole

Innerhalb des Scriptums werden folgende Symbole verwendet:

Tabelle 1: Eine Tabelle

A	B	C
x11	x12	x13
x21	x22	x23
x31	x32	x33

...

Definition 1: Eine Definition

Hier folgt ein furchtbar wichtiger Inhalt ;)

...

Beispiel 1: Ein Beispiel

```
<?xml version="1.0"?>
<firstExample>
   <welcome>hello world!</welcome>
</firstExample>
```

Download des Beispiels

...

Übung 1: Eine Übung

Besorgen Sie sich die XML-Spezifikation von der W3C-Homepage

...

Web-Referenzen 1: Einige interessante oder weiterführende Web-Adressen

Das World Wide Web Consortium (W3C)
Homepage der Vorlesung

...

Empfohlene Literatur

Originalliteratur

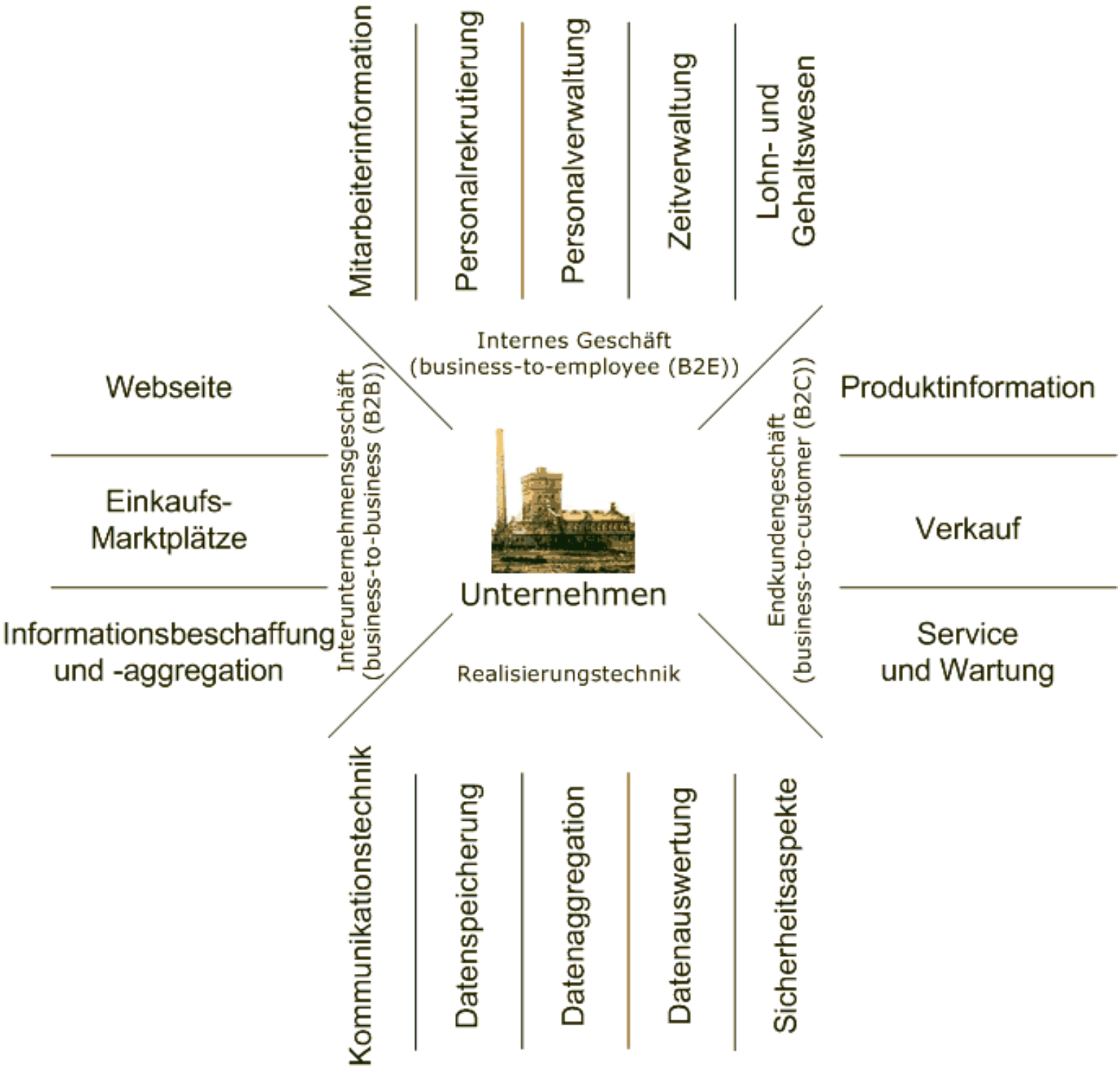
- XML v1.0, 2nd edition
- Namespaces in XML
- XML-Schema
  - Part 0: Primer
  - Part 1: Structures
  - Part 2: Datatypes
- XHTML 1.1 - Module-based XHTML
- XSL Transformations (XSLT) Version 1.0
- XML Path Language (XPath) Version 1.0
- HTTP v1.1, RFC 2616
- SSL v3 Specification
- TLS v1.1 Specification

Primärliteratur

- Scriptum zur Vorlesung
- Kommentierte deutsche Übersetzung der zentralen XML-Standards
- Zusammenstellung zur XML-Spezifikation.
  Bestehend aus dem offiziellen (englischsprachigen) W3C-Dokument, einer durch Tim Bray kommentierten Fassung und der deutschen Übersetzung.
- B. Schneier, N. Ferguson: Practical Cryptography, John Wiley & Sons, 2003.
- D. Stinson: Cryptography, Theory and Practice, 2nd ed., CRC Press, Boca Raton, 2002.

Sekundärliteratur und Web Links

- J2EE (Java Enterprise Beans, Java Server Pages, Servlets, Java Server Pages)
- JDBC @ SUN
- RMI @ SUN
- S. Mintert, H. Behme: *XML in der Praxis*
- WML @ WAPForum.org


Mitarbeiterinformation
Personalrekrutierung
Personalverwaltung
Zeitverwaltung
Lohn- und Gehaltswesen
Internes Geschäft (business-to-employee (B2E))
Webseite
Einkaufs-Marktplätze
Informationsbeschaffung und -aggregation
Interunternehmensgeschäft (business-to-business (B2B))
Unternehmen
Realisierungstechnik
Endkundengeschäft (business-to-customer (B2C))
Produktinformation
Verkauf
Service und Wartung
Kommunikationstechnik
Datenspeicherung
Datenaggregation
Datenauswertung
Sicherheitsaspekte

about XML . XML Activity Statement

Extensible Markup Language (XML)

Nearby: XML Specifications and Translations of them.

Introduction

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.

This page describes the work being done at W3C within the XML Activity, and how it is structured. Work at W3C takes place in *Working Groups*. The Working Groups within the XML Activity are listed below, together with links to their individual web pages.

You can find and download formal technical specifications here, because we publish them. This is ***not*** a place to find tutorials, products, courses, books or other XML-related information. There are some links below that may help you find such resources.

You will find links to W3C Recommendations, Proposed Recommendations, Working Drafts, conformance test suites and other documents on the pages for each Working Group. Each document also contains email addresses you can use to send comments or questions, for example if you have been writing software to implement them and have found problems or errors.

Please do ***not*** send us email asking you to help learn a language or specification; there are plenty of resources online, and the people editing and developing the specifications are very busy. We ***are*** interested in technical comments and errata.

If your organization would like to join the W3C, or if you would like to participate

formally in a working group (and have the necessary resources to attend meetings), you can read more about the Consortium.

Working Groups

There is more detail about each of these Working Groups in the Activity Statement and also on the individual Working Group public web pages.

Most Working Groups have both a public web page and another more private one that is only accessible to W3C Members. The private page has telephone numbers, schedules for meetings and conference calls, links to internal editing drafts, and other administrative information.

If you find you can't get to the private pages but believe you should be able to do so, you can apply for access.

XML Coordination Group

The membership of this group is the Chairs of the individual Working Groups. Its role is to provide a forum for coordination between the Working Groups of the XML Activity, and between the XML Activity and other parts of W3C, and between the XML Activity and other organizations.

This group does not produce specifications, so does not have a public page of its own. You can read the XML CG Charter, and there is more information about the XML CG in the Activity Statement. There is is also a member-only page.

XML Core Working Group

The mission of the XML Core Working Group is to develop and maintain the specifications for XML itself and closely related specifications such as Namespaces in XML, the XML Information Set, and XInclude.

You can read the XML Core Working Group Public Page and the XML Core Working Group Charter, and there is also a member-only page.

XSL Working Group

The XSL Working Group is responsible for the Extensible Stylesheet Language (XSL), including both XSL Transformations (XSLT) and XSL Formatting Objects (XSL/FO). They moved into the W3C Architecture Domain at the start of 2003.

You can read the XSL Working Group Public Page which links to their Charter and to their member-only page.

XML Binary Characterization Working Group

In September of 2003 the W3C held a public Workshop to discuss whether W3C should develop a binary interchange format for people needing greater efficiency than was claimed possible using text. The conclusion of the Workshop was that it was not clear whether the benefits of such a format could outweigh the costs, but that the issues involved needed further study. The XML Binary Characterization Working Group has been created to investigate this area, and to enable precise measurements to be made of the benefits (if any) of a binary interchange format over the existing methods of textual interchange. The Working Group is explicitly ***not*** chartered to do any technical work on developing or choosing a single format: that would be the work of a future Working Group, if and only if the XML Binary Characterization Working Group can demonstrate that such efforts would be worth while.

You can read the XML Binary Characterization Working Group Public Page which links to their charter and to their Member-only page.

XML Linking Working Group

The charter of the XML Linking Working Group has expired, and the group is not currently active. When still active, it was working on hypertext links for XML. This includes the XML Linking Language (XLink) and the XML Pointer Language (XPointer).

You can read the XML Linking Working Group Public Page and the XML Linking Working Group Charter, and there is also a member-only page.

XML Query Working Group

The XML Query Working Group is working on the XML Query Language, a way to provide flexible query facilities to extract data from real and virtual XML documents on the Web. This includes publication of XQuery and also XPath, in conjunction with the XSL Working Group (part of the Style Activity).

You can read the XML Query Working Group Public Page and the XML Query Working Group Charter, and there is also a member-only page.

XML Schema Working Group

W3C XML Schemas provide mechanisms to define and describe the structure, content, and to some extent semantics of XML documents.

You can read the XML Schema Group Public Page and the XML Schema Working Group Charter, and there is also a member-only page.

Other Resources

There are so many resources related to XML that we can't possibly list them all here. This is a ***good*** thing, because it means XML is a success! In addition to a history of the development of XML at W3C, there is an extensive index at the Cover Pages, maintained by Robin Cover. The individual Working Group public web pages may have links to specific resources. There are Usenet newsgroups (e.g. comp.text.xml) and public mailing lists (e.g. xml-dev).

You could also try a search engine such as Google for:

- XML conferences
- books
- training courses
- online tutorials
- bibliographies
- parsers (both proprietary and open source)
- magazines
- and even movies

Contact

Liam Quin, XML Activity Lead

Last modified $Date: 2004/03/26 23:28:48 $

Note

The XML specification, and other information specific to the XML Core Working Group, has moved to the XML Core Working Group Public Page.

There is also a separate page for Translations.

There is a separate page documenting the xml-spec DTD used for many of our specifications.

Namespaces in XML 1.1

W3C Recommendation 4 February 2004

This version:
http://www.w3.org/TR/2004/REC-xml-names11-20040204
Latest version:
http://www.w3.org/TR/xml-names11
Previous version:
http://www.w3.org/TR/2003/PR-xml-names11-20031105
Editors:
Tim Bray, Textuality <tbray@textuality.com>
Dave Hollander, Contivo, Inc. <dmh@contivo.com>
Andrew Layman, Microsoft <andrewl@microsoft.com>
Richard Tobin, University of Edinburgh and Markup Technology Ltd <richard@cogsci.ed.ac.uk> - Version 1.1

Please refer to the **errata** for this document, which may include some normative corrections.

See also **translations**.

This document is also available in these non-normative formats: XML.

Abstract

XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by

associating them with namespaces identified by IRI references.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document is a Recommendation of the W3C. It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document is a product of the W3C XML Activity. The English version of this specification is the only normative version. However, for translations of this document, see http://www.w3.org/2003/03/Translations/byTechnology?technology=xml-names11.

Documentation of intellectual property possibly relevant to this recommendation may be found at the Working Group's public IPR disclosure page.

Known implementations are documented in the Namespaces 1.1 implementation report. A test suite is also available via the XML Test Suite page.

Please report errors in this document to xml-names-editor@w3.org; public archives are available. The errata list for this document is available at http://www.w3.org/XML/2004/xml-names11-errata.

Table of Contents

Appendices

---

1 Motivation and Summary

We envision applications of Extensible Markup Language (XML) where a single XML document may contain elements and attributes (here referred to as a "markup vocabulary") that are defined for and used by multiple software modules. One motivation for this is modularity: if such a markup vocabulary exists which is well-understood and for which there is useful software available, it is better to re-use this markup rather than re-invent it.

Such documents, containing multiple markup vocabularies, pose problems of recognition and collision. Software modules need to be able to recognize the elements and attributes which they are designed to process, even in the face of "collisions" occurring when markup intended for some other software package uses the same element name or attribute name.

These considerations require that document constructs should have names constructed so as to avoid clashes between names from different markup vocabularies. This specification describes a mechanism, *XML namespaces*, which accomplishes this by assigning expanded names to elements and attributes.

1.1 A Note on Notation and Usage

Where *EMPHASIZED*, the key words *MUST*, *MUST NOT*, *REQUIRED*, *SHOULD*, *SHOULD NOT*, *MAY* in this document are to be interpreted as described in [Keywords].

Note that many of the nonterminals in the productions in this specification are defined not here but in the XML specification [XML]. When nonterminals defined here have the same names as nonterminals defined in the XML specification, the productions here in all cases match a subset of the strings matched by the corresponding ones there.

In this document's productions, the NSC is a "Namespace Constraint", one of the rules that documents conforming to this specification *MUST* follow.

2 XML Namespaces

2.1 Basic Concepts

[Definition: An **XML namespace** is identified by an IRI reference; element and attribute names may be placed in an XML namespace using the mechanisms described in this specification.]

[Definition: An **expanded name** is a pair consisting of a namespace name and a local name.] [Definition: For a name *N* in a namespace identified by an IRI *I*, the **namespace name** is *I*. For a name *N* that is not in a namespace, the **namespace name** has no value.] [Definition: In either case the **local name** is *N*.] It is this combination of the universally managed IRI namespace with the vocabulary's local names that is effective in avoiding name clashes.

IRI references can contain characters not allowed in names, and are often inconveniently long, so expanded names are not used directly to name elements and attributes in XML documents. Instead qualified names are used. [Definition: A **qualified name** is a name subject to namespace interpretation.] In documents conforming to this specification, element and attribute names appear as qualified names. Syntactically, they are either prefixed names or unprefixed names. An attribute-based declaration syntax is provided to bind prefixes to namespace names and to bind a default namespace that applies to unprefixed element names; these declarations are scoped by the elements on which they appear so that different bindings may apply in different parts of a document. Processors conforming to this specification *MUST* recognize and act on these declarations and prefixes.

2.2 Use of IRIs as Namespace Names

The empty string, though it is a legal IRI reference, cannot be used as a namespace name.

The use of relative IRI references, including same-document references, in namespace declarations is deprecated.

> **Note:**
>
> This deprecation of relative URI references was decided on by a W3C XML Plenary Ballot [Relative URI deprecation]. It also declares that "later specifications such as DOM, XPath, etc. will define no interpretation for them".

2.3 Comparing IRI References

IRI references identifying namespaces are compared when determining whether a name belongs to a given namespace, and whether two names belong to the same namespace. [Definition: The two IRIs are treated as strings, and they are **identical** if and only if the strings are identical, that is, if they are the same sequence of characters.] The comparison is case-sensitive, and no %-escaping is done or undone.

A consequence of this is that IRI references which are not identical in this sense may resolve to the same resource. Examples include IRI references which differ only in case or %-escaping, or which are in external entities which have different base URIs (but note that relative IRIs are deprecated as namespace names).

In a namespace declaration, the IRI reference is the normalized value of the attribute, so replacement of XML character and entity references has already been done before any comparison.

Examples:

The IRI references below are all different for the purposes of identifying namespaces, since they differ in case:

- `http://www.example.org/wine`
- `http://www.Example.org/wine`
- `http://www.example.org/Wine`

The IRI references below are also all different for the purposes of identifying

namespaces:

- `http://www.example.org/rosé`
- `http://www.example.org/ros%c3%a9`
- `http://www.example.org/ros%c3%A9`
- `http://www.example.org/ros%C3%a9`
- `http://www.example.org/ros%C3%A9`

As are these:

- `http://www.example.org/~wilbur`
- `http://www.example.org/%7ewilbur`
- `http://www.example.org/%7Ewilbur`

If the entity **eacute** has been defined to be **é**, the start tags below all contain namespace declarations binding the prefix **p** to the same IRI reference, `http://example.org/rosé`.

- `<p:foo xmlns:p="http://example.org/rosé">`
- `<p:foo xmlns:p="http://example.org/ros&#xe9;">`
- `<p:foo xmlns:p="http://example.org/ros&#xE9;">`
- `<p:foo xmlns:p="http://example.org/ros&#233;">`
- `<p:foo xmlns:p="http://example.org/ros&eacute;">`

Because of the risk of confusion between IRIs that would be equivalent if dereferenced, the use of %-escaped characters in namespace names is strongly discouraged.

3 Declaring Namespaces

[Definition: A namespace (or more precisely, a namespace binding) is **declared** using a family of reserved attributes. Such an attribute's name must either be **xmlns** or begin **xmlns:**. These attributes, like any other XML attributes, may be provided directly or by default.]

Attribute Names for Namespace Declaration

```
[1]   NSAttName         ::  PrefixedAttName
                          =
                            | DefaultAttName
```

```
[2]   PrefixedAttName  ::= 'xmlns:' NCName        [NSC: Reserved Prefixes and Namespace Names]
[3]   DefaultAttName   ::= 'xmlns'
[4]   NCName           ::= NCNameStartChar NCNameChar*    /* An XML Name, minus the ":" */
[5]   NCNameChar       ::= NameChar - ':'
[5a]  NCNameStartChar  ::= NameStartChar - ':'
```

The attribute's normalized value *MUST* be either an IRI reference — the namespace name identifying the namespace — or an empty string. The namespace name, to serve its intended purpose, *SHOULD* have the characteristics of uniqueness and persistence. It is not a goal that it be directly usable for retrieval of a schema (if any exists). Uniform Resource Names [RFC2141] is an example of a syntax that is designed with these goals in mind. However, it should be noted that ordinary URLs can be managed in such a way as to achieve these same goals.

[Definition: If the attribute name matches PrefixedAttName, then the NCName gives the **namespace prefix**, used to associate element and attribute names with the namespace name in the attribute value in the scope of the element to which the declaration is attached.]

[Definition: If the attribute name matches DefaultAttName, then the namespace name in the attribute value is that of the **default namespace** in the scope of the element to which the declaration is attached.] Default namespaces and overriding of declarations are discussed in **6 Applying Namespaces to Elements and Attributes**.

An example namespace declaration, which associates the namespace prefix **edi** with the namespace name `http://ecommerce.example.org/schema`:

```
<x xmlns:edi='http://ecommerce.example.org/schema'>
  <!-- the "edi" prefix is bound to http://ecommerce.
example.org/schema
```

```
        for the "x" element and contents -->
  </x>
```

Namespace constraint: Reserved Prefixes and Namespace Names

The prefix **xml** is by definition bound to the namespace name `http://www.w3.org/XML/1998/namespace`. It *MAY*, but need not, be declared, and *MUST NOT* be undeclared or bound to any other namespace name. Other prefixes *MUST NOT* be bound to this namespace name.

The prefix **xmlns** is used only to declare namespace bindings and is by definition bound to the namespace name `http://www.w3.org/2000/xmlns/`. It *MUST NOT* be declared or undeclared. Other prefixes *MUST NOT* be bound to this namespace name.

All other prefixes beginning with the three-letter sequence x, m, l, in any case combination, are reserved. This means that:

- users *SHOULD NOT* use them except as defined by later specifications
- processors *MUST NOT* treat them as fatal errors.

Though they are not themselves reserved, it is inadvisable to use prefixed names whose LocalPart begins with the letters x, m, l, in any case combination, as these names would be reserved if used without a prefix.

4 Qualified Names

In XML documents conforming to this specification, some names (constructs corresponding to the nonterminal Name) *MUST* be given as qualified names, defined as follows:

Qualified Name

```
[6]    QName             ::=  PrefixedName
                              | UnprefixedName
[6a]   PrefixedName      ::=  Prefix ':'
                              LocalPart
[6b]   UnprefixedName    ::=  LocalPart
[7]    Prefix            ::=  NCName
```

[8]	LocalPart	::=	NCName

The Prefix provides the namespace prefix part of the qualified name, and *MUST* be associated with a namespace IRI reference in a namespace declaration. [Definition: The LocalPart provides the **local part** of the qualified name.]

Note that the prefix functions *only* as a placeholder for a namespace name. Applications *SHOULD* use the namespace name, not the prefix, in constructing names whose scope extends beyond the containing document.

5 Using Qualified Names

In XML documents conforming to this specification, element names are given as qualified names, as follows:

Element Names

[9]	STag	::=	'<' QName (S Attribute)* S? '>'	[NSC: Prefix Declared]
[10]	ETag	::=	'</' QName S? '>'	[NSC: Prefix Declared]
[11]	EmptyElemTag	::=	'<' QName (S Attribute)* S? '/>'	[NSC: Prefix Declared]

An example of a qualified name serving as an element name:

```
  <!-- the 'price' element's namespace is http://
ecommerce.example.org/schema -->
  <edi:price xmlns:edi='http://ecommerce.example.org/
schema' units='Euro'>32.18</edi:price>
```

Attributes are either namespace declarations or their names are given as qualified names:

Attribute

[12]	Attribute	::=	NSAttName Eq AttValue	
			| QName Eq AttValue	[NSC: Prefix Declared]

An example of a qualified name serving as an attribute name:

```
<x xmlns:edi='http://ecommerce.example.org/schema'>
  <!-- the 'taxClass' attribute's namespace is http://
ecommerce.example.org/schema -->
  <lineItem edi:taxClass="exempt">Baby food</lineItem>
</x>
```

Namespace constraint: Prefix Declared

The namespace prefix, unless it is `xml` or `xmlns`, *MUST* have been declared in a namespace declaration attribute in either the start-tag of the element where the prefix is used or in an ancestor element (i.e. an element in whose content the prefixed markup occurs). Furthermore, the attribute value in the innermost such declaration *MUST NOT* be an empty string.

This constraint may lead to operational difficulties in the case where the namespace declaration attribute is provided, not directly in the XML document entity, but via a default attribute declared in an external entity. Such declarations may not be read by software which is based on a non-validating XML processor. Many XML applications, presumably including namespace-sensitive ones, fail to require validating processors. If correct operation with such applications is required, namespace declarations *MUST* be provided either directly or via default attributes declared in the internal subset of the DTD.

Element names and attribute names are also given as qualified names when they appear in declarations in the DTD:

Qualified Names in Declarations

```
[13]  doctypedecl  ::=  '<!DOCTYPE' S QName (S ExternalID)?
                        S? ('[' (markupdecl | PEReference |
                        S)* ']' S?)? '>'
[14]  elementdecl  ::=  '<!ELEMENT' S QName S contentspec
                        S? '>'
[15]  cp           ::=  (QName | choice | seq) ('?' | '*' |
                        '+')?
```

```
[16]  Mixed        ::=  '(' S? '#PCDATA' (S? '|' S? QName)*
                        S? ')*'
                      | '(' S? '#PCDATA' S? ')'
[17]  AttlistDecl  ::=  '<!ATTLIST' S QName AttDef* S? '>'
[18]  AttDef       ::=  S (QName | NSAttName) S AttType S
                        DefaultDecl
```

Note that DTD-based validation is not namespace-aware in the following sense: a DTD constrains the elements and attributes that may appear in a document by their uninterpreted names, not by (namespace name, local name) pairs. To validate a document that uses namespaces against a DTD, the same prefixes must be used in the DTD as in the instance. A DTD may however indirectly constrain the namespaces used in a valid document by providing `#FIXED` values for attributes that declare namespaces.

6 Applying Namespaces to Elements and Attributes

6.1 Namespace Scoping

The scope of a namespace declaration declaring a prefix extends from the beginning of the start-tag in which it appears to the end of the corresponding end-tag, excluding the scope of any inner declarations with the same NSAttName part. In the case of an empty tag, the scope is the tag itself.

Such a namespace declaration applies to all element and attribute names within its scope whose prefix matches that specified in the declaration.

The expanded name corresponding to a prefixed element or attribute name has the IRI to which the prefix is bound as its namespace name, and the local part as its local name.

```
<?xml version="1.1"?>

<html:html xmlns:html='http://www.w3.org/1999/xhtml'>

  <html:head><html:title>Frobnostication</html:title></
html:head>
  <html:body><html:p>Moved to
    <html:a href='http://frob.example.com'>here.</html:
a></html:p></html:body>
</html:html>
```

Multiple namespace prefixes can be declared as attributes of a single element, as shown in this example:

```
<?xml version="1.1"?>
<!-- both namespace prefixes are available throughout
-->
<bk:book xmlns:bk='urn:loc.gov:books'
         xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <bk:title>Cheaper by the Dozen</bk:title>
    <isbn:number>1568491379</isbn:number>
</bk:book>
```

The attribute value in a namespace declaration for a prefix *MAY* be empty. This has the effect, within the scope of the declaration, of removing any association of the prefix with a namespace name. Further declarations *MAY* re-declare the prefix again:

```
<?xml version="1.1"?>
<x xmlns:n1="http://www.w3.org">
    <n1:a/>               <!-- legal; the prefix n1 is
bound to http://www.w3.org -->
    <x xmlns:n1="">
        <n1:a/>           <!-- illegal; the prefix n1
is not bound here -->
        <x xmlns:n1="http://www.w3.org">
            <n1:a/>       <!-- legal; the prefix n1 is
bound again -->
        </x>
    </x>
</x>
```

6.2 Namespace Defaulting

The scope of a default namespace declaration extends from the beginning of the start-tag in which it appears to the end of the corresponding end-tag, excluding the scope of any inner default namespace declarations. In the case of an empty tag, the scope is the tag itself.

A default namespace declaration applies to all unprefixed element names within its scope. Default namespace declarations do not apply directly to attribute names; the interpretation of unprefixed attributes is determined by the element on which they appear.

If there is a default namespace declaration in scope, the expanded name corresponding to an unprefixed element name has the IRI of the default namespace as its namespace name. If there is no default namespace declaration in scope, the namespace name has no value. The namespace name for an unprefixed attribute name always has no value. In all cases, the local name is local part (which is of course the same as the unprefixed name itself).

```
<?xml version="1.1"?>
<!-- elements are in the HTML namespace, in this case
by default -->
<html xmlns='http://www.w3.org/1999/xhtml'>
  <head><title>Frobnostication</title></head>
  <body><p>Moved to
    <a href='http://frob.example.com'>here</a>.</p></
body>
</html>
```

```
<?xml version="1.1"?>
<!-- unprefixed element types are from "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <title>Cheaper by the Dozen</title>
    <isbn:number>1568491379</isbn:number>
</book>
```

A larger example of namespace scoping:

```
<?xml version="1.1"?>
<!-- initially, the default namespace is "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <title>Cheaper by the Dozen</title>
    <isbn:number>1568491379</isbn:number>
    <notes>
      <!-- make HTML the default namespace for some
commentary -->
      <p xmlns='http://www.w3.org/1999/xhtml'>
          This is a <i>funny</i> book!
      </p>
    </notes>
</book>
```

The attribute value in a default namespace declaration *MAY* be empty. This has the same effect, within the scope of the declaration, of there being no default namespace.

```
<?xml version='1.1'?>
<Beers>
  <!-- the default namespace inside tables is that of
HTML -->
  <table xmlns='http://www.w3.org/1999/xhtml'>
   <th><td>Name</td><td>Origin</td><td>Description</
td></th>
   <tr>
     <!-- no default namespace inside table cells -->
     <td><brandName xmlns="">Huntsman</brandName></td>
     <td><origin xmlns="">Bath, UK</origin></td>
     <td>
       <details xmlns=""><class>Bitter</
class><hop>Fuggles</hop>
         <pro>Wonderful hop, light alcohol, good
summer beer</pro>
         <con>Fragile; excessive variance pub to pub</
con>
         </details>
        </td>
      </tr>
    </table>
  </Beers>
```

6.3 Uniqueness of Attributes

In XML documents conforming to this specification, no tag may contain two attributes which:

1. have identical names, or

2. have qualified names with the same local part and with prefixes which have been bound to namespace names that are identical.

This constraint is equivalent to requiring that no element have two attributes with the same expanded name.

For example, each of the `bad` start-tags is illegal in the following:

```
<!-- http://www.w3.org is bound to n1 and n2 -->
<x xmlns:n1="http://www.w3.org"
   xmlns:n2="http://www.w3.org" >
  <bad a="1"     a="2" />
  <bad n1:a="1"  n2:a="2" />
</x>
```

However, each of the following is legal, the second because the default namespace does not apply to attribute names:

```
<!-- http://www.w3.org is bound to n1 and is the
default -->
<x xmlns:n1="http://www.w3.org"
   xmlns="http://www.w3.org" >
  <good a="1"     b="2" />
  <good a="1"     n1:a="2" />
</x>
```

7 Conformance of Documents

This specification applies to XML 1.1 documents. To conform to this specification, a document *MUST* be well-formed according to the XML 1.1 specification [XML 1.1].

In XML documents which conform to this specification, element and attribute names *MUST* match the production for QName and *MUST* satisfy the "Namespace Constraints". All other tokens in the document which are *REQUIRED*, for XML 1.1 well-formedness, to match the XML production for Name, *MUST* match this specification's production for NCName.

[Definition: A document is **namespace-well-formed** if it conforms to this specification.]

It follows that in a namespace-well-formed document:

- All element and attribute names contain either zero or one colon;
- No entity names, processing instruction targets, or notation names contain any colons.

In addition, a namespace-well-formed document may also be namespace-valid.

[Definition: A namespace-well-formed document is **namespace-valid** if it is valid according to the XML 1.1 specification, and all tokens other than element and attribute names which are *REQUIRED*, for XML 1.1 validity, to match the XML production for Name, match this specification's production for NCName.]

It follows that in a namespace-valid document:

- No attributes with a declared type of **ID**, **IDREF(S)**, **ENTITY(IES)**, or **NOTATION** contain any colons.

8 Conformance of Processors

To conform to this specification, a processor *MUST* report violations of namespace well-formedness, with the exception that it is not *REQUIRED* to check that namespace names are legal IRIs.

[Definition: A validating XML processor that conforms to this specification is **namespace-validating** if in addition it reports violations of namespace validity.]

9 Internationalized Resource Identifiers (IRIs)

Work is currently in progress to produce an RFC defining Internationalized Resource Identifiers (IRIs). Since this work is not yet complete, this section gives a syntactic definition of IRIs for the purposes of this specification. The XML Core Working Group expects to issue an erratum replacing this section with a reference to the RFC when it is published.

Users defining namespaces are advised to restrict namespace names to URIs until the RFC is published and software supporting IRIs is in common use. Implementors are likewise advised not to reject namespace names that violate the drafts in terms of the allowed characters.

For a more general definition and discussion of IRIs see [IRI draft 5] (work in progress).

URI references are restricted to a subset of the ASCII characters; IRI references allow most Unicode characters from #xA0 onwards. Earlier drafts of the IRI RFC (eg [IRI draft 3]) also allowed some of the disallowed ASCII characters, but the current draft ([IRI draft 5]) does not.

[Definition: The **additional characters** allowed in IRIs by [IRI draft 5] are:]

- the Unicode plane 0 characters #xA0 - #xD7FF, #xF900-#xFDCF, #xFDF0-#xFFEF
- the Unicode plane 1-14 characters #x10000-#x1FFFD ... #xD0000-#xDFFFD, #xE1000-#xEFFFD

[Definition: An **IRI reference** is a string that can be converted to a URI reference by applying the following steps:]

1. Convert the hostname part, if present, using the ToASCII operation specified in Section 4.1 of [RFC3490] with the flags UseSTD3ASCIIRules and AllowUnassigned set to TRUE.
2. Escape all additional characters as follows:

    1. Each additional character is converted to UTF-8 [RFC3629] as one or more bytes.
    2. The resulting bytes are escaped with the URI escaping mechanism (that is, converted to %HH, where HH is the hexadecimal notation of the byte value).
    3. The original character is replaced by the resulting character sequence.

Note:

The algorithm in [IRI draft 5] includes a UCS normalization step, but this makes no difference to which strings are IRI references.

A Normative References

Keywords
RFC 2119: Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, ed. IETF (Internet Engineering Task Force), March 1997. Available at http://www.rfc-editor.org/rfc/rfc2119.txt

RFC2141
RFC 2141: URN Syntax, R. Moats, ed. IETF (Internet Engineering Task Force), May 1997. Available at http://www.rfc-editor.org/rfc/rfc2141.txt.

RFC2396
RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, and L. Masinter, eds. IETF (Internet Engineering Task Force), August 1998. Available at http://www.rfc-editor.org/rfc/rfc2396.txt

RFC2732
RFC 2732: Format for Literal IPv6 Addresses in URL's, R. Hinden, B. Carpenter, and L. Masinter, eds. IETF (Internet Engineering Task

Force), December 1999. Available at http://www.rfc-editor.org/rfc/rfc2732.txt.

RFC3490

RFC 3490: Internationalizing Domain Names in Applications (IDNA), P. Faltstrom, P. Hoffman, and A. Costello, eds. IETF (Internet Engineering Task Force), March 2003. Available at http://www.rfc-editor.org/rfc/rfc3490.txt

RFC3629

RFC 3629: UTF-8, a transformation format of ISO 10646, F. Yergeau, ed. IETF (Internet Engineering Task Force), November 2003. Available at http://www.rfc-editor.org/rfc/rfc3629.txt

XML

Extensible Markup Language (XML) 1.0 (Third Edition), Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau eds. W3C (World Wide Web Consortium), 4 February 2004. Available at http://www.w3.org/TR/REC-xml.

XML 1.1

Extensible Markup Language (XML) 1.1, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and John Cowan eds. W3C (World Wide Web Consortium), 4 February 2004. Available at http://www.w3.org/TR/xml11.

B Other references (Non-Normative)

IRI draft 3

Internationalized Resource Identifiers (IRIs), M. Duerst and M. Suignard eds. March 2, 2003. Available at http://www.w3.org/International/iri-edit/draft-duerst-iri-03.txt.

IRI draft 5

Internationalized Resource Identifiers (IRIs), M. Duerst and M. Suignard eds. October 26, 2003. Available at http://www.w3.org/International/iri-edit/draft-duerst-iri-05.txt.

1.0 Errata

Namespaces in XML Errata. W3C (World Wide Web Consortium). Available at http://www.w3.org/XML/xml-names-19990114-errata.

Relative URI deprecation

Results of W3C XML Plenary Ballot on relative URI References In namespace declarations 3-17 July 2000, Dave Hollander and C. M. Sperberg-McQueen, 6 September 2000. Available at http://www.w3.org/2000/09/xppa.

Requirements

Namespaces in XML 1.1 Requirements, Jonathan Marsh, ed. W3C (World Wide Web Consortium), March 2002. Available at http://www.w3.org/TR/2002/WD-xml-names11-req-20020403/.

C The Internal Structure of XML Namespaces (Non-Normative)

This appendix has been deleted.

D Changes since version 1.0 (Non-Normative)

This version incorporates the errata to version 1.0 as of 6 December 2002 [1.0 Errata]. There are two further substantive changes:

- A mechanism is provided for undeclaring prefixes;
- Namespace names are IRIs, rather than URIs.

There are several editorial changes, including a number of terminology changes and additions intended to produce greater consistency. The non-normative appendix "The Internal Structure of XML Namespaces" has been removed.

E Acknowledgements (Non-Normative)

This work reflects input from a very large number of people, including especially the participants in the World Wide Web Consortium XML Working Group and Special Interest Group and the participants in the W3C Metadata Activity. The contributions of Charles Frankston of Microsoft were particularly valuable.

XML Schema Part 0: Primer

W3C Recommendation, 2 May 2001

This version:
http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/
Latest version:
http://www.w3.org/TR/xmlschema-0/
Previous version:
http://www.w3.org/TR/2001/PR-xmlschema-0-20010330/
Editor:
David C. Fallside (IBM) fallside@us.ibm.com

Abstract

XML Schema Part 0: Primer is a non-normative document intended to provide an easily readable description of the XML Schema facilities, and is oriented towards quickly understanding how to create schemas using the XML Schema language. XML Schema Part 1: Structures and XML Schema Part 2: Datatypes provide the complete normative description of the XML Schema language. This primer describes the language features through numerous examples which are complemented by extensive references to the normative texts.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document has been produced by the W3C XML Schema Working Group as part of the

W3C XML Activity. The goals of the XML Schema language are discussed in the XML Schema Requirements document. The authors of this document are the members of the XML Schema Working Group. Different parts of the document have different editors.

This version of this document incorporates some editorial changes from earlier versions.

Please report errors in this document to www-xml-schema-comments@w3.org (archive). The list of known errors in this specification is available at http://www.w3.org/2001/05/xmlschema-errata.

The English version of this specification is the only normative version. Information about translations of this document is available at http://www.w3.org/2001/05/xmlschema-translations.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

Table of contents

Appendices

---

1 Introduction

This document, XML Schema Part 0: Primer, provides an easily approachable description of the XML Schema definition language, and should be used alongside the formal descriptions of the language contained in Parts 1 and 2 of the XML Schema specification. The intended audience of this document includes application developers whose programs read and write schema documents, and schema authors who need to know about the features of the language, especially features that provide functionality above and beyond what is provided by DTDs. The text assumes that you have a basic understanding of XML 1.0 and XML-Namespaces. Each major section of the primer introduces new features of the language, and describes those features in the context of concrete examples.

Section 2 covers the basic mechanisms of XML Schema. It describes how to declare the elements and attributes that appear in XML documents, the distinctions between simple and complex types, defining complex types, the use of simple types for element and attribute values, schema annotation, a simple mechanism for re-using element and attribute definitions, and nil values.

Section 3, the first advanced section in the primer, explains the basics of how namespaces are used in XML and schema documents. This section is important for understanding many of the topics that appear in the other advanced sections.

Section 4, the second advanced section in the primer, describes mechanisms for deriving types from existing types, and for controlling these derivations. The section also describes mechanisms for merging together fragments of a schema from multiple sources, and for element substitution.

Section 5 covers more advanced features, including a mechanism for specifying uniqueness among attributes and elements, a mechanism for using types across namespaces, a mechanism for extending types based on namespaces, and a description of how documents are checked for conformance.

In addition to the sections just described, the primer contains a number of appendices that provide detailed reference information on simple types and a regular expression language.

The primer is a non-normative document, which means that it does not provide a definitive (from the W3C's point of view) specification of the XML Schema language. The examples and other explanatory material in this document are provided to help you understand XML Schema, but they may not always provide definitive answers. In such cases, you will need to refer to the XML Schema specification, and to help you do this, we provide many links pointing to the relevant parts of the specification. More specifically, XML Schema items mentioned in the primer text are linked to an index of element names and attributes, and a summary table of datatypes, both in the primer. The table and the index contain links to the relevant sections of XML Schema parts 1 and 2.

2 Basic Concepts: The Purchase Order

The purpose of a schema is to define a class of XML documents, and so the term "instance document" is often used to describe an XML document that conforms to a particular schema. In fact, neither instances nor schemas need to exist as documents *per se* -- they may exist as streams of bytes sent between applications, as fields in a database record, or as collections of XML Infoset "Information Items" -- but to simplify the primer, we have chosen to always refer to instances and schemas as if they are documents and files.

Let us start by considering an instance document in a file called `po.xml`. It describes a purchase order generated by a home products ordering and billing application:

The Purchase Order, po.xml

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <items>
        <item partNum="872-AA">
```

```
            <productName>Lawnmower</productName>
            <quantity>1</quantity>
            <USPrice>148.95</USPrice>
            <comment>Confirm this is electric</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity>
            <USPrice>39.98</USPrice>
            <shipDate>1999-05-21</shipDate>
        </item>
    </items>
</purchaseOrder>
```

The purchase order consists of a main element, `purchaseOrder`, and the subelements `shipTo`, `billTo`, `comment`, and `items`. These subelements (except `comment`) in turn contain other subelements, and so on, until a subelement such as `USPrice` contains a number rather than any subelements. Elements that contain subelements or carry attributes are said to have complex types, whereas elements that contain numbers (and strings, and dates, etc.) but do not contain any subelements are said to have simple types. Some elements have attributes; attributes always have simple types.

The complex types in the instance document, and some of the simple types, are defined in the schema for purchase orders. The other simple types are defined as part of XML Schema's repertoire of built-in simple types.

Before going on to examine the purchase order schema, we digress briefly to mention the association between the instance document and the purchase order schema. As you can see by inspecting the instance document, the purchase order schema is not mentioned. An instance is not actually required to reference a schema, and although many will, we have chosen to keep this first section simple, and to assume that any processor of the instance document can obtain the purchase order schema without any information from the instance document. In later sections, we will introduce explicit mechanisms for associating instances and schemas.

2.1 The Purchase Order Schema

The purchase order schema is contained in the file `po.xsd`:

The Purchase Order Schema, po.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
  <xsd:documentation xml:lang="en">
   Purchase order schema for Example.com.
   Copyright 2000 Example.com. All rights reserved.
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
```

```
 <xsd:element name="comment" type="xsd:string"/>

 <xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
   <xsd:element name="shipTo" type="USAddress"/>
   <xsd:element name="billTo" type="USAddress"/>
   <xsd:element ref="comment" minOccurs="0"/>
   <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>

 <xsd:complexType name="USAddress">
  <xsd:sequence>
   <xsd:element name="name"   type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city"   type="xsd:string"/>
   <xsd:element name="state"  type="xsd:string"/>
   <xsd:element name="zip"    type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN"
     fixed="US"/>
 </xsd:complexType>

 <xsd:complexType name="Items">
  <xsd:sequence>
   <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="productName" type="xsd:string"/>
      <xsd:element name="quantity">
       <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
         <xsd:maxExclusive value="100"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:element>
      <xsd:element name="USPrice"  type="xsd:decimal"/>
      <xsd:element ref="comment"   minOccurs="0"/>
      <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/
>
     </xsd:sequence>
     <xsd:attribute name="partNum" type="SKU" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <!-- Stock Keeping Unit, a code for identifying products -->
 <xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
   <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
 </xsd:simpleType>
```

```
</xsd:schema>
```

The purchase order schema consists of a schema element and a variety of subelements, most notably element, complexType, and simpleType which determine the appearance of elements and their content in instance documents.

Each of the elements in the schema has a prefix xsd: which is associated with the XML Schema namespace through the declaration, xmlns:xsd="http://www.w3.org/2001/XMLSchema", that appears in the schema element. The prefix xsd: is used by convention to denote the XML Schema namespace, although any prefix can be used. The same prefix, and hence the same association, also appears on the names of built-in simple types, e.g. xsd:string. The purpose of the association is to identify the elements and simple types as belonging to the vocabulary of the XML Schema language rather than the vocabulary of the schema author. For the sake of clarity in the text, we just mention the names of elements and simple types (e.g. simpleType), and omit the prefix.

2.2 Complex Type Definitions, Element & Attribute Declarations

In XML Schema, there is a basic difference between complex types which allow elements in their content and may carry attributes, and simple types which cannot have element content and cannot carry attributes. There is also a major distinction between definitions which create new types (both simple and complex), and declarations which enable elements and attributes with specific names and types (both simple and complex) to appear in document instances. In this section, we focus on defining complex types and declaring the elements and attributes that appear within them.

New complex types are defined using the complexType element and such definitions typically contain a set of element declarations, element references, and attribute declarations. The declarations are not themselves types, but rather an association between a name and the constraints which govern the appearance of that name in documents governed by the associated schema. Elements are declared using the element element, and attributes are declared using the attribute element. For example, USAddress is defined as a complex type, and within the definition of USAddress we see five element declarations and one attribute declaration:

Defining the USAddress Type

```
<xsd:complexType name="USAddress" >
 <xsd:sequence>
  <xsd:element name="name"   type="xsd:string"/>
  <xsd:element name="street" type="xsd:string"/>
  <xsd:element name="city"   type="xsd:string"/>
  <xsd:element name="state"  type="xsd:string"/>
  <xsd:element name="zip"    type="xsd:decimal"/>
 </xsd:sequence>
 <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

The consequence of this definition is that any element appearing in an instance whose type is

declared to be USAddress (e.g. shipTo in po.xml) must consist of five elements and one attribute. These elements must be called name, street, city, state and zip as specified by the values of the declarations' name attributes, and the elements must appear in the same sequence (order) in which they are declared. The first four of these elements will each contain a string, and the fifth will contain a number. The element whose type is declared to be USAddress may appear with an attribute called country which must contain the string US.

The USAddress definition contains only declarations involving the simple types: string, decimal and NMTOKEN. In contrast, the PurchaseOrderType definition contains element declarations involving complex types, e.g. USAddress, although note that both declarations use the same type attribute to identify the type, regardless of whether the type is simple or complex.

Defining PurchaseOrderType

```
<xsd:complexType name="PurchaseOrderType">
 <xsd:sequence>
  <xsd:element name="shipTo" type="USAddress"/>
  <xsd:element name="billTo" type="USAddress"/>
  <xsd:element ref="comment" minOccurs="0"/>
  <xsd:element name="items"  type="Items"/>
 </xsd:sequence>
 <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

In defining PurchaseOrderType, two of the element declarations, for shipTo and billTo, associate different element names with the same complex type, namely USAddress. The consequence of this definition is that any element appearing in an instance document (e.g. po.xml) whose type is declared to be PurchaseOrderType must consist of elements named shipTo and billTo, each containing the five subelements (name, street, city, state and zip) that were declared as part of USAddress. The shipTo and billTo elements may also carry the country attribute that was declared as part of USAddress.

The PurchaseOrderType definition contains an orderDate attribute declaration which, like the country attribute declaration, identifies a simple type. In fact, all attribute declarations must reference simple types because, unlike element declarations, attributes cannot contain other elements or other attributes.

The element declarations we have described so far have each associated a name with an existing type definition. Sometimes it is preferable to use an existing element rather than declare a new element, for example:

```
<xsd:element ref="comment" minOccurs="0"/>
```

This declaration references an existing element, comment, that was declared elsewhere in the purchase order schema. In general, the value of the ref attribute must reference a global element, i.e. one that has been declared under schema rather than as part of a complex type definition. The consequence of this declaration is that an element called comment may appear in an instance document, and its content must be consistent with that element's type, in this case, string.

2.2.1 Occurrence Constraints

The `comment` element is optional within `PurchaseOrderType` because the value of the `minOccurs` attribute in its declaration is 0. In general, an element is required to appear when the value of `minOccurs` is 1 or more. The maximum number of times an element may appear is determined by the value of a `maxOccurs` attribute in its declaration. This value may be a positive integer such as 41, or the term `unbounded` to indicate there is no maximum number of occurrences. The default value for both the `minOccurs` and the `maxOccurs` attributes is 1. Thus, when an element such as `comment` is declared without a `maxOccurs` attribute, the element may not occur more than once. Be sure that if you specify a value for only the `minOccurs` attribute, it is less than or equal to the default value of `maxOccurs`, i.e. it is 0 or 1. Similarly, if you specify a value for only the `maxOccurs` attribute, it must be greater than or equal to the default value of `minOccurs`, i.e. 1 or more. If both attributes are omitted, the element must appear exactly once.

Attributes may appear once or not at all, but no other number of times, and so the syntax for specifying occurrences of attributes is different than the syntax for elements. In particular, attributes can be declared with a `use` attribute to indicate whether the attribute is `required` (see for example, the `partNum` attribute declaration in `po.xsd`), `optional`, or even `prohibited`.

Default values of both attributes and elements are declared using the `default` attribute, although this attribute has a slightly different consequence in each case. When an attribute is declared with a default value, the value of the attribute is whatever value appears as the attribute's value in an instance document; if the attribute does not appear in the instance document, the schema processor provides the attribute with a value equal to that of the `default` attribute. Note that default values for attributes only make sense if the attributes themselves are optional, and so it is an error to specify both a default value and anything other than a value of `optional` for `use`.

The schema processor treats defaulted elements slightly differently. When an element is declared with a default value, the value of the element is whatever value appears as the element's content in the instance document; if the element appears without any content, the schema processor provides the element with a value equal to that of the `default` attribute. However, if the element does not appear in the instance document, the schema processor does not provide the element at all. In summary, the differences between element and attribute defaults can be stated as: Default attribute values apply when attributes are missing, and default element values apply when elements are empty.

The `fixed` attribute is used in both attribute and element declarations to ensure that the attributes and elements are set to particular values. For example, `po.xsd` contains a declaration for the `country` attribute, which is declared with a `fixed` value US. This declaration means that the appearance of a `country` attribute in an instance document is optional (the default value of `use` is `optional`), although if the attribute does appear, its value must be US, and if the attribute does not appear, the schema processor will provide a `country` attribute with the value US. Note that the concepts of a fixed value and a default value are mutually exclusive, and so it is an error for a declaration to contain both `fixed` and `default` attributes.

The values of the attributes used in element and attribute declarations to constrain their

occurrences are summarized in Table 1.

Table 1. Occurrence Constraints for Elements and Attributes

Elements (minOccurs, maxOccurs) fixed, default	Attributes use, fixed, default	Notes
(1, 1) -, -	required, -, -	element/attribute must appear once, it may have any value
(1, 1) 37, -	required, 37, -	element/attribute must appear once, its value must be 37
(2, unbounded) 37, -	n/a	element must appear twice or more, its value must be 37; in general, minOccurs and maxOccurs values may be positive integers, and maxOccurs value may also be "unbounded"
(0, 1) -, -	optional, -, -	element/attribute may appear once, it may have any value
(0, 1) 37, -	optional, 37, -	element/attribute may appear once, if it does appear its value must be 37, if it does not appear its value is 37
(0, 1) -, 37	optional, -, 37	element/attribute may appear once; if it does not appear its value is 37, otherwise its value is that given
(0, 2) -, 37	n/a	element may appear once, twice, or not at all; if the element does not appear it is not provided; if it does appear and it is empty, its value is 37; otherwise its value is that given; in general, minOccurs and maxOccurs values may be positive integers, and maxOccurs value may also be "unbounded"
(0, 0) -, -	prohibited, -, -	element/attribute must not appear

Note that neither minOccurs, maxOccurs, nor use may appear in the declarations of global elements and attributes.

2.2.2 Global Elements & Attributes

Global elements, and global attributes, are created by declarations that appear as the children of the `schema` element. Once declared, a global element or a global attribute can be referenced in one or more declarations using the `ref` attribute as described above. A declaration that references a global element enables the referenced element to appear in the instance document in the context of the referencing declaration. So, for example, the `comment` element appears in `po.xml` at the same level as the `shipTo`, `billTo` and `items` elements because the declaration that references `comment` appears in the complex type definition at the same level as the declarations of the other three elements.

The declaration of a global element also enables the element to appear at the top-level of an

instance document. Hence `purchaseOrder`, which is declared as a global element in `po.xsd`, can appear as the top-level element in `po.xml`. Note that this rationale will also allow a `comment` element to appear as the top-level element in a document like `po.xml`.

There are a number of caveats concerning the use of global elements and attributes. One caveat is that global declarations cannot contain references; global declarations must identify simple and complex types directly. Put concretely, global declarations cannot contain the `ref` attribute, they must use the `type` attribute (or, as we describe shortly, be followed by an anonymous type definition). A second caveat is that cardinality constraints cannot be placed on global declarations, although they can be placed on local declarations that reference global declarations. In other words, global declarations cannot contain the attributes minOccurs, maxOccurs, or use.

2.2.3 Naming Conflicts

We have now described how to define new complex types (e.g. `PurchaseOrderType`), declare elements (e.g. `purchaseOrder`) and declare attributes (e.g. `orderDate`). These activities generally involve naming, and so the question naturally arises: What happens if we give two things the same name? The answer depends upon the two things in question, although in general the more similar are the two things, the more likely there will be a conflict.

Here are some examples to illustrate when same names cause problems. If the two things are both types, say we define a complex type called USStates and a simple type called USStates, there is a conflict. If the two things are a type and an element or attribute, say we define a complex type called USAddress and we declare an element called USAddress, there is no conflict. If the two things are elements within different types (i.e. not global elements), say we declare one element called name as part of the USAddress type and a second element called name as part of the Item type, there is no conflict. (Such elements are sometimes called local element declarations.) Finally, if the two things are both types and you define one and XML Schema has defined the other, say you define a simple type called decimal, there is no conflict. The reason for the apparent contradiction in the last example is that the two types belong to different namespaces. We explore the use of namespaces in schema in a later section.

2.3 Simple Types

The purchase order schema declares several elements and attributes that have simple types. Some of these simple types, such as `string` and `decimal`, are built in to XML Schema, while others are derived from the built-in's. For example, the `partNum` attribute has a type called SKU (Stock Keeping Unit) that is derived from `string`. Both built-in simple types and their derivations can be used in all element and attribute declarations. Table 2 lists all the simple types built in to XML Schema, along with examples of the different types.

Table 2. Simple Types Built In to XML Schema

Simple Type	Examples (delimited by commas)	Notes
string	Confirm this is electric	
normalizedString	Confirm this is electric	see (3)
token	Confirm this is electric	see (4)
byte	-1, 126	see (2)

unsignedByte	0, 126	see (2)
base64Binary	GpM7	
hexBinary	0FB7	
integer	-126789, -1, 0, 1, 126789	see (2)
positiveInteger	1, 126789	see (2)
negativeInteger	-126789, -1	see (2)
nonNegativeInteger	0, 1, 126789	see (2)
nonPositiveInteger	-126789, -1, 0	see (2)
int	-1, 126789675	see (2)
unsignedInt	0, 1267896754	see (2)
long	-1, 12678967543233	see (2)
unsignedLong	0, 12678967543233	see (2)
short	-1, 12678	see (2)
unsignedShort	0, 12678	see (2)
decimal	-1.23, 0, 123.4, 1000.00	see (2)
float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN	equivalent to single-precision 32-bit floating point, NaN is "not a number", see (2)
double	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN	equivalent to double-precision 64-bit floating point, see (2)
boolean	true, false 1, 0	
time	13:20:00.000, 13:20:00.000-05:00	see (2)
dateTime	1999-05-31T13:20:00.000-05:00	May 31st 1999 at 1.20pm Eastern Standard Time which is 5 hours behind Co-Ordinated Universal Time, see (2)
duration	P1Y2M3DT10H30M12.3S	1 year, 2 months, 3 days, 10 hours, 30 minutes, and 12.3 seconds
date	1999-05-31	see (2)
gMonth	--05--	May, see (2) (5)
gYear	1999	1999, see (2) (5)
gYearMonth	1999-02	the month of February 1999, regardless of the number of days, see (2) (5)

gDay	---31	the 31st day, see (2) (5)
gMonthDay	--05-31	every May 31st, see (2) (5)
Name	shipTo	XML 1.0 Name type
QName	po:USAddress	XML Namespace QName
NCName	USAddress	XML Namespace NCName, i.e. a QName without the prefix and colon
anyURI	http://www.example.com/, http://www.example.com/doc.html#ID5	
language	en-GB, en-US, fr	valid values for xml:lang as defined in XML 1.0
ID		XML 1.0 ID attribute type, see (1)
IDREF		XML 1.0 IDREF attribute type, see (1)
IDREFS		XML 1.0 IDREFS attribute type, see (1)
ENTITY		XML 1.0 ENTITY attribute type, see (1)
ENTITIES		XML 1.0 ENTITIES attribute type, see (1)
NOTATION		XML 1.0 NOTATION attribute type, see (1)
NMTOKEN	US, Brésil	XML 1.0 NMTOKEN attribute type, see (1)
NMTOKENS	US UK, Brésil Canada Mexique	XML 1.0 NMTOKENS attribute type, i.e. a whitespace separated list of NMTOKEN's, see (1)

Notes: (1) To retain compatibility between XML Schema and XML 1.0 DTDs, the simple types ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS should only be used in attributes. (2) A value of this type can be represented by more than one lexical format, e.g. 100 and 1.0E2 are both valid float formats representing "one hundred". However, rules have been established for this type that define a canonical lexical format, see XML Schema Part 2. (3) Newline, tab and carriage-return characters in a normalizedString type are converted to space characters before schema processing. (4) As normalizedString, and adjacent space characters are collapsed to a single space character, and leading and trailing spaces are removed. (5) The "g" prefix signals time periods in the Gregorian calender.

New simple types are defined by deriving them from existing simple types (built-in's and derived). In particular, we can derive a new simple type by restricting an existing simple type, in other words, the legal range of values for the new type are a subset of the existing type's range of values. We use the `simpleType` element to define and name the new simple type. We use the `restriction` element to indicate the existing (base) type, and to identify the

"facets" that constrain the range of values. A complete list of facets is provided in Appendix B.

Suppose we wish to create a new type of integer called `myInteger` whose range of values is between 10000 and 99999 (inclusive). We base our definition on the built-in simple type `integer`, whose range of values also includes integers less than 10000 and greater than 99999. To define `myInteger`, we restrict the range of the `integer` base type by employing two facets called `minInclusive` and `maxInclusive`:

Defining myInteger, Range 10000-99999

```
<xsd:simpleType name="myInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="10000"/>
    <xsd:maxInclusive value="99999"/>
  </xsd:restriction>
</xsd:simpleType>
```

The example shows one particular combination of a base type and two facets used to define `myInteger`, but a look at the list of built-in simple types and their facets (Appendix B) should suggest other viable combinations.

The purchase order schema contains another, more elaborate, example of a simple type definition. A new simple type called SKU is derived (by restriction) from the simple type `string`. Furthermore, we constrain the values of SKU using a facet called `pattern` in conjunction with the regular expression "`\d{3}-[A-Z]{2}`" that is read "three digits followed by a hyphen followed by two upper-case ASCII letters":

Defining the Simple Type "SKU"

```
<xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

This regular expression language is described more fully in Appendix D.

XML Schema defines fifteen facets which are listed in Appendix B. Among these, the `enumeration` facet is particularly useful and it can be used to constrain the values of almost every simple type, except the `boolean` type. The `enumeration` facet limits a simple type to a set of distinct values. For example, we can use the `enumeration` facet to define a new simple type called `USState`, derived from `string`, whose value must be one of the standard US state abbreviations:

Using the Enumeration Facet

```
<xsd:simpleType name="USState">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AK"/>
    <xsd:enumeration value="AL"/>
```

```
      <xsd:enumeration value="AR"/>
      <!-- and so on ... -->
    </xsd:restriction>
  </xsd:simpleType>
```

USState would be a good replacement for the string type currently used in the state element declaration. By making this replacement, the legal values of a state element, i.e. the state subelements of billTo and shipTo, would be limited to one of AK, AL, AR, etc. Note that the enumeration values specified for a particular type must be unique.

2.3.1 List Types

XML Schema has the concept of a list type, in addition to the so-called atomic types that constitute most of the types listed in Table 2. (Atomic types, list types, and the union types described in the next section are collectively called simple types.) The value of an atomic type is indivisible from XML Schema's perspective. For example, the NMTOKEN value US is indivisible in the sense that no part of US, such as the character "S", has any meaning by itself. In contrast, list types are comprised of sequences of atomic types and consequently the parts of a sequence (the "atoms") themselves are meaningful. For example, NMTOKENS is a list type, and an element of this type would be a white-space delimited list of NMTOKEN's, such as "US UK FR". XML Schema has three built-in list types, they are NMTOKENS, IDREFS, and ENTITIES.

In addition to using the built-in list types, you can create new list types by derivation from existing atomic types. (You cannot create list types from existing list types, nor from complex types.) For example, to create a list of myInteger's:

Creating a List of myInteger's

```
  <xsd:simpleType name="listOfMyIntType">
    <xsd:list itemType="myInteger"/>
  </xsd:simpleType>
```

And an element in an instance document whose content conforms to listOfMyIntType is:

```
  <listOfMyInt>20003 15037 95977 95945</listOfMyInt>
```

Several facets can be applied to list types: length, minLength, maxLength, and enumeration. For example, to define a list of exactly six US states (SixUSStates), we first define a new list type called USStateList from USState, and then we derive SixUSStates by restricting USStateList to only six items:

List Type for Six US States

```
  <xsd:simpleType name="USStateList">
   <xsd:list itemType="USState"/>
  </xsd:simpleType>

  <xsd:simpleType name="SixUSStates">
```

```
  <xsd:restriction base="USStateList">
   <xsd:length value="6"/>
  </xsd:restriction>
 </xsd:simpleType>
```

Elements whose type is `SixUSStates` must have six items, and each of the six items must be one of the (atomic) values of the enumerated type `USState`, for example:

```
<sixStates>PA NY CA NY LA AK</sixStates>
```

Note that it is possible to derive a list type from the atomic type `string`. However, a `string` may contain white space, and white space delimits the items in a list type, so you should be careful using list types whose base type is `string`. For example, suppose we have defined a list type with a `length` facet equal to 3, and base type `string`, then the following 3 item list is legal:

```
Asie Europe Afrique
```

But the following 3 "item" list is illegal:

```
Asie Europe Amérique Latine
```

Even though "Amérique Latine" may exist as a single string outside of the list, when it is included in the list, the whitespace between Amérique and Latine effectively creates a fourth item, and so the latter example will not conform to the 3-item list type.

2.3.2 Union Types

Atomic types and list types enable an element or an attribute value to be one or more instances of one atomic type. In contrast, a union type enables an element or attribute value to be one or more instances of one type drawn from the union of multiple atomic and list types. To illustrate, we create a union type for representing American states as singleton letter abbreviations or lists of numeric codes. The `zipUnion` union type is built from one atomic type and one list type:

Union Type for Zipcodes

```
<xsd:simpleType name="zipUnion">
  <xsd:union memberTypes="USState listOfMyIntType"/>
</xsd:simpleType>
```

When we define a union type, the `memberTypes` attribute value is a list of all the types in the union.

Now, assuming we have declared an element called `zips` of type `zipUnion`, valid instances of the element are:

```
<zips>CA</zips>

<zips>95630 95977 95945</zips>

<zips>AK</zips>
```

Two facets, pattern and enumeration, can be applied to a union type.

2.4 Anonymous Type Definitions

Schemas can be constructed by defining sets of named types such as PurchaseOrderType and then declaring elements such as purchaseOrder that reference the types using the type= construction. This style of schema construction is straightforward but it can be unwieldy, especially if you define many types that are referenced only once and contain very few constraints. In these cases, a type can be more succinctly defined as an anonymous type which saves the overhead of having to be named and explicitly referenced.

The definition of the type Items in po.xsd contains two element declarations that use anonymous types (item and quantity). In general, you can identify anonymous types by the lack of a type= in an element (or attribute) declaration, and by the presence of an un-named (simple or complex) type definition:

Two Anonymous Type Definitions

```
<xsd:complexType name="Items">
 <xsd:sequence>
  <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="productName" type="xsd:string"/>
     <xsd:element name="quantity">
      <xsd:simpleType>
       <xsd:restriction base="xsd:positiveInteger">
        <xsd:maxExclusive value="100"/>
       </xsd:restriction>
      </xsd:simpleType>
     </xsd:element>
     <xsd:element name="USPrice"  type="xsd:decimal"/>
     <xsd:element ref="comment"   minOccurs="0"/>
     <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="partNum" type="SKU" use="required"/>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

In the case of the item element, it has an anonymous complex type consisting of the elements productName, quantity, USPrice, comment, and shipDate, and an attribute called partNum. In the case of the quantity element, it has an anonymous simple type

derived from integer whose value ranges between 1 and 99.

2.5 Element Content

The purchase order schema has many examples of elements containing other elements (e.g. items), elements having attributes and containing other elements (e.g. shipTo), and elements containing only a simple type of value (e.g. USPrice). However, we have not seen an element having attributes but containing only a simple type of value, nor have we seen an element that contains other elements mixed with character content, nor have we seen an element that has no content at all. In this section we'll examine these variations in the content models of elements.

2.5.1 Complex Types from Simple Types

Let us first consider how to declare an element that has an attribute and contains a simple value. In an instance document, such an element might appear as:

```
<internationalPrice currency="EUR">423.46</internationalPrice>
```

The purchase order schema declares a USPrice element that is a starting point:

```
<xsd:element name="USPrice" type="decimal"/>
```

Now, how do we add an attribute to this element? As we have said before, simple types cannot have attributes, and decimal is a simple type. Therefore, we must define a complex type to carry the attribute declaration. We also want the content to be simple type decimal. So our original question becomes: How do we define a complex type that is based on the simple type decimal? The answer is to *derive* a new complex type from the simple type decimal:

Deriving a Complex Type from a Simple Type

```
<xsd:element name="internationalPrice">
 <xsd:complexType>
  <xsd:simpleContent>
   <xsd:extension base="xsd:decimal">
    <xsd:attribute name="currency" type="xsd:string"/>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
</xsd:element>
```

We use the complexType element to start the definition of a new (anonymous) type. To indicate that the content model of the new type contains only character data and no elements, we use a simpleContent element. Finally, we derive the new type by extending the simple decimal type. The extension consists of adding a currency attribute using a standard attribute declaration. (We cover type derivation in detail in Section 4.) The internationalPrice element declared in this way will appear in an instance as shown in

the example at the beginning of this section.

2.5.2 Mixed Content

The construction of the purchase order schema may be characterized as elements containing subelements, and the deepest subelements contain character data. XML Schema also provides for the construction of schemas where character data can appear alongside subelements, and character data is not confined to the deepest subelements.

To illustrate, consider the following snippet from a customer letter that uses some of the same elements as the purchase order:

Snippet of Customer Letter

```
<letterBody>
<salutation>Dear Mr.<name>Robert Smith</name>.</salutation>
Your order of <quantity>1</quantity> <productName>Baby
Monitor</productName> shipped from our warehouse on
<shipDate>1999-05-21</shipDate>. ....
</letterBody>
```

Notice the text appearing between elements and their child elements. Specifically, text appears between the elements `salutation`, `quantity`, `productName` and `shipDate` which are all children of `letterBody`, and text appears around the element name which is the child of a child of `letterBody`. The following snippet of a schema declares `letterBody`:

Snippet of Schema for Customer Letter

```
<xsd:element name="letterBody">
 <xsd:complexType mixed="true">
  <xsd:sequence>
   <xsd:element name="salutation">
    <xsd:complexType mixed="true">
     <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:element name="quantity"    type="xsd:positiveInteger"/>
   <xsd:element name="productName" type="xsd:string"/>
   <xsd:element name="shipDate"    type="xsd:date" minOccurs="0"/
>
   <!-- etc. -->
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

The elements appearing in the customer letter are declared, and their types are defined using the `element` and `complexType` element constructions we have seen before. To enable character data to appear between the child-elements of `letterBody`, the `mixed` attribute on the type definition is set to true.

Note that the `mixed` model in XML Schema differs fundamentally from the `mixed` model in XML 1.0. Under the XML Schema mixed model, the order and number of child elements appearing in an instance must agree with the order and number of child elements specified in the model. In contrast, under the XML 1.0 mixed model, the order and number of child elements appearing in an instance cannot be constrained. In summary, XML Schema provides full validation of mixed models in contrast to the partial schema validation provided by XML 1.0.

2.5.3 Empty Content

Now suppose that we want the `internationalPrice` element to convey both the unit of currency and the price as attribute values rather than as separate attribute and content values. For example:

```
<internationalPrice currency="EUR" value="423.46"/>
```

Such an element has no content at all; its content model is empty. To define a type whose content is empty, we essentially define a type that allows only elements in its content, but we do not actually declare any elements and so the type's content model is empty:

An Empty Complex Type

```
<xsd:element name="internationalPrice">
 <xsd:complexType>
  <xsd:complexContent>
   <xsd:restriction base="xsd:anyType">
    <xsd:attribute name="currency" type="xsd:string"/>
    <xsd:attribute name="value"    type="xsd:decimal"/>
   </xsd:restriction>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
```

In this example, we define an (anonymous) type having `complexContent`, i.e. only elements. The `complexContent` element signals that we intend to restrict or extend the content model of a complex type, and the `restriction` of `anyType` declares two attributes but does not introduce any element content (see Section 4.4 for more details on restriction). The `internationalPrice` element declared in this way may legitimately appear in an instance as shown in the example above.

The preceding syntax for an empty-content element is relatively verbose, and it is possible to declare the `internationalPrice` element more compactly:

Shorthand for an Empty Complex Type

```
<xsd:element name="internationalPrice">
 <xsd:complexType>
  <xsd:attribute name="currency" type="xsd:string"/>
  <xsd:attribute name="value"    type="xsd:decimal"/>
 </xsd:complexType>
```

```
  </xsd:element>
```

This compact syntax works because a complex type defined without any `simpleContent` or `complexContent` is interpreted as shorthand for complex content that restricts `anyType`.

2.5.4 anyType

The `anyType` represents an abstraction called the `ur-type` which is the base type from which all simple and complex types are derived. An `anyType` type does not constrain its content in any way. It is possible to use `anyType` like other types, for example:

```
  <xsd:element name="anything" type="xsd:anyType"/>
```

The content of the element declared in this way is unconstrained, so the element value may be 423.46, but it may be any other sequence of characters as well, or indeed a mixture of characters and elements. In fact, `anyType` is the default type when none is specified, so the above could also be written as follows:

```
  <xsd:element name="anything"/>
```

If unconstrained element content is needed, for example in the case of elements containing prose which requires embedded markup to support internationalization, then the default declaration or a slightly restricted form of it may be suitable. The `text` type described in Section 5.5 is an example of such a type that is suitable for such purposes.

2.6 Annotations

XML Schema provides three elements for annotating schemas for the benefit of both human readers and applications. In the purchase order schema, we put a basic schema description and copyright information inside the `documentation` element, which is the recommended location for human readable material. We recommend you use the `xml:lang` attribute with any `documentation` elements to indicate the language of the information. Alternatively, you may indicate the language of all information in a schema by placing an `xml:lang` attribute on the `schema` element.

The `appInfo` element, which we did not use in the purchase order schema, can be used to provide information for tools, stylesheets and other applications. An interesting example using `appInfo` is a schema that describes the simple types in XML Schema Part 2: Datatypes. Information describing this schema, e.g. which facets are applicable to particular simple types, is represented inside `appInfo` elements, and this information was used by an application to automatically generate text for the XML Schema Part 2 document.

Both `documentation` and `appInfo` appear as subelements of `annotation`, which may itself appear at the beginning of most schema constructions. To illustrate, the following example shows `annotation` elements appearing at the beginning of an element declaration and a complex type definition:

```
```

Annotations in Element Declaration & Complex Type Definition

```
<xsd:element name="internationalPrice">
 <xsd:annotation>
  <xsd:documentation xml:lang="en">
      element declared with anonymous type
  </xsd:documentation>
 </xsd:annotation>
 <xsd:complexType>
  <xsd:annotation>
   <xsd:documentation xml:lang="en">
       empty anonymous type with 2 attributes
   </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
   <xsd:restriction base="xsd:anyType">
    <xsd:attribute name="currency" type="xsd:string"/>
    <xsd:attribute name="value"    type="xsd:decimal"/>
   </xsd:restriction>
  </xsd:complexContent>
 </xsd:complexType>
</xsd:element>
```

The `annotation` element may also appear at the beginning of other schema constructions such as those indicated by the elements `schema`, `simpleType`, and `attribute`.

2.7 Building Content Models

The definitions of complex types in the purchase order schema all declare sequences of elements that must appear in the instance document. The occurrence of individual elements declared in the so-called content models of these types may be optional, as indicated by a 0 value for the attribute `minOccurs` (e.g. in `comment`), or be otherwise constrained depending upon the values of `minOccurs` and `maxOccurs`. XML Schema also provides constraints that apply to groups of elements appearing in a content model. These constraints mirror those available in XML 1.0 plus some additional constraints. Note that the constraints do not apply to attributes.

XML Schema enables groups of elements to be defined and named, so that the elements can be used to build up the content models of complex types (thus mimicking common usage of parameter entities in XML 1.0). Un-named groups of elements can also be defined, and along with elements in named groups, they can be constrained to appear in the same order (sequence) as they are declared. Alternatively, they can be constrained so that only one of the elements may appear in an instance.

To illustrate, we introduce two groups into the `PurchaseOrderType` definition from the purchase order schema so that purchase orders may contain either separate shipping and billing addresses, or a single address for those cases in which the shippee and billee are co-located:

Nested Choice and Sequence Groups

```
<xsd:complexType name="PurchaseOrderType">
```

```
 <xsd:sequence>
  <xsd:choice>
   <xsd:group   ref="shipAndBill"/>
   <xsd:element name="singleUSAddress" type="USAddress"/>
  </xsd:choice>
  <xsd:element ref="comment" minOccurs="0"/>
  <xsd:element name="items"  type="Items"/>
 </xsd:sequence>
 <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>

<xsd:group name="shipAndBill">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
  </xsd:sequence>
</xsd:group>
```

The choice group element allows only one of its children to appear in an instance. One child is an inner group element that references the named group shipAndBill consisting of the element sequence shipTo, billTo, and the second child is a singleUSAddress. Hence, in an instance document, the purchaseOrder element must contain either a shipTo element followed by a billTo element or a singleUSAddress element. The choice group is followed by the comment and items element declarations, and both the choice group and the element declarations are children of a sequence group. The effect of these various groups is that the address element(s) must be followed by comment and items elements in that order.

There exists a third option for constraining elements in a group: All the elements in the group may appear once or not at all, and they may appear in any order. The all group (which provides a simplified version of the SGML &-Connector) is limited to the top-level of any content model. Moreover, the group's children must all be individual elements (no groups), and no element in the content model may appear more than once, i.e. the permissible values of minOccurs and maxOccurs are 0 and 1. For example, to allow the child elements of purchaseOrder to appear in any order, we could redefine PurchaseOrderType as:

An 'All' Group

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:all>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items"  type="Items"/>
  </xsd:all>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

By this definition, a comment element may optionally appear within purchaseOrder, and it may appear before or after any shipTo, billTo and items elements, but it can appear only once. Moreover, the stipulations of an all group do not allow us to declare an element such

as `comment` outside the group as a means of enabling it to appear more than once. XML Schema stipulates that an `all` group must appear as the sole child at the top of a content model. In other words, the following is illegal:

Illegal Example with an 'All' Group

```
<xsd:complexType name="PurchaseOrderType">
 <xsd:sequence>
  <xsd:all>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element name="items"  type="Items"/>
  </xsd:all>
  <xsd:sequence>
   <xsd:element ref="comment" minOccurs="0"
maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:sequence>
 <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

Finally, named and un-named groups that appear in content models (represented by `group` and `choice`, `sequence`, `all` respectively) may carry `minOccurs` and `maxOccurs` attributes. By combining and nesting the various groups provided by XML Schema, and by setting the values of `minOccurs` and `maxOccurs`, it is possible to represent any content model expressible with an XML 1.0 DTD. Furthermore, the `all` group provides additional expressive power.

2.8 Attribute Groups

Suppose we want to provide more information about each item in a purchase order, for example, each item's weight and preferred shipping method. We can accomplish this by adding `weightKg` and `shipBy` attribute declarations to the `item` element's (anonymous) type definition:

Adding Attributes to the Inline Type Definition

```
<xsd:element name="Item" minOccurs="0" maxOccurs="unbounded">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element   name="productName" type="xsd:string"/>
    <xsd:element   name="quantity">
     <xsd:simpleType>
      <xsd:restriction base="xsd:positiveInteger">
       <xsd:maxExclusive value="100"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:element>
    <xsd:element name="USPrice"  type="xsd:decimal"/>
    <xsd:element ref="comment"   minOccurs="0"/>
    <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
   </xsd:sequence>
```

```
    <xsd:attribute name="partNum"  type="SKU" use="required"/>
    <!-- add weightKg and shipBy attributes -->
    <xsd:attribute name="weightKg" type="xsd:decimal"/>
    <xsd:attribute name="shipBy">
     <xsd:simpleType>
      <xsd:restriction base="xsd:string">
       <xsd:enumeration value="air"/>
       <xsd:enumeration value="land"/>
       <xsd:enumeration value="any"/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:attribute>
   </xsd:complexType>
  </xsd:element>
```

Alternatively, we can create a named attribute group containing all the desired attributes of an `item` element, and reference this group by name in the `item` element declaration:

Adding Attributes Using an Attribute Group

```
  <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="productName" type="xsd:string"/>
     <xsd:element name="quantity">
      <xsd:simpleType>
       <xsd:restriction base="xsd:positiveInteger">
        <xsd:maxExclusive value="100"/>
       </xsd:restriction>
      </xsd:simpleType>
     </xsd:element>
     <xsd:element name="USPrice"  type="xsd:decimal"/>
     <xsd:element ref="comment"   minOccurs="0"/>
     <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
    </xsd:sequence>

    <!-- attributeGroup replaces individual declarations -->
    <xsd:attributeGroup ref="ItemDelivery"/>
   </xsd:complexType>
  </xsd:element>

  <xsd:attributeGroup name="ItemDelivery">
    <xsd:attribute name="partNum"  type="SKU" use="required"/>
    <xsd:attribute name="weightKg" type="xsd:decimal"/>
    <xsd:attribute name="shipBy">
      <xsd:simpleType>
       <xsd:restriction base="xsd:string">
        <xsd:enumeration value="air"/>
        <xsd:enumeration value="land"/>
        <xsd:enumeration value="any"/>
       </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
```

```
  </xsd:attributeGroup>
```

Using an attribute group in this way can improve the readability of schemas, and facilitates updating schemas because an attribute group can be defined and edited in one place and referenced in multiple definitions and declarations. These characteristics of attribute groups make them similar to parameter entities in XML 1.0. Note that an attribute group may contain other attribute groups. Note also that both attribute declarations and attribute group references must appear at the end of complex type definitions.

2.9 Nil Values

One of the purchase order items listed in `po.xml`, the `Lawnmower`, does not have a `shipDate` element. Within the context of our scenario, the schema author may have intended such absences to indicate `items` not yet shipped. But in general, the absence of an element does not have any particular meaning: It may indicate that the information is unknown, or not applicable, or the element may be absent for some other reason. Sometimes it is desirable to represent an unshipped `item`, unknown information, or inapplicable information *explicitly* with an element, rather than by an absent element. For example, it may be desirable to represent a "null" value being sent to or from a relational database with an element that is present. Such cases can be represented using XML Schema's nil mechanism which enables an element to appear with or without a non-nil value.

XML Schema's nil mechanism involves an "out of band" nil signal. In other words, there is no actual nil value that appears as element content, instead there is an attribute to indicate that the element content is nil. To illustrate, we modify the `shipDate` element declaration so that nils can be signalled:

```
  <xsd:element name="shipDate" type="xsd:date" nillable="true"/>
```

And to explicitly represent that `shipDate` has a nil value in the instance document, we set the nil attribute (from the XML Schema namespace for instances) to true:

```
  <shipDate xsi:nil="true"></shipDate>
```

The `nil` attribute is defined as part of the XML Schema namespace for instances, `http://www.w3.org/2001/XMLSchema-instance`, and so it must appear in the instance document with a prefix (such as `xsi:`) associated with that namespace. (As with the `xsd:` prefix, the `xsi:` prefix is used by convention only.) Note that the nil mechanism applies only to element values, and not to attribute values. An element with `xsi:nil="true"` may not have any element content but it may still carry attributes.

3. Advanced Concepts I: Namespaces, Schemas & Qualification

A schema can be viewed as a collection (vocabulary) of type definitions and element declarations whose names belong to a particular namespace called a target namespace. Target namespaces enable us to distinguish between definitions and declarations from different vocabularies. For example, target namespaces would enable us to distinguish

between the declaration for `element` in the XML Schema language vocabulary, and a declaration for `element` in a hypothetical chemistry language vocabulary. The former is part of the `http://www.w3.org/2001/XMLSchema` target namespace, and the latter is part of another target namespace.

When we want to check that an instance document conforms to one or more schemas (through a process called schema validation), we need to identify which element and attribute declarations and type definitions in the schemas should be used to check which elements and attributes in the instance document. The target namespace plays an important role in the identification process. We examine the role of the target namespace in the next section.

The schema author also has several options that affect how the identities of elements and attributes are represented in instance documents. More specifically, the author can decide whether or not the appearance of locally declared elements and attributes in an instance must be qualified by a namespace, using either an explicit prefix or implicitly by default. The schema author's choice regarding qualification of local elements and attributes has a number of implications regarding the structures of schemas and instance documents, and we examine some of these implications in the following sections.

3.1 Target Namespaces & Unqualified Locals

In a new version of the purchase order schema, `po1.xsd`, we explicitly declare a target namespace, and specify that both locally defined elements and locally defined attributes must be unqualified. The target namespace in `po1.xsd` is `http://www.example.com/PO1`, as indicated by the value of the `targetNamespace` attribute.

Qualification of local elements and attributes can be globally specified by a pair of attributes, `elementFormDefault` and `attributeFormDefault`, on the `schema` element, or can be specified separately for each local declaration using the `form` attribute. All such attributes' values may each be set to `unqualified` or `qualified`, to indicate whether or not locally declared elements and attributes must be unqualified.

In `po1.xsd` we globally specify the qualification of elements and attributes by setting the values of both `elementFormDefault` and `attributeFormDefault` to `unqualified`. Strictly speaking, these settings are unnecessary because the values are the defaults for the two attributes; we make them here to highlight the contrast between this case and other cases we describe later.

Purchase Order Schema with Target Namespace, po1.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="unqualified"
        attributeFormDefault="unqualified">

 <element name="purchaseOrder" type="po:PurchaseOrderType"/>
 <element name="comment"       type="string"/>

 <complexType name="PurchaseOrderType">
  <sequence>
   <element name="shipTo"    type="po:USAddress"/>
```

```
    <element name="billTo"    type="po:USAddress"/>
    <element ref="po:comment" minOccurs="0"/>
    <!-- etc. -->
   </sequence>
   <!-- etc. -->
  </complexType>

  <complexType name="USAddress">
   <sequence>
    <element name="name"   type="string"/>
    <element name="street" type="string"/>
    <!-- etc. -->
   </sequence>
  </complexType>

  <!-- etc. -->

 </schema>
```

To see how the target namespace of this schema is populated, we examine in turn each of the type definitions and element declarations. Starting from the end of the schema, we first define a type called `USAddress` that consists of the elements `name`, `street`, etc. One consequence of this type definition is that the `USAddress` type is included in the schema's target namespace. We next define a type called `PurchaseOrderType` that consists of the elements `shipTo`, `billTo`, `comment`, etc. `PurchaseOrderType` is also included in the schema's target namespace. Notice that the type references in the three element declarations are prefixed, i.e. `po:USAddress`, `po:USAddress` and `po:comment`, and the prefix is associated with the namespace `http://www.example.com/PO1`. This is the same namespace as the schema's target namespace, and so a processor of this schema will know to look within this schema for the definition of the type `USAddress` and the declaration of the element `comment`. It is also possible to refer to types in another schema with a different target namespace, hence enabling re-use of definitions and declarations between schemas.

At the beginning of the schema `po1.xsd`, we declare the elements `purchaseOrder` and `comment`. They are included in the schema's target namespace. The `purchaseOrder` element's type is prefixed, for the same reason that `USAddress` is prefixed. In contrast, the `comment` element's type, `string`, is not prefixed. The `po1.xsd` schema contains a default namespace declaration, and so unprefixed types such as `string` and unprefixed elements such as `element` and `complexType` are associated with the default namespace `http://www.w3.org/2001/XMLSchema`. In fact, this is the target namespace of XML Schema itself, and so a processor of `po1.xsd` will know to look within the schema of XML Schema -- otherwise known as the "schema for schemas" -- for the definition of the type `string` and the declaration of the element called `element`.

Let us now examine how the target namespace of the schema affects a conforming instance document:

A Purchase Order with Unqualified Locals, po1.xml

```
<?xml version="1.0"?>
<apo:purchaseOrder xmlns:apo="http://www.example.com/PO1"
                   orderDate="1999-10-20">
```

```
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <!-- etc. -->
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <!-- etc. -->
    </billTo>
    <apo:comment>Hurry, my lawn is going wild!</apo:comment>
    <!-- etc. -->
</apo:purchaseOrder>
```

The instance document declares one namespace, `http://www.example.com/PO1`, and associates it with the prefix `apo:`. This prefix is used to qualify two elements in the document, namely `purchaseOrder` and `comment`. The namespace is the same as the target namespace of the schema in `po1.xsd`, and so a processor of the instance document will know to look in that schema for the declarations of `purchaseOrder` and `comment`. In fact, target namespaces are so named because of the sense in which there exists a target namespace for the elements `purchaseOrder` and `comment`. Target namespaces in the schema therefore control the validation of corresponding namespaces in the instance.

The prefix `apo:` is applied to the global elements `purchaseOrder` and `comment` elements. Furthermore, `elementFormDefault` and `attributeFormDefault` require that the prefix is *not* applied to any of the locally declared elements such as `shipTo`, `billTo`, `name` and `street`, and it is *not* applied to any of the attributes (which were all declared locally). The `purchaseOrder` and `comment` are global elements because they are declared in the context of the schema as a whole rather than within the context of a particular type. For example, the declaration of `purchaseOrder` appears as a child of the `schema` element in `po1.xsd`, whereas the declaration of `shipTo` appears as a child of the `complexType` element that defines `PurchaseOrderType`.

When local elements and attributes are not required to be qualified, an instance author may require more or less knowledge about the details of the schema to create schema valid instance documents. More specifically, if the author can be sure that only the root element (such as `purchaseOrder`) is global, then it is a simple matter to qualify only the root element. Alternatively, the author may know that all the elements are declared globally, and so all the elements in the instance document can be prefixed, perhaps taking advantage of a default namespace declaration. (We examine this approach in Section 3.3.) On the other hand, if there is no uniform pattern of global and local declarations, the author will need detailed knowledge of the schema to correctly prefix global elements and attributes.

3.2 Qualified Locals

Elements and attributes can be independently required to be qualified, although we start by describing the qualification of local elements. To specify that all locally declared elements in a schema must be qualified, we set the value of `elementFormDefault` to `qualified`:

Modifications to po1.xsd for Qualified Locals

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
```

```
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">

 <element name="purchaseOrder" type="po:PurchaseOrderType"/>
 <element name="comment"       type="string"/>

 <complexType name="PurchaseOrderType">
  <!-- etc. -->
 </complexType>

 <!-- etc. -->

</schema>
```

And in this conforming instance document, we qualify all the elements explicitly:

A Purchase Order with Explicitly Qualified Locals

```
<?xml version="1.0"?>
<apo:purchaseOrder xmlns:apo="http://www.example.com/PO1"
                   orderDate="1999-10-20">
    <apo:shipTo country="US">
        <apo:name>Alice Smith</apo:name>
        <apo:street>123 Maple Street</apo:street>
        <!-- etc. -->
    </apo:shipTo>
    <apo:billTo country="US">
        <apo:name>Robert Smith</apo:name>
        <apo:street>8 Oak Avenue</apo:street>
        <!-- etc. -->
    </apo:billTo>
    <apo:comment>Hurry, my lawn is going wild!</apo:comment>
    <!-- etc. -->
</apo:purchaseOrder>
```

Alternatively, we can replace the explicit qualification of every element with implicit qualification provided by a default namespace, as shown here in po2.xml:

A Purchase Order with Default Qualified Locals, po2.xml

```
<?xml version="1.0"?>
<purchaseOrder xmlns="http://www.example.com/PO1"
               orderDate="1999-10-20">
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <!-- etc. -->
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
```

```
        <street>8 Oak Avenue</street>
        <!-- etc. -->
    </billTo>
    <comment>Hurry, my lawn is going wild!</comment>
    <!-- etc. -->
</purchaseOrder>
```

In po2.xml, all the elements in the instance belong to the same namespace, and the namespace statement declares a default namespace that applies to all the elements in the instance. Hence, it is unnecessary to explicitly prefix any of the elements. As another illustration of using qualified elements, the schemas in Section 5 all require qualified elements.

Qualification of attributes is very similar to the qualification of elements. Attributes that must be qualified, either because they are declared globally or because the `attributeFormDefault` attribute is set to `qualified`, appear prefixed in instance documents. One example of a qualified attribute is the `xsi:nil` attribute that was introduced in Section 2.9. In fact, attributes that are required to be qualified must be explicitly prefixed because the XML-Namespaces specification does not provide a mechanism for defaulting the namespaces of attributes. Attributes that are not required to be qualified appear in instance documents without prefixes, which is the typical case.

The qualification mechanism we have described so far has controlled all local element and attribute declarations within a particular target namespace. It is also possible to control qualification on a declaration by declaration basis using the `form` attribute. For example, to require that the locally declared attribute `publicKey` is qualified in instances, we declare it in the following way:

Requiring Qualification of Single Attribute

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
 <!-- etc. -->
 <element name="secure">
  <complexType>
   <sequence>
    <!-- element declarations -->
   </sequence>
   <attribute name="publicKey" type="base64Binary"
form="qualified"/>
  </complexType>
 </element>
</schema>
```

Notice that the value of the `form` attribute overrides the value of the `attributeFormDefault` attribute for the `publicKey` attribute only. Also, the `form` attribute can be applied to an element declaration in the same manner. An instance document that conforms to the schema is:

Instance with a Qualified Attribute

```
<?xml version="1.0"?>
<purchaseOrder xmlns="http://www.example.com/PO1"
               xmlns:po="http://www.example.com/PO1"
               orderDate="1999-10-20">
    <!-- etc. -->
    <secure po:publicKey="GpM7">
        <!-- etc. -->
    </secure>
</purchaseOrder>
```

3.3 Global vs. Local Declarations

Another authoring style, applicable when all element names are unique within a namespace, is to create schemas in which all elements are global. This is similar in effect to the use of <! ELEMENT> in a DTD. In the example below, we have modified the original `po1.xsd` such that all the elements are declared globally. Notice that we have omitted the `elementFormDefault` and `attributeFormDefault` attributes in this example to emphasize that their values are irrelevant when there are only global element and attribute declarations.

Modified version of po1.xsd using only global element declarations

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:po="http://www.example.com/PO1"
        targetNamespace="http://www.example.com/PO1">

 <element name="purchaseOrder" type="po:PurchaseOrderType"/>

 <element name="shipTo"  type="po:USAddress"/>
 <element name="billTo"  type="po:USAddress"/>
 <element name="comment" type="string"/>

 <element name="name" type="string"/>
 <element name="street" type="string"/>

 <complexType name="PurchaseOrderType">
  <sequence>
   <element ref="po:shipTo"/>
   <element ref="po:billTo"/>
   <element ref="po:comment" minOccurs="0"/>
   <!-- etc. -->
  </sequence>
 </complexType>

 <complexType name="USAddress">
  <sequence>
   <element ref="po:name"/>
   <element ref="po:street"/>
   <!-- etc. -->
  </sequence>
 </complexType>
```

```
  <!-- etc. -->

</schema>
```

This "global" version of po1.xsd will validate the instance document po2.xml which, as we described previously, is also schema valid against the "qualified" version of po1.xsd. In other words, both schema approaches can validate the same, namespace defaulted, document. Thus, in one respect the two schema approaches are similar, although in another important respect the two schema approaches are very different. Specifically, when all elements are declared globally, it is not possible to take advantage of local names. For example, you can only declare one global element called "title". However, you can locally declare one element called "title" that has a string type, and is a subelement of "book". Within the same schema (target namespace) you can declare a second element also called "title" that is an enumeration of the values "Mr Mrs Ms".

3.4 Undeclared Target Namespaces

In Section 2 we explained the basics of XML Schema using a schema that did not declare a target namespace and an instance document that did not declare a namespace. So the question naturally arises: What is the target namespace in these examples and how is it referenced?

In the purchase order schema, po.xsd, we did not declare a target namespace for the schema, nor did we declare a prefix (like `po:` above) associated with the schema's target namespace with which we could refer to types and elements defined and declared within the schema. The consequence of not declaring a target namespace in a schema is that the definitions and declarations from that schema, such as `USAddress` and `purchaseOrder`, are referenced without namespace qualification. In other words there is no explicit namespace prefix applied to the references nor is there any implicit namespace applied to the reference by default. So for example, the `purchaseOrder` element is declared using the type reference `PurchaseOrderType`. In contrast, all the XML Schema elements and types used in po.xsd are explicitly qualified with the prefix `xsd:` that is associated with the XML Schema namespace.

In cases where a schema is designed without a target namespace, it is strongly recommended that all XML Schema elements and types are *explicitly* qualified with a prefix such as `xsd:` that is associated with the XML Schema namespace (as in po.xsd). The rationale for this recommendation is that if XML Schema elements and types are associated with the XML Schema namespace by default, i.e. without prefixes, then references to XML Schema types may not be distinguishable from references to user-defined types.

Element declarations from a schema with no target namespace validate unqualified elements in the instance document. That is, they validate elements for which no namespace qualification is provided by either an explicit prefix or by default (`xmlns:`). So, to validate a traditional XML 1.0 document which does not use namespaces at all, you must provide a schema with no target namespace. Of course, there are many XML 1.0 documents that do not use namespaces, so there will be many schema documents written without target namespaces; you must be sure to give to your processor a schema document that corresponds to the vocabulary you wish to validate.

4. Advanced Concepts II: The International Purchase Order

The purchase order schema described in Chapter 2 was contained in a single document, and most of the schema constructions-- such as element declarations and type definitions-- were constructed from scratch. In reality, schema authors will want to compose schemas from constructions located in multiple documents, and to create new types based on existing types. In this section, we examine mechanisms that enable such compositions and creations.

4.1 A Schema in Multiple Documents

As schemas become larger, it is often desirable to divide their content among several schema documents for purposes such as ease of maintenance, access control, and readability. For these reasons, we have taken the schema constructs concerning addresses out of `po.xsd`, and put them in a new file called `address.xsd`. The modified purchase order schema file is called `ipo.xsd`:

The International Purchase Order Schema, ipo.xsd

```
<schema targetNamespace="http://www.example.com/IPO"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ipo="http://www.example.com/IPO">

 <annotation>
  <documentation xml:lang="en">
   International Purchase order schema for Example.com
   Copyright 2000 Example.com. All rights reserved.
  </documentation>
 </annotation>

 <!-- include address constructs -->
 <include
  schemaLocation="http://www.example.com/schemas/address.xsd"/>

 <element name="purchaseOrder" type="ipo:PurchaseOrderType"/>

 <element name="comment" type="string"/>

 <complexType name="PurchaseOrderType">
  <sequence>
   <element name="shipTo"     type="ipo:Address"/>
   <element name="billTo"     type="ipo:Address"/>
   <element ref="ipo:comment" minOccurs="0"/>
   <element name="items"      type="ipo:Items"/>
  </sequence>
  <attribute name="orderDate" type="date"/>
 </complexType>

 <complexType name="Items">
  <sequence>
   <element name="item" minOccurs="0" maxOccurs="unbounded">
    <complexType>
     <sequence>
      <element name="productName" type="string"/>
```

```
      <element name="quantity">
       <simpleType>
        <restriction base="positiveInteger">
         <maxExclusive value="100"/>
        </restriction>
       </simpleType>
      </element>
      <element name="USPrice"    type="decimal"/>
      <element ref="ipo:comment" minOccurs="0"/>
      <element name="shipDate"   type="date" minOccurs="0"/>
     </sequence>
     <attribute name="partNum" type="ipo:SKU" use="required"/>
    </complexType>
   </element>
  </sequence>
 </complexType>

 <simpleType name="SKU">
  <restriction base="string">
   <pattern value="\d{3}-[A-Z]{2}"/>
  </restriction>
 </simpleType>

</schema>
```

The file containing the address constructs is:

Addresses for International Purchase Order schema, address.xsd

```
<schema targetNamespace="http://www.example.com/IPO"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ipo="http://www.example.com/IPO">

 <annotation>
  <documentation xml:lang="en">
   Addresses for International Purchase order schema
   Copyright 2000 Example.com. All rights reserved.
  </documentation>
 </annotation>

 <complexType name="Address">
  <sequence>
   <element name="name"   type="string"/>
   <element name="street" type="string"/>
   <element name="city"   type="string"/>
  </sequence>
 </complexType>

 <complexType name="USAddress">
  <complexContent>
   <extension base="ipo:Address">
    <sequence>
     <element name="state" type="ipo:USState"/>
```

```
      <element name="zip"   type="positiveInteger"/>
     </sequence>
    </extension>
   </complexContent>
  </complexType>

  <complexType name="UKAddress">
   <complexContent>
    <extension base="ipo:Address">
     <sequence>
      <element name="postcode" type="ipo:UKPostcode"/>
     </sequence>
     <attribute name="exportCode" type="positiveInteger"
fixed="1"/>
    </extension>
   </complexContent>
  </complexType>

  <!-- other Address derivations for more countries -->

  <simpleType name="USState">
   <restriction base="string">
    <enumeration value="AK"/>
    <enumeration value="AL"/>
    <enumeration value="AR"/>
    <!-- and so on ... -->
   </restriction>
  </simpleType>

  <!-- simple type definition for UKPostcode -->

 </schema>
```

The various purchase order and address constructions are now contained in two schema files, `ipo.xsd` and `address.xsd`. To include these constructions as part of the international purchase order schema, in other words to include them in the international purchase order's namespace, `ipo.xsd` contains the `include` element:

```
 <include schemaLocation="http://www.example.com/schemas/address.
 xsd"/>
```

The effect of this `include` element is to bring in the definitions and declarations contained in `address.xsd`, and make them available as part of the international purchase order schema target namespace. The one important caveat to using `include` is that the target namespace of the included components must be the same as the target namespace of the including schema, in this case `http://www.example.com/IPO`. Bringing in definitions and declarations using the `include` mechanism effectively adds these components to the existing target namespace. In Section 4.5, we describe a similar mechanism that enables you to modify certain components when they are brought in.

In our example, we have shown only one including document and one included document. In

practice it is possible to include more than one document using multiple `include` elements, and documents can include documents that themselves include other documents. However, nesting documents in this manner is legal only if all the included parts of the schema are declared with the same target namespace.

Instance documents that conform to schema whose definitions span multiple schema documents need only reference the 'topmost' document and the common namespace, and it is the responsibility of the processor to gather together all the definitions specified in the various included documents. In our example above, the instance document `ipo.xml` (see Section 4.3) references only the common target namespace, `http://www.example.com/IPO`, and (by implication) the one schema file `http://www.example.com/schemas/ipo.xsd`. The processor is responsible for obtaining the schema file `address.xsd`.

In Section 5.4 we describe how schemas can be used to validate content from more than one namespace.

4.2 Deriving Types by Extension

To create our address constructs, we start by creating a complex type called `Address` in the usual way (see `address.xsd`). The `Address` type contains the basic elements of an address: a name, a street and a city. (Such a definition will not work for all countries, but it serves the purpose of our example.) From this starting point we derive two new complex types that contain all the elements of the original type plus additional elements that are specific to addresses in the US and the UK. The technique we use here to derive new (complex) address types by extending an existing type is the same technique we used in in Section 2.5.1, except that our base type here is a complex type whereas our base type in the previous section was a simple type.

We define the two new complex types, `USAddress` and `UKAddress`, using the `complexType` element. In addition, we indicate that the content models of the new types are complex, i.e. contain elements, by using the `complexContent` element, and we indicate that we are extending the base type `Address` by the value of the `base` attribute on the `extension` element.

When a complex type is derived by extension, its effective content model is the content model of the base type plus the content model specified in the type derivation. Furthermore, the two content models are treated as two children of a sequential group. In the case of `UKAddress`, the content model of `UKAddress` is the content model of `Address` plus the declarations for a `postcode` element and an `exportCode` attribute. This is like defining the `UKAddress` from scratch as follows:

Example

```
<complexType name="UKAddress">
 <sequence>
  <!-- content model of Address -->
  <element name="name"   type="string"/>
  <element name="street" type="string"/>
  <element name="city"   type="string"/>

  <!-- appended element declaration -->
  <element name="postcode" type="ipo:UKPostcode"/>
```

```
    </sequence>

    <!-- appended attribute declaration -->
    <attribute name="exportCode" type="positiveInteger" fixed="1"/>
   </complexType>
```

4.3 Using Derived Types in Instance Documents

In our example scenario, purchase orders are generated in response to customer orders which may involve shipping and billing addresses in different countries. The international purchase order, `ipo.xml` below, illustrates one such case where goods are shipped to the UK and the bill is sent to a US address. Clearly it is better if the schema for international purchase orders does not have to spell out every possible combination of international addresses for billing and shipping, and even more so if we can add new complex types of international address simply by creating new derivations of `Address`.

XML Schema allows us to define the `billTo` and `shipTo` elements as `Address` types (see `ipo.xsd`) but to use instances of international addresses in place of instances of `Address`. In other words, an instance document whose content conforms to the `UKAddress` type will be valid if that content appears within the document at a location where an `Address` is expected (assuming the `UKAddress` content itself is valid). To make this feature of XML Schema work, and to identify exactly which derived type is intended, the derived type must be identified in the instance document. The type is identified using the `xsi:type` attribute which is part of the XML Schema instance namespace. In the example, `ipo.xml`, use of the `UKAddress` and `USAddress` derived types is identified through the values assigned to the `xsi:type` attributes.

An International Purchase order, ipo.xml

```
<?xml version="1.0"?>
<ipo:purchaseOrder
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ipo="http://www.example.com/IPO"
  orderDate="1999-12-01">

    <shipTo exportCode="1" xsi:type="ipo:UKAddress">
        <name>Helen Zoe</name>
        <street>47 Eden Street</street>
        <city>Cambridge</city>
        <postcode>CB1 1JR</postcode>
    </shipTo>

    <billTo xsi:type="ipo:USAddress">
        <name>Robert Smith</name>
        <street>8 Oak Avenue</street>
        <city>Old Town</city>
        <state>PA</state>
        <zip>95819</zip>
    </billTo>

    <items>
        <item partNum="833-AA">
```

```
            <productName>Lapis necklace</productName>
            <quantity>1</quantity>
            <USPrice>99.95</USPrice>
            <ipo:comment>Want this for the holidays!</ipo:
comment>
            <shipDate>1999-12-05</shipDate>
         </item>
     </items>
</ipo:purchaseOrder>
```

In Section 4.8 we describe how to prevent derived types from being used in this sort of substitution.

4.4 Deriving Complex Types by Restriction

In addition to deriving new complex types by extending content models, it is possible to derive new types by restricting the content models of existing types. Restriction of complex types is conceptually the same as restriction of simple types, except that the restriction of complex types involves a type's declarations rather than the acceptable range of a simple type's values. A complex type derived by restriction is very similar to its base type, except that its declarations are more limited than the corresponding declarations in the base type. In fact, the values represented by the new type are a subset of the values represented by the base type (as is the case with restriction of simple types). In other words, an application prepared for the values of the base type would not be surprised by the values of the restricted type.

For example, suppose we want to update our definition of the list of `items` in an international purchase order so that it must contain *at least* one `item` on order; the schema shown in `ipo.xsd` allows an `items` element to appear without any child `item` elements. To create our new `ConfirmedItems` type, we define the new type in the usual way, indicate that it is derived by restriction from the base type `Items`, and provide a new (more restrictive) value for the minimum number of `item` element occurrences. Notice that types derived by restriction must repeat all the components of the base type definition that are to be included in the derived type:

Deriving ConfirmedItems by Restriction from Items

```
<complexType name="ConfirmedItems">
 <complexContent>
  <restriction base="ipo:Items">
   <sequence>

    <!-- item element is different than in Items -->
    <element name="item" minOccurs="1" maxOccurs="unbounded">

     <!-- remainder of definition is same as Items -->
     <complexType>
      <sequence>
       <element name="productName" type="string"/>
       <element name="quantity">
        <simpleType>
         <restriction base="positiveInteger">
          <maxExclusive value="100"/>
```

```
          </restriction>
         </simpleType>
        </element>
        <element name="USPrice"    type="decimal"/>
        <element ref="ipo:comment" minOccurs="0"/>
        <element name="shipDate"   type="date" minOccurs="0"/>
       </sequence>
       <attribute name="partNum" type="ipo:SKU" use="required"/>
      </complexType>
     </element>

    </sequence>
   </restriction>
  </complexContent>
 </complexType>
```

This change, requiring at least one child element rather than allowing zero or more child elements, narrows the allowable number of child elements from a minimum of 0 to a minimum of 1. Note that all `ConfirmedItems` type elements will also be acceptable as `Item` type elements.

To further illustrate restriction, Table 3 shows several examples of how element and attribute declarations within type definitions may be restricted (the table shows element syntax although the first three examples are equally valid attribute restrictions).

Table 3. Restriction Examples

Base	Restriction	Notes
	default="1"	setting a default value where none was previously given
	fixed="100"	setting a fixed value where none was previously given
	type="string"	specifying a type where none was previously given
(minOccurs, maxOccurs)	**(minOccurs, maxOccurs)**	
(0, 1)	(0, 0)	exclusion of an optional component; this may also be accomplished by omitting the component's declaration from the restricted type definition
(0, unbounded)	(0, 0) (0, 37)	
(1, 9)	(1, 8) (2, 9) (4, 7) (3, 3)	
(1, unbounded)	(1, 12) (3, unbounded) (6, 6)	
(1, 1)	-	cannot restrict minOccurs or maxOccurs

4.5 Redefining Types & Groups

In Section 4.1 we described how to include definitions and declarations obtained from external schema files having the same target namespace. The `include` mechanism enables you to use externally created schema components "as-is", that is, without any modification. We have just described how to derive new types by extension and by restriction, and the `redefine` mechanism we describe here enables you to redefine simple and complex types, groups, and attribute groups that are obtained from external schema files. Like the `include` mechanism, `redefine` requires the external components to be in the same target namespace as the redefining schema, although external components from schemas that have no namespace can also be redefined. In the latter cases, the redefined components become part of the redefining schema's target namespace.

To illustrate the `redefine` mechanism, we use it instead of the `include` mechanism in the International Purchase Order schema, `ipo.xsd`, and we use it to modify the definition of the complex type `Address` contained in `address.xsd`:

Using redefine in the International Purchase Order

```
<schema targetNamespace="http://www.example.com/IPO"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:ipo="http://www.example.com/IPO">

 <!-- bring in address constructs -->
 <redefine
  schemaLocation="http://www.example.com/schemas/address.xsd">

  <!-- redefinition of Address -->
  <complexType name="Address">
   <complexContent>
     <extension base="ipo:Address">
      <sequence>
       <element name="country" type="string"/>
      </sequence>
     </extension>
   </complexContent>
  </complexType>

 </redefine>

 <!-- etc. -->

</schema>
```

The `redefine` element acts very much like the `include` element as it includes all the declarations and definitions from the `address.xsd` file. The complex type definition of `Address` uses the familiar extension syntax to add a `country` element to the definition of `Address`. However, note that the base type is also `Address`. Outside of the `redefine` element, any such attempt to define a complex type with the same name (and in the same namespace) as the base from which it is being derived would cause an error. But in this case, there is no error, and the extended definition of `Address` becomes the only definition of `Address`.

Now that Address has been redefined, the extension applies to all schema components that make use of Address. For example, address.xsd contains definitions of international address types that are derived from Address. These derivations reflect the redefined Address type, as shown in the following snippet:

Snippet of ipo.xml using Redefined Address

```
  ....
  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
   <name>Helen Zoe</name>
   <street>47 Eden Street</street>
   <city>Cambridge</city>
   <!-- country was added to Address which is base type of
UKAddress -->
   <country>United Kingdom</country>
   <!-- postcode was added as part of UKAddress -->
   <postcode>CB1 1JR</postcode>
  </shipTo>
  ....
```

Our example has been carefully constructed so that the redefined Address type does not conflict in any way with the types that are derived from the original Address definition. But note that it would be very easy to create a conflict. For example, if the international address type derivations had extended Address by adding a country element, then the redefinition of Address would be adding an element of the same name to the content model of Address. It is illegal to have two elements of the same name (and in the same target namespace) but different types in a content model, and so the attempt to redefine Address would cause an error. In general, redefine does not protect you from such errors, and it should be used cautiously.

4.6 Substitution Groups

XML Schema provides a mechanism, called substitution groups, that allows elements to be substituted for other elements. More specifically, elements can be assigned to a special group of elements that are said to be substitutable for a particular named element called the head element. (Note that the head element must be declared as a global element.) To illustrate, we declare two elements called customerComment and shipComment and assign them to a substitution group whose head element is comment, and so customerComment and shipComment can be used anyplace that we are able to use comment. Elements in a substitution group must have the same type as the head element, or they can have a type that has been derived from the head element's type. To declare these two new elements, and to make them substitutable for the comment element, we use the following syntax:

Declaring Elements Substitutable for comment

```
<element name="shipComment" type="string"
         substitutionGroup="ipo:comment"/>
<element name="customerComment" type="string"
         substitutionGroup="ipo:comment"/>
```

When these declarations are added to the international purchase order schema,

shipComment and customerComment can be substituted for comment in the instance document, for example:

Snippet of ipo.xml with Substituted Elements

```
....
 <items>
   <item partNum="833-AA">
     <productName>Lapis necklace</productName>
     <quantity>1</quantity>
     <USPrice>99.95</USPrice>
     <ipo:shipComment>
       Use gold wrap if possible
     </ipo:shipComment>
     <ipo:customerComment>
       Want this for the holidays!
     </ipo:customerComment>
     <shipDate>1999-12-05</shipDate>
   </item>
 </items>
....
```

Note that when an instance document contains element substitutions whose types are derived from those of their head elements, it is *not* necessary to identify the derived types using the xsi:type construction that we described in Section 4.3.

The existence of a substitution group does not require any of the elements in that class to be used, nor does it preclude use of the head element. It simply provides a mechanism for allowing elements to be used interchangeably.

4.7 Abstract Elements and Types

XML Schema provides a mechanism to force substitution for a particular element or type. When an element or type is declared to be "abstract", it cannot be used in an instance document. When an element is declared to be abstract, a member of that element's substitution group must appear in the instance document. When an element's corresponding type definition is declared as abstract, all instances of that element must use xsi:type to indicate a derived type that is not abstract.

In the substitution group example we described in Section 4.6, it would be useful to specifically disallow use of the comment element so that instances must make use of the customerComment and shipComment elements. To declare the comment element abstract, we modify its original declaration in the international purchase order schema, ipo.xsd, as follows:

```
<element name="comment" type="string" abstract="true"/>
```

With comment declared as abstract, instances of international purchase orders are now only valid if they contain customerComment and shipComment elements.

Declaring an element as abstract requires the use of a substitution group. Declaring a type as abstract simply requires the use of a type derived from it (and identified by the `xsi:type` attribute) in the instance document. Consider the following schema definition:

Schema for Vehicles

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://cars.example.com/schema"
        xmlns:target="http://cars.example.com/schema">

 <complexType name="Vehicle" abstract="true"/>

 <complexType name="Car">
  <complexContent>
   <extension base="target:Vehicle"/>
  </complexContent>
 </complexType>

 <complexType name="Plane">
  <complexContent>
   <extension base="target:Vehicle"/>
  </complexContent>
 </complexType>

 <element name="transport" type="target:Vehicle"/>
</schema>
```

The `transport` element is not abstract, therefore it can appear in instance documents. However, because its type definition is abstract, it may never appear in an instance document without an `xsi:type` attribute that refers to a derived type. That means the following is not schema-valid:

```
<transport xmlns="http://cars.example.com/schema"/>
```

because the `transport` element's type is abstract. However, the following is schema-valid:

```
<transport xmlns="http://cars.example.com/schema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:type="Car"/>
```

because it uses a non-abstract type that is substitutable for `Vehicle`.

4.8 Controlling the Creation & Use of Derived Types

So far, we have been able to derive new types and use them in instance documents without any restraints. In reality, schema authors will sometimes want to control derivations of particular types, and the use of derived types in instances.

XML Schema provides a couple of mechanisms that control the derivation of types. One of

these mechanisms allows the schema author to specify that for a particular complex type, new types may not be derived from it, either (a) by restriction, (b) by extension, or (c) at all. To illustrate, suppose we want to prevent any derivation of the `Address` type by restriction because we intend for it only to be used as the base for extended types such as `USAddress` and `UKAddress`. To prevent any such derivations, we slightly modify the original definition of `Address` as follows:

Preventing Derivations by Restriction of Address

```
<complexType name="Address" final="restriction">
 <sequence>
  <element name="name"   type="string"/>
  <element name="street" type="string"/>
  <element name="city"   type="string"/>
 </sequence>
</complexType>
```

The `restriction` value of the `final` attribute prevents derivations by restriction. Preventing derivations at all, or by extension, are indicated by the values `#all` and `extension` respectively. Moreover, there exists an optional `finalDefault` attribute on the `schema` element whose value can be one of the values allowed for the `final` attribute. The effect of specifying the `finalDefault` attribute is equivalent to specifying a `final` attribute on every type definition and element declaration in the schema.

Another type-derivation mechanism controls which facets can be applied in the derivation of a new simple type. When a simple type is defined, the `fixed` attribute may be applied to any of its facets to prevent a derivation of that type from modifying the value of the fixed facets. For example, we can define a `Postcode` simple type as:

Preventing Changes to Simple Type Facets

```
<simpleType name="Postcode">
  <restriction base="string">
    <length value="7" fixed="true"/>
  </restriction>
</simpleType>
```

Once this simple type has been defined, we can derive a new postal code type in which we apply a facet not fixed in the base definition, for example:

Legal Derivation from Postcode

```
<simpleType name="UKPostcode">
  <restriction base="ipo:Postcode">
    <pattern value="[A-Z]{2}\d\s\d[A-Z]{2}"/>
  </restriction>
</simpleType>
```

However, we cannot derive a new postal code in which we re-apply any facet that was fixed in the base definition:

Illegal Derivation from Postcode

```
<simpleType name="UKPostcode">
 <restriction base="ipo:Postcode">
  <pattern value="[A-Z]{2}\d\d[A-Z]{2}"/>
  <!-- illegal attempt to modify facet fixed in base type -->
  <length value="6" fixed="true"/>
 </restriction>
</simpleType>
```

In addition to the mechanisms that control type derivations, XML Schema provides a mechanism that controls which derivations and substitution groups may be used in instance documents. In Section 4.3, we described how the derived types, USAddress and UKAddress, could be used by the shipTo and billTo elements in instance documents. These derived types can replace the content model provided by the Address type because they are derived from the Address type. However, replacement by derived types can be controlled using the block attribute in a type definition. For example, if we want to block any derivation-by-restriction from being used in place of Address (perhaps for the same reason we defined Address with final="restriction"), we can modify the original definition of Address as follows:

Preventing Derivations by Restriction of Address in the Instance

```
<complexType name="Address" block="restriction">
 <sequence>
  <element name="name"   type="string"/>
  <element name="street" type="string"/>
  <element name="city"   type="string"/>
 </sequence>
</complexType>
```

The restriction value on the block attribute prevents derivations-by-restriction from replacing Address in an instance. However, it would not prevent UKAddress and USAddress from replacing Address because they were derived by extension. Preventing replacement by derivations at all, or by derivations-by-extension, are indicated by the values #all and extension respectively. As with final, there exists an optional blockDefault attribute on the schema element whose value can be one of the values allowed for the block attribute. The effect of specifying the blockDefault attribute is equivalent to specifying a block attribute on every type definition and element declaration in the schema.

5. Advanced Concepts III: The Quarterly Report

The home-products ordering and billing application can generate ad-hoc reports that summarize how many of which types of products have been billed on a per region basis. An example of such a report, one that covers the fourth quarter of 1999, is shown in 4Q99.xml.

Notice that in this section we use qualified elements in the schema, and default namespaces where possible in the instances.

Quarterly Report, 4Q99.xml

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  period="P3M" periodEnding="1999-12-31">

 <regions>
  <zip code="95819">
   <part number="872-AA" quantity="1"/>
   <part number="926-AA" quantity="1"/>
   <part number="833-AA" quantity="1"/>
   <part number="455-BX" quantity="1"/>
  </zip>
  <zip code="63143">
   <part number="455-BX" quantity="4"/>
  </zip>
 </regions>

 <parts>
  <part number="872-AA">Lawnmower</part>
  <part number="926-AA">Baby Monitor</part>
  <part number="833-AA">Lapis Necklace</part>
  <part number="455-BX">Sturdy Shelves</part>
 </parts>

</purchaseReport>
```

The report lists, by number and quantity, the parts billed to various zip codes, and it provides a description of each part mentioned. In summarizing the billing data, the intention of the report is clear and the data is unambiguous because a number of constraints are in effect. For example, each zip code appears only once (uniqueness constraint). Similarly, the description of every billed part appears only once although parts may be billed to several zip codes (referential constraint), see for example part number `455-BX`. In the following sections, we'll see how to specify these constraints using XML Schema.

The Report Schema, report.xsd

```
<schema targetNamespace="http://www.example.com/Report"
        xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:r="http://www.example.com/Report"
        xmlns:xipo="http://www.example.com/IPO"
        elementFormDefault="qualified">

 <!-- for SKU -->
 <import namespace="http://www.example.com/IPO"/>

 <annotation>
  <documentation xml:lang="en">
   Report schema for Example.com
   Copyright 2000 Example.com. All rights reserved.
  </documentation>
 </annotation>

 <element name="purchaseReport">
  <complexType>
```

```
    <sequence>
     <element name="regions" type="r:RegionsType">
      <keyref name="dummy2" refer="r:pNumKey">
       <selector xpath="r:zip/r:part"/>
       <field xpath="@number"/>
      </keyref>
     </element>

     <element name="parts" type="r:PartsType"/>
    </sequence>
    <attribute name="period"       type="duration"/>
    <attribute name="periodEnding" type="date"/>
   </complexType>

   <unique name="dummy1">
    <selector xpath="r:regions/r:zip"/>
    <field xpath="@code"/>
   </unique>

   <key name="pNumKey">
    <selector xpath="r:parts/r:part"/>
    <field xpath="@number"/>
   </key>
  </element>

  <complexType name="RegionsType">
   <sequence>
    <element name="zip" maxOccurs="unbounded">
     <complexType>
      <sequence>
       <element name="part" maxOccurs="unbounded">
        <complexType>
         <complexContent>
          <restriction base="anyType">
           <attribute name="number"   type="xipo:SKU"/>
           <attribute name="quantity" type="positiveInteger"/>
          </restriction>
         </complexContent>
        </complexType>
       </element>
      </sequence>
      <attribute name="code" type="positiveInteger"/>
     </complexType>
    </element>
   </sequence>
  </complexType>

  <complexType name="PartsType">
   <sequence>
    <element name="part" maxOccurs="unbounded">
     <complexType>
      <simpleContent>
       <extension base="string">
        <attribute name="number" type="xipo:SKU"/>
```

```
        </extension>
       </simpleContent>
      </complexType>
     </element>
    </sequence>
   </complexType>

  </schema>
```

5.1 Specifying Uniqueness

XML Schema enables us to indicate that any attribute or element value must be unique within a certain scope. To indicate that one particular attribute or element value is unique, we use the `unique` element first to "select" a set of elements, and then to identify the attribute or element "field" relative to each selected element that has to be unique within the scope of the set of selected elements. In the case of our report schema, `report.xsd`, the `selector` element's `xpath` attribute contains an XPath expression, `regions/zip`, that selects a list of all the `zip` elements in a report instance. Likewise, the `field` element's `xpath` attribute contains a second XPath expression, `@code`, that specifies that the `code` attribute values of those elements must be unique. Note that the XPath expressions limit the scope of what must be unique. The report might contain another `code` attribute, but its value does not have to be unique because it lies outside the scope defined by the XPath expressions. Also note that the XPath expressions you can use in the `xpath` attribute are limited to a subset of the full XPath language defined in XML Path Language 1.0.

We can also indicate combinations of fields that must be unique. To illustrate, suppose we can relax the constraint that zip codes may only be listed once, although we still want to enforce the constraint that any product is listed only once within a given zip code. We could achieve such a constraint by specifying that the combination of zip code and product number must be unique. From the report document, `4Q99.xml`, the combined values of zip `code` and `number` would be: {`95819 872-AA`}, {`95819 926-AA`}, {`95819 833-AA`}, {`95819 455-BX`}, and {`63143 455-BX`}. Clearly, these combinations do not distinguish between zip `code` and `number` combinations derived from single or multiple listings of any particular zip, but the combinations would unambiguously represent a product listed more than once within a single zip. In other words, a schema processor could detect violations of the uniqueness constraint.

To define combinations of values, we simply add `field` elements to identify all the values involved. So, to add the part number value to our existing definition, we add a new `field` element whose `xpath` attribute value, `part/@number`, identifies the `number` attribute of `part` elements that are children of the `zip` elements identified by `regions/zip`:

A Unique Composed Value

```
  <unique name="dummy1">
   <selector xpath="r:regions/r:zip"/>
   <field    xpath="@code"/>
   <field    xpath="r:part/@number"/>
  </unique>
```

5.2 Defining Keys & their References

In the 1999 quarterly report, the description of every billed part appears only once. We could enforce this constraint using `unique`, however, we also want to ensure that every part-quantity element listed under a zip code has a corresponding part description. We enforce the constraint using the `key` and `keyref` elements. The report schema, `report.xsd`, shows that the `key` and `keyref` constructions are applied using almost the same syntax as `unique`. The key element applies to the `number` attribute value of `part` elements that are children of the `parts` element. This declaration of `number` as a key means that its value must be unique and cannot be set to nil (i.e. is not nillable), and the name that is associated with the key, `pNumKey`, makes the key referenceable from elsewhere.

To ensure that the part-quantity elements have corresponding part descriptions, we say that the `number` attribute (`<field>@number</field>`) of those elements (`<selector>zip/part</selector>`) must reference the `pNumKey` key. This declaration of `number` as a `keyref` does not mean that its value must be unique, but it does mean there must exist a `pNumKey` with the same value.

As you may have figured out by analogy with `unique`, it is possible to define combinations of `key` and `keyref` values. Using this mechanism, we could go beyond simply requiring the product numbers to be equal, and define a combination of values that must be equal. Such values may involve combinations of multiple value types (`string`, `integer`, `date`, etc.), provided that the order and type of the `field` element references is the same in both the `key` and `keyref` definitions.

5.3 XML Schema Constraints vs. XML 1.0 ID Attributes

XML 1.0 provides a mechanism for ensuring uniqueness using the ID attribute and its associated attributes IDREF and IDREFS. This mechanism is also provided in XML Schema through the `ID`, `IDREF`, and `IDREFS` simple types which can be used for declaring XML 1.0-style attributes. XML Schema also introduces new mechanisms that are more flexible and powerful. For example, XML Schema's mechanisms can be applied to any element and attribute content, regardless of its type. In contrast, ID is a type of *attribute* and so it cannot be applied to attributes, elements or their content. Furthermore, Schema enables you to specify the scope within which uniqueness applies whereas the scope of an ID is fixed to be the whole document. Finally, Schema enables you to create `key`s or a `keyref` from combinations of element and attribute content whereas ID has no such facility.

5.4 Importing Types

The report schema, `report.xsd`, makes use of the simple type `xipo:SKU` that is defined in another schema, and in another target namespace. Recall that we used `include` so that the schema in `ipo.xsd` could make use of definitions and declarations from `address.xsd`. We cannot use `include` here because it can only pull in definitions and declarations from a schema whose target namespace is the same as the including schema's target namespace. Hence, the `include` element does not identify a namespace (although it does require a `schemaLocation`). The import mechanism that we describe in this section is an important mechanism that enables schema components from different target namespaces to be used together, and hence enables the schema validation of instance content defined across multiple namespaces.

To import the type SKU and use it in the report schema, we identify the namespace in which

SKU is defined, and associate that namespace with a prefix for use in the report schema. Concretely, we use the `import` element to identify SKU's target namespace, `http://www.example.com/IPO`, and we associate the namespace with the prefix `xipo` using a standard namespace declaration. The simple type SKU, defined in the namespace `http://www.example.com/IPO`, may then be referenced as `xipo:SKU` in any of the report schema's definitions and declarations.

In our example, we imported one simple type from one external namespace, and used it for declaring attributes. XML Schema in fact permits multiple schema components to be imported, from multiple namespaces, and they can be referred to in both definitions and declarations. For example in `report.xsd` we could additionally reuse the `comment` element declared in `ipo.xsd` by referencing that element in a declaration:

```
<element ref="xipo:comment"/>
```

Note however, that we cannot reuse the `shipTo` element from `po.xsd`, and the following is not legal because only *global* schema components can be imported:

```
<element ref="xipo:shipTo"/>
```

In `ipo.xsd`, `comment` is declared as a global element, in other words it is declared as an element of the `schema`. In contrast, `shipTo` is declared locally, in other words it is an element declared inside a complex type definition, specifically the `PurchaseOrderType` type.

Complex types can also be imported, and they can be used as the base types for deriving new types. Only named complex types can be imported; local, anonymously defined types cannot. Suppose we want to include in our reports the name of an analyst, along with contact information. We can reuse the (globally defined) complex type `USAddress` from `address.xsd`, and extend it to define a new type called `Analyst` by adding the new elements `phone` and `email`:

Defining Analyst by Extending USAddress

```
<complexType name="Analyst">
 <complexContent>
  <extension base="xipo:USAddress">
   <sequence>
    <element name="phone" type="string"/>
    <element name="email" type="string"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

Using this new type we declare an element called `analyst` as part of the `purchaseReport` element declaration (declarations not shown) in the report schema. Then, the following instance document would conform to the modified report schema:

Instance Document Conforming to Report Schema with Analyst Type

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  period="P3M" periodEnding="1999-12-31">
  <!-- regions and parts elements omitted -->
   <analyst>
        <name>Wendy Uhro</name>
        <street>10 Corporate Towers</street>
        <city>San Jose</city>
        <state>CA</state>
        <zip>95113</zip>
        <phone>408-271-3366</phone>
        <email>uhro@example.com</email>
   </analyst>
</purchaseReport>
```

When schema components are imported from multiple namespaces, each namespace must be identified with a separate `import` element. The `import` elements themselves must appear as the first children of the `schema` element. Furthermore, each namespace must be associated with a prefix, using a standard namespace declaration, and that prefix is used to qualify references to any schema components belonging to that namespace. Finally, `import` elements optionally contain a `schemaLocation` attribute to help locate resources associated with the namespaces. We discuss the `schemaLocation` attribute in more detail in a later section.

5.4.1 Type Libraries

As XML schemas become more widespread, schema authors will want to create simple and complex types that can be shared and used as building blocks for creating new schemas. XML Schemas already provides types that play this role, in particular, the types described in the Simple Types appendix and in an introductory type library.

Schema authors will undoubtedly want to create their own libraries of types to represent currency, units of measurement, business addresses, and so on. Each library might consist of a schema containing one or more definitions, for example, a schema containing a currency type:

Example Currency Type in Type Library

```
<schema targetNamespace="http://www.example.com/Currency"
        xmlns:c="http://www.example.com/Currency"
        xmlns="http://www.w3.org/2001/XMLSchema">

 <annotation>
  <documentation xml:lang="en">
   Definition of Currency type based on ISO 4217
  </documentation>
 </annotation>

 <complexType name="Currency">
  <simpleContent>
```

```
    <extension base="decimal">
     <attribute name="name">
      <simpleType>
       <restriction base="string">

         <enumeration value="AED">
          <annotation>
           <documentation xml:lang="en">
            United Arab Emirates: Dirham (1 Dirham = 100 Fils)
           </documentation>
          </annotation>
         </enumeration>

         <enumeration value="AFA">
          <annotation>
           <documentation xml:lang="en">
            Afghanistan: Afghani (1 Afghani = 100 Puls)
           </documentation>
          </annotation>
         </enumeration>

         <enumeration value="ALL">
          <annotation>
           <documentation xml:lang="en">
            Albania, Lek (1 Lek = 100 Qindarka)
           </documentation>
          </annotation>
         </enumeration>

         <!-- and other currencies -->

       </restriction>
      </simpleType>
     </attribute>
    </extension>
   </simpleContent>
  </complexType>

</schema>
```

An example of an element appearing in an instance and having this type:

```
<convertFrom name="AFA">199.37</convertFrom>
```

Once we have defined the currency type, we can make it available for re-use in other schemas through the `import` mechanism just described.

5.5 Any Element, Any Attribute

In previous sections we have seen several mechanisms for extending the content models of complex types. For example, a mixed content model can contain arbitrary character data in

addition to elements, and for example, a content model can contain elements whose types are imported from external namespaces. However, these mechanisms provide very broad and very narrow controls respectively. The purpose of this section is to describe a flexible mechanism that enables content models to be extended by any elements and attributes belonging to specified namespaces.

To illustrate, consider a version of the quarterly report, 4Q99html.xml, in which we have embedded an HTML representation of the XML parts data. The HTML content appears as the content of the element `htmlExample`, and the default namespace is changed on the outermost HTML element (`table`) so that all the HTML elements belong to the HTML namespace, `http://www.w3.org/1999/xhtml`:

Quarterly Report with HTML, 4Q99html.xml

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  period="P3M" periodEnding="1999-12-31">

 <regions>
   <!-- part sales listed by zip code, data from 4Q99.xml -->
 </regions>

 <parts>
   <!-- part descriptions from 4Q99.xml -->
 </parts>

 <htmlExample>
  <table xmlns="http://www.w3.org/1999/xhtml"
         border="0" width="100%">
   <tr>
     <th align="left">Zip Code</th>
     <th align="left">Part Number</th>
     <th align="left">Quantity</th>
   </tr>
   <tr><td>95819</td><td> </td><td> </td></tr>
   <tr><td> </td><td>872-AA</td><td>1</td></tr>
   <tr><td> </td><td>926-AA</td><td>1</td></tr>
   <tr><td> </td><td>833-AA</td><td>1</td></tr>
   <tr><td> </td><td>455-BX</td><td>1</td></tr>
   <tr><td>63143</td><td> </td><td> </td></tr>
   <tr><td> </td><td>455-BX</td><td>4</td></tr>
  </table>
 </htmlExample>

</purchaseReport>
```

To permit the appearance of HTML in the instance document we modify the report schema by declaring a new element `htmlExample` whose content is defined by the `any` element. In general, an `any` element specifies that any well-formed XML is permissible in a type's content model. In the example, we require the XML to belong to the namespace `http://www.w3.org/1999/xhtml`, in other words, it should be HTML. The example also requires there to be at least one element present from this namespace, as indicated by the values of `minOccurs`

and maxOccurs:

Modification to purchaseReport Declaration to Allow HTML in Instance

```
<element name="purchaseReport">
 <complexType>
  <sequence>
   <element name="regions" type="r:RegionsType"/>
   <element name="parts"   type="r:PartsType"/>
   <element name="htmlExample">
    <complexType>
     <sequence>
      <any namespace="http://www.w3.org/1999/xhtml"
           minOccurs="1" maxOccurs="unbounded"
           processContents="skip"/>
     </sequence>
    </complexType>
   </element>
  </sequence>
  <attribute name="period"       type="duration"/>
  <attribute name="periodEnding" type="date"/>
 </complexType>
</element>
```

The modification permits some well-formed XML belonging to the namespace `http://www.w3.org/1999/xhtml` to appear inside the `htmlExample` element. Therefore 4Q99html.xml is permissible because there is one element which (with its children) is well-formed, the element appears inside the appropriate element (`htmlExample`), and the instance document asserts that the element and its content belongs to the required namespace. However, the HTML may not actually be valid because nothing in 4Q99html.xml by itself can provide that guarantee. If such a guarantee is required, the value of the `processContents` attribute should be set to `strict` (the default value). In this case, an XML processor is obliged to obtain the schema associated with the required namespace, and validate the HTML appearing within the `htmlExample` element.

In another example, we define a `text` type which is similar to the text type defined in XML Schema's introductory type library (see also Section 5.4.1), and is suitable for internationalized human-readable text. The text type allows an unrestricted mixture of character content and element content from any namespace, for example Ruby annotations, along with an optional `xml:lang` attribute. The `lax` value of the `processContents` attribute instructs an XML processor to validate the element content on a can-do basis: It will validate elements and attributes for which it can obtain schema information, but it will not signal errors for those it cannot obtain any schema information.

Text Type

```
<xsd:complexType name="text">
 <xsd:complexContent mixed="true">
  <xsd:restriction base="xsd:anyType">
   <xsd:sequence>
    <xsd:any processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
```

```
    </xsd:sequence>
    <xsd:attribute ref="xml:lang"/>
   </xsd:restriction>
  </xsd:complexContent>
 </xsd:complexType>
```

Namespaces may be used to permit and forbid element content in various ways depending upon the value of the `namespace` attribute, as shown in Table 4:

Table 4. Namespace Attribute in Any

Value of Namespace Attribute	Allowable Element Content
##any	Any well-formed XML from any namespace (default)
##local	Any well-formed XML that is not qualified, i. e. not declared to be in a namespace
##other	Any well-formed XML that is not from the target namespace of the type being defined
"http://www.w3.org/1999/xhtml ##targetNamespace"	Any well-formed XML belonging to any namespace in the (whitespace separated) list; ##targetNamespace is shorthand for the target namespace of the type being defined

In addition to the `any` element which enables element content according to namespaces, there is a corresponding `anyAttribute` element which enables attributes to appear in elements. For example, we can permit any HTML attribute to appear as part of the `htmlExample` element by adding `anyAttribute` to its declaration:

Modification to htmlExample Declaration to Allow HTML Attributes

```
<element name="htmlExample">
 <complexType>
  <sequence>
   <any namespace="http://www.w3.org/1999/xhtml"
        minOccurs="1" maxOccurs="unbounded"
        processContents="skip"/>
  </sequence>
  <anyAttribute namespace="http://www.w3.org/1999/xhtml"/>
 </complexType>
</element>
```

This declaration permits an HTML attribute, say `href`, to appear in the `htmlExample` element. For example:

An HTML attribute in the htmlExample Element

```
....
  <htmlExample xmlns:h="http://www.w3.org/1999/xhtml"
               h:href="http://www.example.com/reports/4Q99.html">
     <!-- HTML markup here -->
```

```
    </htmlExample>
  ....
```

The namespace attribute in an anyAttribute element can be set to any of the values listed in Table 4 for the any element, and anyAttribute can be specified with a processContents attribute. In contrast to an any element, anyAttribute cannot constrain the number of attributes that may appear in an element.

5.6 schemaLocation

XML Schema uses the schemaLocation and xsi:schemaLocation attributes in three circumstances.

1. In an instance document, the attribute xsi:schemaLocation provides hints from the author to a processor regarding the location of schema documents. The author warrants that these schema documents are relevant to checking the validity of the document content, on a namespace by namespace basis. For example, we can indicate the location of the Report schema to a processor of the Quarterly Report:

Using schemaLocation in the Quarterly Report, 4Q99html.xml

```
<purchaseReport
  xmlns="http://www.example.com/Report"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.com/Report
  http://www.example.com/Report.xsd"
  period="P3M" periodEnding="1999-12-31">

 <!-- etc. -->

</purchaseReport>
```

The schemaLocation attribute contains pairs of values: The first member of each pair is the namespace for which the second member is the hint describing where to find to an appropriate schema document. The presence of these hints does not require the processor to obtain or use the cited schema documents, and the processor is free to use other schemas obtained by any suitable means, or to use no schema at all.

A schema is not required to have a namespace (see Section 3.4) and so there is a noNamespaceSchemaLocation attribute which is used to provide hints for the locations of schema documents that do not have target namespaces.

2. In a schema, the include element has a required schemaLocation attribute, and it contains a URI reference which must identify a schema document. The effect is to compose a final effective schema by merging the declarations and definitions of the including and the included schemas. For example, in Section 4, the type definitions of Address, USAddress, UKAddress, USState (along with their attribute and local element declarations) from address.xsd were added to the element declarations of purchaseOrder and comment, and the type definitions of PurchaseOrderType, Items and SKU (along with their attribute and local element declarations) from ipo.xsd to create a single schema.

3. Also in a schema, the `import` element has optional `namespace` and `schemaLocation` attributes. If present, the `schemaLocation` attribute is understood in a way which parallels the interpretation of `xsi:schemaLocation` in (1). Specifically, it provides a hint from the author to a processor regarding the location of a schema document that the author warrants supplies the required components for the namespace identified by the `namespace` attribute. To import components that are not in any target namespace, the `import` element is used without a `namespace` attribute (and with or without a `schemaLocation` attribute). References to components imported in this manner are unqualified.

Note that the `schemaLocation` is only a hint and some processors and applications will have reasons to not use it. For example, an HTML editor may have a built-in HTML schema.

5.7 Conformance

An instance document may be processed against a schema to verify whether the rules specified in the schema are honored in the instance. Typically, such processing actually does two things, (1) it checks for conformance to the rules, a process called schema validation, and (2) it adds supplementary information that is not immediately present in the instance, such as types and default values, called infoset contributions.

The author of an instance document, such as a particular purchase order, may claim, in the instance itself, that it conforms to the rules in a particular schema. The author does this using the `schemaLocation` attribute discussed above. But regardless of whether a `schemaLocation` attribute is present, an application is free to process the document against any schema. For example, a purchasing application may have the policy of always using a certain purchase order schema, regardless of any `schemaLocation` values.

Conformance checking can be thought of as proceeding in steps, first checking that the root element of the document instance has the right contents, then checking that each subelement conforms to its description in a schema, and so on until the entire document is verified. Processors are required to report what checking has been carried out.

To check an element for conformance, the processor first locates the declaration for the element in a schema, and then checks that the `targetNamespace` attribute in the schema matches the actual namespace URI of the element. Alternatively, it may determine that the schema does not have a `targetNamespace` attribute and the instance element is not namespace-qualified.

Supposing the namespaces match, the processor then examines the type of the element, either as given by the declaration in the schema, or by an `xsi:type` attribute in the instance. If the latter, the instance type must be an allowed substitution for the type given in the schema; what is allowed is controlled by the `block` attribute in the element declaration. At this same time, default values and other infoset contributions are applied.

Next the processor checks the immediate attributes and contents of the element, comparing these against the attributes and contents permitted by the element's type. For example, considering a `shipTo` element such as the one in Section 2.1, the processor checks what is permitted for an `Address`, because that is the `shipTo` element's type.

If the element has a simple type, the processor verifies that the element has no attributes or

contained elements, and that its character content matches the rules for the simple type. This sometimes involves checking the character sequence against regular expressions or enumerations, and sometimes it involves checking that the character sequence represents a value in a permitted range.

If the element has a complex type, then the processor checks that any required attributes are present and that their values conform to the requirements of their simple types. It also checks that all required subelements are present, and that the sequence of subelements (and any mixed text) matches the content model declared for the complex type. Regarding subelements, schemas can either require exact name matching, permit substitution by an equivalent element or permit substitution by any element allowed by an 'any' particle.

Unless a schema indicates otherwise (as it can for 'any' particles) conformance checking then proceeds one level more deeply by looking at each subelement in turn, repeating the process described above.

A Acknowledgements

Many people have contributed ideas, material and feedback that has improved this document. In particular, the editor acknowledges contributions from David Beech, Paul Biron, Don Box, Allen Brown, David Cleary, Dan Connolly, Roger Costello, Martin Dürst, Martin Gudgin, Dave Hollander, Joe Kesselman, John McCarthy, Andrew Layman, Eve Maler, Ashok Malhotra, Noah Mendelsohn, Michael Sperberg-McQueen, Henry Thompson, Misha Wolf, and Priscilla Walmsley for validating the examples.

B Simple Types & their Facets

The legal values for each simple type can be constrained through the application of one or more facets. Tables B1.a and B1.b list all of XML Schema's built-in simple types and the facets applicable to each type. The names of the simple types and the facets are linked from the tables to the corresponding descriptions in XML Schema Part 2: Datatypes

Table B1.a. Simple Types & Applicable Facets

Simple Types	Facets					
	length	minLength	maxLength	pattern	enumeration	whiteSpace
string	y	y	y	y	y	y
normalizedString	y	y	y	y	y	y
token	y	y	y	y	y	y
byte				y	y	y
unsignedByte				y	y	y
base64Binary	y	y	y	y	y	y
hexBinary	y	y	y	y	y	y
integer				y	y	y
positiveInteger				y	y	y

negativeInteger				y	y	y
nonNegativeInteger				y	y	y
nonPositiveInteger				y	y	y
int				y	y	y
unsignedInt				y	y	y
long				y	y	y
unsignedLong				y	y	y
short				y	y	y
unsignedShort				y	y	y
decimal				y	y	y
float				y	y	y
double				y	y	y
boolean				y		y
time				y	y	y
dateTime				y	y	y
duration				y	y	y
date				y	y	y
gMonth				y	y	y
gYear				y	y	y
gYearMonth				y	y	y
gDay				y	y	y
gMonthDay				y	y	y
Name	y	y	y	y	y	y
QName	y	y	y	y	y	y
NCName	y	y	y	y	y	y
anyURI	y	y	y	y	y	y
language	y	y	y	y	y	y
ID	y	y	y	y	y	y
IDREF	y	y	y	y	y	y
IDREFS	y	y	y		y	y
ENTITY	y	y	y	y	y	y
ENTITIES	y	y	y		y	y
NOTATION	y	y	y	y	y	y
NMTOKEN	y	y	y	y	y	y
NMTOKENS	y	y	y		y	y

The facets listed in Table B1.b apply only to simple types which are ordered. Not all simple

types are ordered and so B1.b does not list all of the simple types.

Table B1.b. Simple Types & Applicable Facets						
Simple Types	**Facets**					
	max Inclusive	**max Exclusive**	**min Inclusive**	**min Exclusive**	**totalDigits**	**fractionDigits**
byte	y	y	y	y	y	y
unsignedByte	y	y	y	y	y	y
integer	y	y	y	y	y	y
positiveInteger	y	y	y	y	y	y
negativeInteger	y	y	y	y	y	y
nonNegativeInteger	y	y	y	y	y	y
nonPositiveInteger	y	y	y	y	y	y
int	y	y	y	y	y	y
unsignedInt	y	y	y	y	y	y
long	y	y	y	y	y	y
unsignedLong	y	y	y	y	y	y
short	y	y	y	y	y	y
unsignedShort	y	y	y	y	y	y
decimal	y	y	y	y	y	y
float	y	y	y	y		
double	y	y	y	y		
time	y	y	y	y		
dateTime	y	y	y	y		
duration	y	y	y	y		
date	y	y	y	y		
gMonth	y	y	y	y		
gYear	y	y	y	y		
gYearMonth	y	y	y	y		
gDay	y	y	y	y		
gMonthDay	y	y	y	y		

C Using Entities

XML 1.0 provides various types of entities which are named fragments of content that can be used in the construction of both DTD's (parameter entities) and instance documents. In Section 2.7, we noted how named groups mimic parameter entities. In this section we show how entities can be declared in instance documents, and how the functional equivalents of entities can be declared in schemas.

Suppose we want to declare and use an entity in an instance document, and that document is also constrained by a schema. For example:

Declaring and referencing an entity in an instance document.

```
<?xml version="1.0" ?>
<!DOCTYPE PurchaseOrder [
<!ENTITY eacute "é">
]>
<purchaseOrder xmlns="http://www.example.com/PO1"
               orderDate="1999-10-20>
 <!-- etc. -->
  <city>Montr&eacute;al</city>
 <!-- etc. -->
</purchaseOrder>
```

Here, we declare an entity called `eacute` as part of an internal (DTD) subset, and we reference this entity in the content of the `city` element. Note that when this instance document is processed, the entity will be dereferenced before schema validation takes place. In other words, a schema processor will determine the validity of the `city` element using `Montréal` as the element's value.

We can achieve a similar but not identical outcome by declaring an element in a schema, and by setting the element's content appropriately:

```
<xsd:element name="eacute" type="xsd:token" fixed="é"/>
```

And this element can be used in an instance document:

Using an element instead of an entity in an instance document.

```
<?xml version="1.0" ?>
<purchaseOrder xmlns="http://www.example.com/PO1"
               xmlns:c="http://www.example.com/characterElements"
               orderDate="1999-10-20>
 <!-- etc. -->
  <city>Montr<c:eacute/>al</city>
 <!-- etc. -->
</purchaseOrder>
```

In this case, a schema processor will process two elements, a `city` element, and an `eacute` element for the contents of which the processor will supply the single character é. Note that the extra element will complicate string matching; the two forms of the name "Montréal" given in the two examples above will not match each other using normal string-comparison techniques.

D Regular Expressions

XML Schema's `pattern` facet uses a regular expression language that supports Unicode. It

is fully described in XML Schema Part 2. The language is similar to the regular expression language used in the Perl Programming language, although expressions are matched against entire lexical representations rather than user-scoped lexical representations such as line and paragraph. For this reason, the expression language does not contain the metacharacters ^ and $, although ^ is used to express exception, e.g. [^0-9]x.

Table D1. Examples of Regular Expressions

Expression	Match(s)
Chapter \d	Chapter 0, Chapter 1, Chapter 2 ....
Chapter\s\d	Chapter followed by a single whitespace character (space, tab, newline, etc.), followed by a single digit
Chapter\s\w	Chapter followed by a single whitespace character (space, tab, newline, etc.), followed by a word character (XML 1.0 Letter or Digit)
Espanñola	Española
\p{Lu}	any uppercase character, the value of \p{} (e.g. "Lu") is defined by Unicode
\p{IsGreek}	any Greek character, the 'Is' construction may be applied to any block name (e.g. "Greek") as defined by Unicode
\P{IsGreek}	any non-Greek character, the 'Is' construction may be applied to any block name (e.g. "Greek") as defined by Unicode
a*x	x, ax, aax, aaax ....
a?x	ax, x
a+x	ax, aax, aaax ....
(a|b)+x	ax, bx, aax, abx, bax, bbx, aaax, aabx, abax, abbx, baax, babx, bbax, bbbx, aaaax ....
[abcde]x	ax, bx, cx, dx, ex
[a-e]x	ax, bx, cx, dx, ex
[-ae]x	-x, ax, ex
[ae-]x	ax, ex, -x
[^0-9]x	any non-digit character followed by the character x
\Dx	any non-digit character followed by the character x
.x	any character followed by the character x
.*abc.*	1x2abc, abc1x2, z3456abchooray ....
ab{2}x	abbx
ab{2,4}x	abbx, abbbx, abbbbx
ab{2,}x	abbx, abbbx, abbbbx ....
(ab){2}x	ababx

E Index

XML Schema Elements. Each element name is linked to a formal XML description in either the Structures or Datatypes parts of the XML Schema specification. Element names are followed by one or more links to examples (identified by section number) in the Primer.

- `all`: 2.7
- `annotation`: 2.6
- `any`: 5.5
- `anyAttribute`: 5.5
- `appInfo`: 2.6
- `attribute`: 2.2
- `attributeGroup`: 2.8
- `choice`: 2.7
- `complexContent`: 2.5.3
- `complexType`: 2.2
- `documentation`: 2.6
- `element`: 2.2
- `enumeration`: 2.3
- `extension`: 2.5.1, 4.2
- `field`: 5.1
- `group`: 2.7
- `import`: 5.4
- `include`: 4.1
- `key`: 5.2
- `keyref`: 5.2
- `length`: 2.3.1
- `list`: 2.3.1
- `maxInclusive`: 2.3
- `maxLength`: 2.3.1
- `minInclusive`: 2.3
- `minLength`: 2.3.1
- `pattern`: 2.3
- `redefine`: 4.5
- `restriction`: 2.3, 4.4
- `schema`: 2.1
- `selector`: 5.1
- `sequence`: 2.7
- `simpleContent`: 2.5.1
- `simpleType`: 2.3
- `union`: 2.3.2
- `unique`: 5.1

XML Schema Attributes. Each attribute name is followed by one or more pairs of references. Each pair of references consists of a link to an example in the Primer, plus a link to a formal XML description in either the Structures or Datatypes parts of the XML Schema specification.

- `abstract`: element declaration [Structures], complex type definition [Structures]
- `attributeFormDefault`: `schema` element [Structures]
- `base`: simple type definition [Datatypes], complex type definition [Structures]
- `block`: complex type definition [Structures],
- `blockDefault`: `schema` element [Structures]
- `default`: attribute declaration [Structures],
- `default`: element declaration [Structures],

- elementFormDefault: schema element [Structures]
- final: complex type definition [Structures]
- finalDefault: schema element [Structures]
- fixed: attribute declaration [Structures],
- fixed: element declaration [Structures]
- fixed: simple type definition [Datatypes]
- form: element declaration [Structures], attribute declaration [Structures]
- itemType: list type definition [Datatypes]
- memberTypes: union type definition [Datatypes]
- maxOccurs: element declaration [Structures]
- minOccurs: element declaration [Structures]
- mixed: complex type definition [Structures]
- name: element declaration [Structures], attribute declaration [Structures], complex type definition [Structures], simple type definition [Datatypes]
- namespace: any element [Structures], include element [Structures]
- noNamespaceSchemaLocation: instance element [Structures]
- xsi:nil: instance element [Structures]
- nillable: element declaration [Structures]
- processContents: any element [Structures], anyAttribute element [Structures]
- ref: element declaration [Structures]
- schemaLocation: include specification [Structures], redefine specification [Structures], import specification [Structures]
- xsi:schemaLocation: instance attribute [Structures]
- substitutionGroup: element declaration [Structures]
- targetNamespace: schema element [Structures]
- type: element declaration [Structures], attribute declaration [Structures]
- xsi:type: instance element [Structures]
- use: attribute declaration [Structures]
- xpath: selector & field elements [Structures]

XML Schema's simple types are described in Table 2.

XML Path Language (XPath) Version 1.0

W3C Recommendation 16 November 1999

This version:
http://www.w3.org/TR/1999/REC-xpath-19991116
(available in XML or HTML)
Latest version:
http://www.w3.org/TR/xpath
Previous versions:
http://www.w3.org/TR/1999/PR-xpath-19991008
http://www.w3.org/1999/08/WD-xpath-19990813
http://www.w3.org/1999/07/WD-xpath-19990709
http://www.w3.org/TR/1999/WD-xslt-19990421
Editors:
James Clark <jjc@jclark.com>
Steve DeRose (Inso Corp. and Brown University)
<Steven_DeRose@Brown.edu>

Abstract

XPath is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer.

Status of this document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It

is a stable document and may be used as reference material or cited as a normative reference from other documents. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

The list of known errors in this specification is available at http://www.w3.org/1999/11/REC-xpath-19991116-errata.

Comments on this specification may be sent to www-xpath-comments@w3.org; archives of the comments are available.

The English version of this specification is the only normative version. However, for translations of this document, see http://www.w3.org/Style/XSL/translations.html.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

This specification is joint work of the XSL Working Group and the XML Linking Working Group and so is part of the W3C Style activity and of the W3C XML activity.

Table of contents

Appendices

---

1 Introduction

XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations [XSLT] and XPointer [XPointer]. The primary purpose of XPath is to address parts of an XML [XML] document. In support of this primary purpose, it also provides basic facilities for manipulation of strings, numbers and booleans. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax. XPath gets its name from its use of a path notation as in URLs for navigating through the hierarchical structure of an XML document.

In addition to its use for addressing, XPath is also designed so that it has a natural subset that can be used for matching (testing whether or not a node matches a pattern); this use of XPath is described in XSLT.

XPath models an XML document as a tree of nodes. There are different types of nodes, including element nodes, attribute nodes and text nodes. XPath defines a way to compute a string-value for each type of node. Some types of

nodes also have names. XPath fully supports XML Namespaces [XML Names]. Thus, the name of a node is modeled as a pair consisting of a local part and a possibly null namespace URI; this is called an expanded-name. The data model is described in detail in **[5 Data Model]**.

The primary syntactic construct in XPath is the expression. An expression matches the production Expr. An expression is evaluated to yield an object, which has one of the following four basic types:

- node-set (an unordered collection of nodes without duplicates)
- boolean (true or false)
- number (a floating-point number)
- string (a sequence of UCS characters)

Expression evaluation occurs with respect to a context. XSLT and XPointer specify how the context is determined for XPath expressions used in XSLT and XPointer respectively. The context consists of:

- a node (the **context node**)
- a pair of non-zero positive integers (the **context position** and the **context size**)
- a set of variable bindings
- a function library
- the set of namespace declarations in scope for the expression

The context position is always less than or equal to the context size.

The variable bindings consist of a mapping from variable names to variable values. The value of a variable is an object, which can be of any of the types that are possible for the value of an expression, and may also be of additional types not specified here.

The function library consists of a mapping from function names to functions. Each function takes zero or more arguments and returns a single result. This document defines a core function library that all XPath implementations must support (see **[4 Core Function Library]**). For a function in the core function library, arguments and result are of the four basic types. Both XSLT and XPointer extend XPath by defining additional functions; some of these functions operate on the four basic types; others operate on additional data types defined by XSLT and XPointer.

The namespace declarations consist of a mapping from prefixes to namespace URIs.

The variable bindings, function library and namespace declarations used to evaluate a subexpression are always the same as those used to evaluate the containing expression. The context node, context position, and context size used to evaluate a subexpression are sometimes different from those used to evaluate the containing expression. Several kinds of expressions change the context node; only predicates change the context position and context size (see [**2.4 Predicates**]). When the evaluation of a kind of expression is described, it will always be explicitly stated if the context node, context position, and context size change for the evaluation of subexpressions; if nothing is said about the context node, context position, and context size, they remain unchanged for the evaluation of subexpressions of that kind of expression.

XPath expressions often occur in XML attributes. The grammar specified in this section applies to the attribute value after XML 1.0 normalization. So, for example, if the grammar uses the character <, this must not appear in the XML source as < but must be quoted according to XML 1.0 rules by, for example, entering it as `<`. Within expressions, literal strings are delimited by single or double quotation marks, which are also used to delimit XML attributes. To avoid a quotation mark in an expression being interpreted by the XML processor as terminating the attribute value the quotation mark can be entered as a character reference (`&quot;` or `&apos;`). Alternatively, the expression can use single quotation marks if the XML attribute is delimited with double quotation marks or vice-versa.

One important kind of expression is a location path. A location path selects a set of nodes relative to the context node. The result of evaluating an expression that is a location path is the node-set containing the nodes selected by the location path. Location paths can recursively contain expressions that are used to filter sets of nodes. A location path matches the production LocationPath.

In the following grammar, the non-terminals QName and NCName are defined in [XML Names], and S is defined in [XML]. The grammar uses the same EBNF notation as [XML] (except that grammar symbols always have initial capital letters).

Expressions are parsed by first dividing the character string to be parsed into tokens and then parsing the resulting sequence of tokens. Whitespace can be freely used between tokens. The tokenization process is described in [**3.7 Lexical Structure**].

2 Location Paths

Although location paths are not the most general grammatical construct in the language (a LocationPath is a special case of an Expr), they are the most important construct and will therefore be described first.

Every location path can be expressed using a straightforward but rather verbose syntax. There are also a number of syntactic abbreviations that allow common cases to be expressed concisely. This section will explain the semantics of location paths using the unabbreviated syntax. The abbreviated syntax will then be explained by showing how it expands into the unabbreviated syntax (see [**2.5 Abbreviated Syntax**]).

Here are some examples of location paths using the unabbreviated syntax:

- `child::para` selects the `para` element children of the context node
- `child::*` selects all element children of the context node
- `child::text()` selects all text node children of the context node
- `child::node()` selects all the children of the context node, whatever their node type
- `attribute::name` selects the `name` attribute of the context node
- `attribute::*` selects all the attributes of the context node
- `descendant::para` selects the `para` element descendants of the context node
- `ancestor::div` selects all `div` ancestors of the context node
- `ancestor-or-self::div` selects the `div` ancestors of the context node and, if the context node is a `div` element, the context node as well
- `descendant-or-self::para` selects the `para` element descendants of the context node and, if the context node is a `para` element, the context node as well
- `self::para` selects the context node if it is a `para` element, and otherwise selects nothing
- `child::chapter/descendant::para` selects the `para` element descendants of the `chapter` element children of the context node

- `child::*/child::para` selects all `para` grandchildren of the context node
- `/` selects the document root (which is always the parent of the document element)
- `/descendant::para` selects all the `para` elements in the same document as the context node
- `/descendant::olist/child::item` selects all the `item` elements that have an `olist` parent and that are in the same document as the context node
- `child::para[position()=1]` selects the first `para` child of the context node
- `child::para[position()=last()]` selects the last `para` child of the context node
- `child::para[position()=last()-1]` selects the last but one `para` child of the context node
- `child::para[position()>1]` selects all the `para` children of the context node other than the first `para` child of the context node
- `following-sibling::chapter[position()=1]` selects the next `chapter` sibling of the context node
- `preceding-sibling::chapter[position()=1]` selects the previous `chapter` sibling of the context node
- `/descendant::figure[position()=42]` selects the forty-second `figure` element in the document
- `/child::doc/child::chapter[position()=5]/child::section[position()=2]` selects the second `section` of the fifth `chapter` of the `doc` document element
- `child::para[attribute::type="warning"]` selects all `para` children of the context node that have a `type` attribute with value `warning`
- `child::para[attribute::type='warning'][position()=5]` selects the fifth `para` child of the context node that has a `type`

attribute with value `warning`

- `child::para[position()=5][attribute::type="warning"]` selects the fifth `para` child of the context node if that child has a `type` attribute with value `warning`

- `child::chapter[child::title='Introduction']` selects the `chapter` children of the context node that have one or more `title` children with string-value equal to `Introduction`

- `child::chapter[child::title]` selects the `chapter` children of the context node that have one or more `title` children

- `child::*[self::chapter or self::appendix]` selects the `chapter` and `appendix` children of the context node

- `child::*[self::chapter or self::appendix][position()=last()]` selects the last `chapter` or `appendix` child of the context node

There are two kinds of location path: relative location paths and absolute location paths.

A relative location path consists of a sequence of one or more location steps separated by `/`. The steps in a relative location path are composed together from left to right. Each step in turn selects a set of nodes relative to a context node. An initial sequence of steps is composed together with a following step as follows. The initial sequence of steps selects a set of nodes relative to a context node. Each node in that set is used as a context node for the following step. The sets of nodes identified by that step are unioned together. The set of nodes identified by the composition of the steps is this union. For example, `child::div/child::para` selects the `para` element children of the `div` element children of the context node, or, in other words, the `para` element grandchildren that have `div` parents.

An absolute location path consists of `/` optionally followed by a relative location path. A `/` by itself selects the root node of the document containing the context node. If it is followed by a relative location path, then the location path selects the set of nodes that would be selected by the relative location path relative to the root node of the document containing the context node.

Location Paths

[1] LocationPath ::= RelativeLocationPath

			| AbsoluteLocationPath
[2]	AbsoluteLocationPath	::=	'/' RelativeLocationPath?
			| AbbreviatedAbsoluteLocationPath
[3]	RelativeLocationPath	::=	Step
			| RelativeLocationPath '/' Step
			| AbbreviatedRelativeLocationPath

2.1 Location Steps

A location step has three parts:

- an axis, which specifies the tree relationship between the nodes selected by the location step and the context node,
- a node test, which specifies the node type and expanded-name of the nodes selected by the location step, and
- zero or more predicates, which use arbitrary expressions to further refine the set of nodes selected by the location step.

The syntax for a location step is the axis name and node test separated by a double colon, followed by zero or more expressions each in square brackets. For example, in `child::para[position()=1]`, `child` is the name of the axis, `para` is the node test and `[position()=1]` is a predicate.

The node-set selected by the location step is the node-set that results from generating an initial node-set from the axis and node-test, and then filtering that node-set by each of the predicates in turn.

The initial node-set consists of the nodes having the relationship to the context node specified by the axis, and having the node type and expanded-name specified by the node test. For example, a location step `descendant::para` selects the `para` element descendants of the context node: `descendant` specifies that each node in the initial node-set must be a descendant of the context; `para` specifies that each node in the initial node-set must be an element named `para`. The available axes are described in [**2.2 Axes**]. The available node tests are described in [**2.3 Node Tests**]. The meaning of some node tests is dependent on the axis.

The initial node-set is filtered by the first predicate to generate a new node-set; this new node-set is then filtered using the second predicate, and so on. The final node-set is the node-set selected by the location step. The axis

affects how the expression in each predicate is evaluated and so the semantics of a predicate is defined with respect to an axis. See [**2.4 Predicates**].

Location Steps

[4]	Step	::=	AxisSpecifier NodeTest Predicate*
			| AbbreviatedStep
[5]	AxisSpecifier	::=	AxisName '::'
			| AbbreviatedAxisSpecifier

2.2 Axes

The following axes are available:

- the `child` axis contains the children of the context node
- the `descendant` axis contains the descendants of the context node; a descendant is a child or a child of a child and so on; thus the descendant axis never contains attribute or namespace nodes
- the `parent` axis contains the parent of the context node, if there is one
- the `ancestor` axis contains the ancestors of the context node; the ancestors of the context node consist of the parent of context node and the parent's parent and so on; thus, the ancestor axis will always include the root node, unless the context node is the root node
- the `following-sibling` axis contains all the following siblings of the context node; if the context node is an attribute node or namespace node, the `following-sibling` axis is empty
- the `preceding-sibling` axis contains all the preceding siblings of the context node; if the context node is an attribute node or namespace node, the `preceding-sibling` axis is empty
- the `following` axis contains all nodes in the same document as the context node that are after the context node in document order, excluding any descendants and excluding attribute nodes and namespace nodes
- the `preceding` axis contains all nodes in the same document as the

context node that are before the context node in document order, excluding any ancestors and excluding attribute nodes and namespace nodes

- the `attribute` axis contains the attributes of the context node; the axis will be empty unless the context node is an element
- the `namespace` axis contains the namespace nodes of the context node; the axis will be empty unless the context node is an element
- the `self` axis contains just the context node itself
- the `descendant-or-self` axis contains the context node and the descendants of the context node
- the `ancestor-or-self` axis contains the context node and the ancestors of the context node; thus, the ancestor axis will always include the root node

  NOTE: The `ancestor`, `descendant`, `following`, `preceding` and `self` axes partition a document (ignoring attribute and namespace nodes): they do not overlap and together they contain all the nodes in the document.

Axes

```
[6]   AxisName   ::=   'ancestor'
                     | 'ancestor-or-self'
                     | 'attribute'
                     | 'child'
                     | 'descendant'
                     | 'descendant-or-self'
                     | 'following'
                     | 'following-sibling'
                     | 'namespace'
                     | 'parent'
                     | 'preceding'
                     | 'preceding-sibling'
                     | 'self'
```

2.3 Node Tests

Every axis has a **principal node type**. If an axis can contain elements, then the principal node type is element; otherwise, it is the type of the nodes that the axis can contain. Thus,

- For the attribute axis, the principal node type is attribute.
- For the namespace axis, the principal node type is namespace.
- For other axes, the principal node type is element.

A node test that is a QName is true if and only if the type of the node (see [**5 Data Model**]) is the principal node type and has an expanded-name equal to the expanded-name specified by the QName. For example, `child::para` selects the `para` element children of the context node; if the context node has no `para` children, it will select an empty set of nodes. `attribute::href` selects the `href` attribute of the context node; if the context node has no `href` attribute, it will select an empty set of nodes.

A QName in the node test is expanded into an expanded-name using the namespace declarations from the expression context. This is the same way expansion is done for element type names in start and end-tags except that the default namespace declared with `xmlns` is not used: if the QName does not have a prefix, then the namespace URI is null (this is the same way attribute names are expanded). It is an error if the QName has a prefix for which there is no namespace declaration in the expression context.

A node test `*` is true for any node of the principal node type. For example, `child::*` will select all element children of the context node, and `attribute::*` will select all attributes of the context node.

A node test can have the form NCName`:*`. In this case, the prefix is expanded in the same way as with a QName, using the context namespace declarations. It is an error if there is no namespace declaration for the prefix in the expression context. The node test will be true for any node of the principal type whose expanded-name has the namespace URI to which the prefix expands, regardless of the local part of the name.

The node test `text()` is true for any text node. For example, `child::text()` will select the text node children of the context node. Similarly, the node test `comment()` is true for any comment node, and the node test `processing-instruction()` is true for any processing instruction. The `processing-instruction()` test may have an argument that is Literal; in this case, it is true for any processing instruction that has a name equal to the value of the Literal.

A node test `node()` is true for any node of any type whatsoever.

[7]	NodeTest	::=	NameTest
			| NodeType '(' ')'
			| 'processing-instruction' '(' Literal ')'

2.4 Predicates

An axis is either a forward axis or a reverse axis. An axis that only ever contains the context node or nodes that are after the context node in document order is a forward axis. An axis that only ever contains the context node or nodes that are before the context node in document order is a reverse axis. Thus, the ancestor, ancestor-or-self, preceding, and preceding-sibling axes are reverse axes; all other axes are forward axes. Since the self axis always contains at most one node, it makes no difference whether it is a forward or reverse axis. The **proximity position** of a member of a node-set with respect to an axis is defined to be the position of the node in the node-set ordered in document order if the axis is a forward axis and ordered in reverse document order if the axis is a reverse axis. The first position is 1.

A predicate filters a node-set with respect to an axis to produce a new node-set. For each node in the node-set to be filtered, the PredicateExpr is evaluated with that node as the context node, with the number of nodes in the node-set as the context size, and with the proximity position of the node in the node-set with respect to the axis as the context position; if PredicateExpr evaluates to true for that node, the node is included in the new node-set; otherwise, it is not included.

A PredicateExpr is evaluated by evaluating the Expr and converting the result to a boolean. If the result is a number, the result will be converted to true if the number is equal to the context position and will be converted to false otherwise; if the result is not a number, then the result will be converted as if by a call to the **boolean** function. Thus a location path `para[3]` is equivalent to `para[position()=3]`.

Predicates

[8]	Predicate	::=	'[' PredicateExpr ']'
[9]	PredicateExpr	::=	Expr

2.5 Abbreviated Syntax

Here are some examples of location paths using abbreviated syntax:

- `para` selects the `para` element children of the context node
- `*` selects all element children of the context node
- `text()` selects all text node children of the context node
- `@name` selects the `name` attribute of the context node
- `@*` selects all the attributes of the context node
- `para[1]` selects the first `para` child of the context node
- `para[last()]` selects the last `para` child of the context node
- `*/para` selects all `para` grandchildren of the context node
- `/doc/chapter[5]/section[2]` selects the second `section` of the fifth `chapter` of the `doc`
- `chapter//para` selects the `para` element descendants of the `chapter` element children of the context node
- `//para` selects all the `para` descendants of the document root and thus selects all `para` elements in the same document as the context node
- `//olist/item` selects all the `item` elements in the same document as the context node that have an `olist` parent
- `.` selects the context node
- `.//para` selects the `para` element descendants of the context node
- `..` selects the parent of the context node
- `../@lang` selects the `lang` attribute of the parent of the context node
- `para[@type="warning"]` selects all `para` children of the context node that have a `type` attribute with value `warning`
- `para[@type="warning"][5]` selects the fifth `para` child of the

context node that has a `type` attribute with value `warning`

- `para[5][@type="warning"]` selects the fifth `para` child of the context node if that child has a `type` attribute with value `warning`

- `chapter[title="Introduction"]` selects the `chapter` children of the context node that have one or more `title` children with string-value equal to `Introduction`

- `chapter[title]` selects the `chapter` children of the context node that have one or more `title` children

- `employee[@secretary and @assistant]` selects all the `employee` children of the context node that have both a `secretary` attribute and an `assistant` attribute

The most important abbreviation is that `child::` can be omitted from a location step. In effect, `child` is the default axis. For example, a location path `div/para` is short for `child::div/child::para`.

There is also an abbreviation for attributes: `attribute::` can be abbreviated to `@`. For example, a location path `para[@type="warning"]` is short for `child::para[attribute::type="warning"]` and so selects `para` children with a `type` attribute with value equal to `warning`.

`//` is short for `/descendant-or-self::node()/`. For example, `//para` is short for `/descendant-or-self::node()/child::para` and so will select any `para` element in the document (even a `para` element that is a document element will be selected by `//para` since the document element node is a child of the root node); `div//para` is short for `div/descendant-or-self::node()/child::para` and so will select all `para` descendants of `div` children.

> **NOTE:** The location path `//para[1]` does *not* mean the same as the location path `/descendant::para[1]`. The latter selects the first descendant `para` element; the former selects all descendant `para` elements that are the first `para` children of their parents.

A location step of `.` is short for `self::node()`. This is particularly useful in conjunction with `//`. For example, the location path `.//para` is short for

```
self::node()/descendant-or-self::node()/child::para
```

and so will select all `para` descendant elements of the context node.

Similarly, a location step of `..` is short for `parent::node()`. For example, `../title` is short for `parent::node()/child::title` and so will select the `title` children of the parent of the context node.

Abbreviations

[10]	AbbreviatedAbsoluteLocationPath	::=	'//' RelativeLocationPath
[11]	AbbreviatedRelativeLocationPath	::=	RelativeLocationPath '//' Step
[12]	AbbreviatedStep	::=	'.' | '..'
[13]	AbbreviatedAxisSpecifier	::=	'@'?

3 Expressions

3.1 Basics

A VariableReference evaluates to the value to which the variable name is bound in the set of variable bindings in the context. It is an error if the variable name is not bound to any value in the set of variable bindings in the expression context.

Parentheses may be used for grouping.

[14]	Expr	::=	OrExpr
[15]	PrimaryExpr	::=	VariableReference | '(' Expr ')' | Literal | Number | FunctionCall

3.2 Function Calls

A FunctionCall expression is evaluated by using the FunctionName to identify a function in the expression evaluation context function library, evaluating

each of the Arguments, converting each argument to the type required by the function, and finally calling the function, passing it the converted arguments. It is an error if the number of arguments is wrong or if an argument cannot be converted to the required type. The result of the FunctionCall expression is the result returned by the function.

An argument is converted to type string as if by calling the **string** function. An argument is converted to type number as if by calling the **number** function. An argument is converted to type boolean as if by calling the **boolean** function. An argument that is not of type node-set cannot be converted to a node-set.

[16]	FunctionCall	::=	FunctionName '(' (Argument (',' Argument)*)? ')'
[17]	Argument	::=	Expr

3.3 Node-sets

A location path can be used as an expression. The expression returns the set of nodes selected by the path.

The | operator computes the union of its operands, which must be node-sets.

Predicates are used to filter expressions in the same way that they are used in location paths. It is an error if the expression to be filtered does not evaluate to a node-set. The Predicate filters the node-set with respect to the child axis.

> **NOTE:** The meaning of a Predicate depends crucially on which axis applies. For example, `preceding::foo[1]` returns the first `foo` element in *reverse document order*, because the axis that applies to the `[1]` predicate is the preceding axis; by contrast, `(preceding::foo)[1]` returns the first `foo` element in *document order*, because the axis that applies to the `[1]` predicate is the child axis.

The `/` and `//` operators compose an expression and a relative location path. It is an error if the expression does not evaluate to a node-set. The `/` operator does composition in the same way as when `/` is used in a location path. As in location paths, `//` is short for `/descendant-or-self::node()/`.

There are no types of objects that can be converted to node-sets.

[18]	UnionExpr	::=	PathExpr
			| UnionExpr '|' PathExpr
[19]	PathExpr	::=	LocationPath
			| FilterExpr
			| FilterExpr '/' RelativeLocationPath
			| FilterExpr '//' RelativeLocationPath
[20]	FilterExpr	::=	PrimaryExpr
			| FilterExpr Predicate

3.4 Booleans

An object of type boolean can have one of two values, true and false.

An `or` expression is evaluated by evaluating each operand and converting its value to a boolean as if by a call to the **boolean** function. The result is true if either value is true and false otherwise. The right operand is not evaluated if the left operand evaluates to true.

An `and` expression is evaluated by evaluating each operand and converting its value to a boolean as if by a call to the **boolean** function. The result is true if both values are true and false otherwise. The right operand is not evaluated if the left operand evaluates to false.

An EqualityExpr (that is not just a RelationalExpr) or a RelationalExpr (that is not just an AdditiveExpr) is evaluated by comparing the objects that result from evaluating the two operands. Comparison of the resulting objects is defined in the following three paragraphs. First, comparisons that involve node-sets are defined in terms of comparisons that do not involve node-sets; this is defined uniformly for =, !=, <=, <, >= and >. Second, comparisons that do not involve node-sets are defined for = and !=. Third, comparisons that do not involve node-sets are defined for <=, <, >= and >.

If both objects to be compared are node-sets, then the comparison will be true if and only if there is a node in the first node-set and a node in the second node-set such that the result of performing the comparison on the string-values of the two nodes is true. If one object to be compared is a node-set and the other is a number, then the comparison will be true if and only if there is a node in the node-set such that the result of performing the comparison on the number to be compared and on the result of converting the string-value of that node to a number using the **number** function is true. If one object to be compared is a node-set and the other is a string, then the comparison will be

true if and only if there is a node in the node-set such that the result of performing the comparison on the string-value of the node and the other string is true. If one object to be compared is a node-set and the other is a boolean, then the comparison will be true if and only if the result of performing the comparison on the boolean and on the result of converting the node-set to a boolean using the **boolean** function is true.

When neither object to be compared is a node-set and the operator is = or `!=`, then the objects are compared by converting them to a common type as follows and then comparing them. If at least one object to be compared is a boolean, then each object to be compared is converted to a boolean as if by applying the **boolean** function. Otherwise, if at least one object to be compared is a number, then each object to be compared is converted to a number as if by applying the **number** function. Otherwise, both objects to be compared are converted to strings as if by applying the **string** function. The = comparison will be true if and only if the objects are equal; the `!=` comparison will be true if and only if the objects are not equal. Numbers are compared for equality according to IEEE 754 [IEEE 754]. Two booleans are equal if either both are true or both are false. Two strings are equal if and only if they consist of the same sequence of UCS characters.

> **NOTE:** If `$x` is bound to a node-set, then `$x="foo"` does not mean the same as `not($x!="foo")`: the former is true if and only if *some* node in `$x` has the string-value `foo`; the latter is true if and only if *all* nodes in `$x` have the string-value `foo`.

When neither object to be compared is a node-set and the operator is <=, <, >= or >, then the objects are compared by converting both objects to numbers and comparing the numbers according to IEEE 754. The < comparison will be true if and only if the first number is less than the second number. The <= comparison will be true if and only if the first number is less than or equal to the second number. The > comparison will be true if and only if the first number is greater than the second number. The >= comparison will be true if and only if the first number is greater than or equal to the second number.

> **NOTE:** When an XPath expression occurs in an XML document, any < and <= operators must be quoted according to XML 1.0 rules by using, for example, `<` and `<=`. In the following example the value of the `test` attribute is an XPath expression:
>
> ```
> <xsl:if test="@value < 10">...</xsl:if>
> ```

[21] OrExpr ::= AndExpr
| OrExpr 'or' AndExpr

```
[22]  AndExpr          ::=  EqualityExpr
                          | AndExpr 'and' EqualityExpr
[23]  EqualityExpr     ::=  RelationalExpr
                          | EqualityExpr '=' RelationalExpr
                          | EqualityExpr '!=' RelationalExpr
[24]  RelationalExpr   ::=  AdditiveExpr
                          | RelationalExpr '<' AdditiveExpr
                          | RelationalExpr '>' AdditiveExpr
                          | RelationalExpr '<=' AdditiveExpr
                          | RelationalExpr '>=' AdditiveExpr
```

NOTE: The effect of the above grammar is that the order of precedence is (lowest precedence first):

- `or`
- `and`
- `=, !=`
- `<=, <, >=, >`

and the operators are all left associative. For example, `3 > 2 > 1` is equivalent to `(3 > 2) > 1`, which evaluates to false.

3.5 Numbers

A number represents a floating-point number. A number can have any double-precision 64-bit format IEEE 754 value [IEEE 754]. These include a special "Not-a-Number" (NaN) value, positive and negative infinity, and positive and negative zero. See Section 4.2.3 of [JLS] for a summary of the key rules of the IEEE 754 standard.

The numeric operators convert their operands to numbers as if by calling the **number** function.

The + operator performs addition.

The - operator performs subtraction.

NOTE: Since XML allows `-` in names, the `-` operator typically needs to be preceded by whitespace. For example, `foo-bar` evaluates to a node-set containing the child elements named `foo-bar`; `foo - bar` evaluates to the difference of the result of converting the string-value of the first `foo` child element to a number and the result of converting the string-value of the first `bar` child to a number.

The `div` operator performs floating-point division according to IEEE 754.

The `mod` operator returns the remainder from a truncating division. For example,

- `5 mod 2` returns `1`
- `5 mod -2` returns `1`
- `-5 mod 2` returns `-1`
- `-5 mod -2` returns `-1`

NOTE: This is the same as the % operator in Java and ECMAScript.

NOTE: This is not the same as the IEEE 754 remainder operation, which returns the remainder from a rounding division.

Numeric Expressions

[25]	AdditiveExpr	::=	MultiplicativeExpr
			| AdditiveExpr '+' MultiplicativeExpr
			| AdditiveExpr '-' MultiplicativeExpr
[26]	MultiplicativeExpr	::=	UnaryExpr
			| MultiplicativeExpr MultiplyOperator UnaryExpr
			| MultiplicativeExpr 'div' UnaryExpr
			| MultiplicativeExpr 'mod' UnaryExpr
[27]	UnaryExpr	::=	UnionExpr
			| '-' UnaryExpr

3.6 Strings

Strings consist of a sequence of zero or more characters, where a character is defined as in the XML Recommendation [XML]. A single character in XPath thus corresponds to a single Unicode abstract character with a single corresponding Unicode scalar value (see [Unicode]); this is not the same thing as a 16-bit Unicode code value: the Unicode coded character representation for an abstract character with Unicode scalar value greater that U+FFFF is a pair of 16-bit Unicode code values (a surrogate pair). In many programming languages, a string is represented by a sequence of 16-bit Unicode code values; implementations of XPath in such languages must take care to ensure that a surrogate pair is correctly treated as a single XPath character.

> **NOTE:** It is possible in Unicode for there to be two strings that should be treated as identical even though they consist of the distinct sequences of Unicode abstract characters. For example, some accented characters may be represented in either a precomposed or decomposed form. Therefore, XPath expressions may return unexpected results unless both the characters in the XPath expression and in the XML document have been normalized into a canonical form. See [Character Model].

3.7 Lexical Structure

When tokenizing, the longest possible token is always returned.

For readability, whitespace may be used in expressions even though not explicitly allowed by the grammar: ExprWhitespace may be freely added within patterns before or after any ExprToken.

The following special tokenization rules must be applied in the order specified to disambiguate the ExprToken grammar:

- If there is a preceding token and the preceding token is not one of `@`, `::`, `(`, `[`, `,` or an Operator, then a `*` must be recognized as a MultiplyOperator and an NCName must be recognized as an OperatorName.

- If the character following an NCName (possibly after intervening ExprWhitespace) is `(`, then the token must be recognized as a NodeType or a FunctionName.

- If the two characters following an NCName (possibly after intervening ExprWhitespace) are `::`, then the token must be recognized as an AxisName.

- Otherwise, the token must not be recognized as a MultiplyOperator, an OperatorName, a NodeType, a FunctionName, or an AxisName.

Expression Lexical Structure

[28]	ExprToken	::=	'(' | ')' | '[' | ']' | '.' | '..' | '@' | ',' | '::' | NameTest | NodeType | Operator | FunctionName | AxisName | Literal | Number | VariableReference
[29]	Literal	::=	'"' [^"]* '"' | "'" [^']* "'"
[30]	Number	::=	Digits ('.' Digits?)? | '.' Digits
[31]	Digits	::=	[0-9]+
[32]	Operator	::=	OperatorName | MultiplyOperator | '/' | '//' | '|' | '+' | '-' | '=' | '!=' | '<' | '<=' | '>' | '>='
[33]	OperatorName	::=	'and' | 'or' | 'mod' | 'div'
[34]	MultiplyOperator	::=	'*'
[35]	FunctionName	::=	QName - NodeType
[36]	VariableReference	::=	'$' QName
[37]	NameTest	::=	'*' | NCName ':' '*' | QName
[38]	NodeType	::=	'comment' | 'text' | 'processing-instruction'

			| 'node'
[39]	ExprWhitespace	::=	S

4 Core Function Library

This section describes functions that XPath implementations must always include in the function library that is used to evaluate expressions.

Each function in the function library is specified using a function prototype, which gives the return type, the name of the function, and the type of the arguments. If an argument type is followed by a question mark, then the argument is optional; otherwise, the argument is required.

4.1 Node Set Functions

Function: *number* **last**()

The **last** function returns a number equal to the context size from the expression evaluation context.

Function: *number* **position**()

The **position** function returns a number equal to the context position from the expression evaluation context.

Function: *number* **count**(*node-set*)

The **count** function returns the number of nodes in the argument node-set.

Function: *node-set* **id**(*object*)

The **id** function selects elements by their unique ID (see [**5.2.1 Unique IDs**]). When the argument to **id** is of type node-set, then the result is the union of the result of applying **id** to the string-value of each of the nodes in the argument node-set. When the argument to **id** is of any other type, the argument is converted to a string as if by a call to the **string** function; the string is split into a whitespace-separated list of tokens (whitespace is any sequence of characters matching the production S); the result is a node-set containing the elements in the same document as the context node that have a unique ID equal to any of the tokens in the list.

- `id("foo")` selects the element with unique ID `foo`

- `id("foo")/child::para[position()=5]` selects the fifth `para` child of the element with unique ID `foo`

Function: *string* **local-name**(*node-set*?)

The **local-name** function returns the local part of the expanded-name of the node in the argument node-set that is first in document order. If the argument node-set is empty or the first node has no expanded-name, an empty string is returned. If the argument is omitted, it defaults to a node-set with the context node as its only member.

Function: *string* **namespace-uri**(*node-set*?)

The **namespace-uri** function returns the namespace URI of the expanded-name of the node in the argument node-set that is first in document order. If the argument node-set is empty, the first node has no expanded-name, or the namespace URI of the expanded-name is null, an empty string is returned. If the argument is omitted, it defaults to a node-set with the context node as its only member.

> **NOTE:** The string returned by the **namespace-uri** function will be empty except for element nodes and attribute nodes.

Function: *string* **name**(*node-set*?)

The **name** function returns a string containing a QName representing the expanded-name of the node in the argument node-set that is first in document order. The QName must represent the expanded-name with respect to the namespace declarations in effect on the node whose expanded-name is being represented. Typically, this will be the QName that occurred in the XML source. This need not be the case if there are namespace declarations in effect on the node that associate multiple prefixes with the same namespace. However, an implementation may include information about the original prefix in its representation of nodes; in this case, an implementation can ensure that the returned string is always the same as the QName used in the XML source. If the argument node-set is empty or the first node has no expanded-name, an empty string is returned. If the argument it omitted, it defaults to a node-set with the context node as its only member.

> **NOTE:** The string returned by the **name** function will be the same as the string returned by the **local-name** function except for element nodes and attribute nodes.

4.2 String Functions

Function: *string* **string**(*object*?)

The **string** function converts an object to a string as follows:

- A node-set is converted to a string by returning the string-value of the node in the node-set that is first in document order. If the node-set is empty, an empty string is returned.
- A number is converted to a string as follows
  - NaN is converted to the string `NaN`
  - positive zero is converted to the string `0`
  - negative zero is converted to the string `0`
  - positive infinity is converted to the string `Infinity`
  - negative infinity is converted to the string `-Infinity`
  - if the number is an integer, the number is represented in decimal form as a Number with no decimal point and no leading zeros, preceded by a minus sign (`-`) if the number is negative
  - otherwise, the number is represented in decimal form as a Number including a decimal point with at least one digit before the decimal point and at least one digit after the decimal point, preceded by a minus sign (`-`) if the number is negative; there must be no leading zeros before the decimal point apart possibly from the one required digit immediately before the decimal point; beyond the one required digit after the decimal point there must be as many, but only as many, more digits as are needed to uniquely distinguish the number from all other IEEE 754 numeric values.
- The boolean false value is converted to the string `false`. The boolean true value is converted to the string `true`.
- An object of a type other than the four basic types is converted to a string in a way that is dependent on that type.

If the argument is omitted, it defaults to a node-set with the context node as its only member.

> **NOTE:** The `string` function is not intended for converting numbers into strings for presentation to users. The `format-number` function and `xsl:number` element in [XSLT] provide this functionality.

Function: *string* **concat**(*string*, *string*, *string**)

The **concat** function returns the concatenation of its arguments.

Function: *boolean* **starts-with**(*string*, *string*)

The **starts-with** function returns true if the first argument string starts with the second argument string, and otherwise returns false.

Function: *boolean* **contains**(*string*, *string*)

The **contains** function returns true if the first argument string contains the second argument string, and otherwise returns false.

Function: *string* **substring-before**(*string*, *string*)

The **substring-before** function returns the substring of the first argument string that precedes the first occurrence of the second argument string in the first argument string, or the empty string if the first argument string does not contain the second argument string. For example, `substring-before("1999/04/01","/")` returns `1999`.

Function: *string* **substring-after**(*string*, *string*)

The **substring-after** function returns the substring of the first argument string that follows the first occurrence of the second argument string in the first argument string, or the empty string if the first argument string does not contain the second argument string. For example, `substring-after("1999/04/01","/")` returns `04/01`, and `substring-after("1999/04/01","19")` returns `99/04/01`.

Function: *string* **substring**(*string*, *number*, *number*?)

The **substring** function returns the substring of the first argument starting at the position specified in the second argument with length specified in the third

argument. For example, `substring("12345",2,3)` returns `"234"`. If the third argument is not specified, it returns the substring starting at the position specified in the second argument and continuing to the end of the string. For example, `substring("12345",2)` returns `"2345"`.

More precisely, each character in the string (see [**3.6 Strings**]) is considered to have a numeric position: the position of the first character is 1, the position of the second character is 2 and so on.

> **NOTE:** This differs from Java and ECMAScript, in which the `String.substring` method treats the position of the first character as 0.

The returned substring contains those characters for which the position of the character is greater than or equal to the rounded value of the second argument and, if the third argument is specified, less than the sum of the rounded value of the second argument and the rounded value of the third argument; the comparisons and addition used for the above follow the standard IEEE 754 rules; rounding is done as if by a call to the **round** function. The following examples illustrate various unusual cases:

- `substring("12345", 1.5, 2.6)` returns `"234"`
- `substring("12345", 0, 3)` returns `"12"`
- `substring("12345", 0 div 0, 3)` returns `""`
- `substring("12345", 1, 0 div 0)` returns `""`
- `substring("12345", -42, 1 div 0)` returns `"12345"`
- `substring("12345", -1 div 0, 1 div 0)` returns `""`

Function: *number* **string-length**(*string*?)

The **string-length** returns the number of characters in the string (see [**3.6 Strings**]). If the argument is omitted, it defaults to the context node converted to a string, in other words the string-value of the context node.

Function: *string* **normalize-space**(*string*?)

The **normalize-space** function returns the argument string with whitespace normalized by stripping leading and trailing whitespace and replacing sequences of whitespace characters by a single space. Whitespace

characters are the same as those allowed by the S production in XML. If the argument is omitted, it defaults to the context node converted to a string, in other words the string-value of the context node.

Function: *string* **translate**(*string*, *string*, *string*)

The **translate** function returns the first argument string with occurrences of characters in the second argument string replaced by the character at the corresponding position in the third argument string. For example, `translate("bar","abc","ABC")` returns the string `BAr`. If there is a character in the second argument string with no character at a corresponding position in the third argument string (because the second argument string is longer than the third argument string), then occurrences of that character in the first argument string are removed. For example, `translate("--aaa--","abc-","ABC")` returns `"AAA"`. If a character occurs more than once in the second argument string, then the first occurrence determines the replacement character. If the third argument string is longer than the second argument string, then excess characters are ignored.

> **NOTE:** The **translate** function is not a sufficient solution for case conversion in all languages. A future version of XPath may provide additional functions for case conversion.

4.3 Boolean Functions

Function: *boolean* **boolean**(*object*)

The **boolean** function converts its argument to a boolean as follows:

- a number is true if and only if it is neither positive or negative zero nor NaN
- a node-set is true if and only if it is non-empty
- a string is true if and only if its length is non-zero
- an object of a type other than the four basic types is converted to a boolean in a way that is dependent on that type

Function: *boolean* **not**(*boolean*)

The **not** function returns true if its argument is false, and false otherwise.

Function: *boolean* **true**()

The **true** function returns true.

Function: *boolean* **false**()

The **false** function returns false.

Function: *boolean* **lang**(*string*)

The **lang** function returns true or false depending on whether the language of the context node as specified by `xml:lang` attributes is the same as or is a sublanguage of the language specified by the argument string. The language of the context node is determined by the value of the `xml:lang` attribute on the context node, or, if the context node has no `xml:lang` attribute, by the value of the `xml:lang` attribute on the nearest ancestor of the context node that has an `xml:lang` attribute. If there is no such attribute, then **lang** returns false. If there is such an attribute, then **lang** returns true if the attribute value is equal to the argument ignoring case, or if there is some suffix starting with `-` such that the attribute value is equal to the argument ignoring that suffix of the attribute value and ignoring case. For example, `lang("en")` would return true if the context node is any of these five elements:

```
<para xml:lang="en"/>
<div xml:lang="en"><para/></div>
<para xml:lang="EN"/>
<para xml:lang="en-us"/>
```

4.4 Number Functions

Function: *number* **number**(*object*?)

The **number** function converts its argument to a number as follows:

- a string that consists of optional whitespace followed by an optional minus sign followed by a Number followed by whitespace is converted to the IEEE 754 number that is nearest (according to the IEEE 754 round-to-nearest rule) to the mathematical value represented by the string; any other string is converted to NaN
- boolean true is converted to 1; boolean false is converted to 0
- a node-set is first converted to a string as if by a call to the **string**

function and then converted in the same way as a string argument

- an object of a type other than the four basic types is converted to a number in a way that is dependent on that type

If the argument is omitted, it defaults to a node-set with the context node as its only member.

> **NOTE:** The **number** function should not be used for conversion of numeric data occurring in an element in an XML document unless the element is of a type that represents numeric data in a language-neutral format (which would typically be transformed into a language-specific format for presentation to a user). In addition, the **number** function cannot be used unless the language-neutral format used by the element is consistent with the XPath syntax for a Number.

Function: *number* **sum**(*node-set*)

The **sum** function returns the sum, for each node in the argument node-set, of the result of converting the string-values of the node to a number.

Function: *number* **floor**(*number*)

The **floor** function returns the largest (closest to positive infinity) number that is not greater than the argument and that is an integer.

Function: *number* **ceiling**(*number*)

The **ceiling** function returns the smallest (closest to negative infinity) number that is not less than the argument and that is an integer.

Function: *number* **round**(*number*)

The **round** function returns the number that is closest to the argument and that is an integer. If there are two such numbers, then the one that is closest to positive infinity is returned. If the argument is NaN, then NaN is returned. If the argument is positive infinity, then positive infinity is returned. If the argument is negative infinity, then negative infinity is returned. If the argument is positive zero, then positive zero is returned. If the argument is negative zero, then negative zero is returned. If the argument is less than zero, but greater than or equal to -0.5, then negative zero is returned.

> **NOTE:** For these last two cases, the result of calling the **round** function is not the same as the result of adding 0.5 and then calling the **floor** function.

5 Data Model

XPath operates on an XML document as a tree. This section describes how XPath models an XML document as a tree. This model is conceptual only and does not mandate any particular implementation. The relationship of this model to the XML Information Set [XML Infoset] is described in [**B XML Information Set Mapping**].

XML documents operated on by XPath must conform to the XML Namespaces Recommendation [XML Names].

The tree contains nodes. There are seven types of node:

- root nodes
- element nodes
- text nodes
- attribute nodes
- namespace nodes
- processing instruction nodes
- comment nodes

For every type of node, there is a way of determining a **string-value** for a node of that type. For some types of node, the string-value is part of the node; for other types of node, the string-value is computed from the string-value of descendant nodes.

> **NOTE:** For element nodes and root nodes, the string-value of a node is not the same as the string returned by the DOM `nodeValue` method (see [DOM]).

Some types of node also have an **expanded-name**, which is a pair consisting of a local part and a namespace URI. The local part is a string. The namespace URI is either null or a string. The namespace URI specified in the

XML document can be a URI reference as defined in [RFC2396]; this means it can have a fragment identifier and can be relative. A relative URI should be resolved into an absolute URI during namespace processing: the namespace URIs of expanded-names of nodes in the data model should be absolute. Two expanded-names are equal if they have the same local part, and either both have a null namespace URI or both have non-null namespace URIs that are equal.

There is an ordering, **document order**, defined on all the nodes in the document corresponding to the order in which the first character of the XML representation of each node occurs in the XML representation of the document after expansion of general entities. Thus, the root node will be the first node. Element nodes occur before their children. Thus, document order orders element nodes in order of the occurrence of their start-tag in the XML (after expansion of entities). The attribute nodes and namespace nodes of an element occur before the children of the element. The namespace nodes are defined to occur before the attribute nodes. The relative order of namespace nodes is implementation-dependent. The relative order of attribute nodes is implementation-dependent. **Reverse document order** is the reverse of document order.

Root nodes and element nodes have an ordered list of child nodes. Nodes never share children: if one node is not the same node as another node, then none of the children of the one node will be the same node as any of the children of another node. Every node other than the root node has exactly one **parent**, which is either an element node or the root node. A root node or an element node is the parent of each of its child nodes. The **descendants** of a node are the children of the node and the descendants of the children of the node.

5.1 Root Node

The root node is the root of the tree. A root node does not occur except as the root of the tree. The element node for the document element is a child of the root node. The root node also has as children processing instruction and comment nodes for processing instructions and comments that occur in the prolog and after the end of the document element.

The string-value of the root node is the concatenation of the string-values of all text node descendants of the root node in document order.

The root node does not have an expanded-name.

5.2 Element Nodes

There is an element node for every element in the document. An element node has an expanded-name computed by expanding the QName of the element specified in the tag in accordance with the XML Namespaces Recommendation [XML Names]. The namespace URI of the element's expanded-name will be null if the QName has no prefix and there is no applicable default namespace.

> **NOTE:** In the notation of Appendix A.3 of [XML Names], the local part of the expanded-name corresponds to the `type` attribute of the `ExpEType` element; the namespace URI of the expanded-name corresponds to the `ns` attribute of the `ExpEType` element, and is null if the `ns` attribute of the `ExpEType` element is omitted.

The children of an element node are the element nodes, comment nodes, processing instruction nodes and text nodes for its content. Entity references to both internal and external entities are expanded. Character references are resolved.

The string-value of an element node is the concatenation of the string-values of all text node descendants of the element node in document order.

5.2.1 Unique IDs

An element node may have a unique identifier (ID). This is the value of the attribute that is declared in the DTD as type `ID`. No two elements in a document may have the same unique ID. If an XML processor reports two elements in a document as having the same unique ID (which is possible only if the document is invalid) then the second element in document order must be treated as not having a unique ID.

> **NOTE:** If a document does not have a DTD, then no element in the document will have a unique ID.

5.3 Attribute Nodes

Each element node has an associated set of attribute nodes; the element is the parent of each of these attribute nodes; however, an attribute node is not a child of its parent element.

> **NOTE:** This is different from the DOM, which does not treat the element bearing an attribute as the parent of the attribute (see [DOM]).

Elements never share attribute nodes: if one element node is not the same node as another element node, then none of the attribute nodes of the one element node will be the same node as the attribute nodes of another element node.

> **NOTE:** The = operator tests whether two nodes have the same value, *not* whether they are the same node. Thus attributes of two different elements may compare as equal using =, even though they are not the same node.

A defaulted attribute is treated the same as a specified attribute. If an attribute was declared for the element type in the DTD, but the default was declared as `#IMPLIED`, and the attribute was not specified on the element, then the element's attribute set does not contain a node for the attribute.

Some attributes, such as `xml:lang` and `xml:space`, have the semantics that they apply to all elements that are descendants of the element bearing the attribute, unless overridden with an instance of the same attribute on another descendant element. However, this does not affect where attribute nodes appear in the tree: an element has attribute nodes only for attributes that were explicitly specified in the start-tag or empty-element tag of that element or that were explicitly declared in the DTD with a default value.

An attribute node has an expanded-name and a string-value. The expanded-name is computed by expanding the QName specified in the tag in the XML document in accordance with the XML Namespaces Recommendation [XML Names]. The namespace URI of the attribute's name will be null if the QName of the attribute does not have a prefix.

> **NOTE:** In the notation of Appendix A.3 of [XML Names], the local part of the expanded-name corresponds to the `name` attribute of the `ExpAName` element; the namespace URI of the expanded-name corresponds to the `ns` attribute of the `ExpAName` element, and is null if the `ns` attribute of the `ExpAName` element is omitted.

An attribute node has a string-value. The string-value is the normalized value as specified by the XML Recommendation [XML]. An attribute whose normalized value is a zero-length string is not treated specially: it results in an attribute node whose string-value is a zero-length string.

> **NOTE:** It is possible for default attributes to be declared in an external DTD or an external parameter entity. The XML Recommendation does not require an XML processor to read an external DTD or an external parameter unless it is validating. A

stylesheet or other facility that assumes that the XPath tree contains default attribute values declared in an external DTD or parameter entity may not work with some non-validating XML processors.

There are no attribute nodes corresponding to attributes that declare namespaces (see [XML Names]).

5.4 Namespace Nodes

Each element has an associated set of namespace nodes, one for each distinct namespace prefix that is in scope for the element (including the `xml` prefix, which is implicitly declared by the XML Namespaces Recommendation [XML Names]) and one for the default namespace if one is in scope for the element. The element is the parent of each of these namespace nodes; however, a namespace node is not a child of its parent element. Elements never share namespace nodes: if one element node is not the same node as another element node, then none of the namespace nodes of the one element node will be the same node as the namespace nodes of another element node. This means that an element will have a namespace node:

- for every attribute on the element whose name starts with `xmlns:`;
- for every attribute on an ancestor element whose name starts `xmlns:` unless the element itself or a nearer ancestor redeclares the prefix;
- for an `xmlns` attribute, if the element or some ancestor has an `xmlns` attribute, and the value of the `xmlns` attribute for the nearest such element is non-empty

  NOTE: An attribute `xmlns=""` "undeclares" the default namespace (see [XML Names]).

A namespace node has an expanded-name: the local part is the namespace prefix (this is empty if the namespace node is for the default namespace); the namespace URI is always null.

The string-value of a namespace node is the namespace URI that is being bound to the namespace prefix; if it is relative, it must be resolved just like a namespace URI in an expanded-name.

5.5 Processing Instruction Nodes

There is a processing instruction node for every processing instruction, except for any processing instruction that occurs within the document type declaration.

A processing instruction has an expanded-name: the local part is the processing instruction's target; the namespace URI is null. The string-value of a processing instruction node is the part of the processing instruction following the target and any whitespace. It does not include the terminating ?>.

> **NOTE:** The XML declaration is not a processing instruction. Therefore, there is no processing instruction node corresponding to the XML declaration.

5.6 Comment Nodes

There is a comment node for every comment, except for any comment that occurs within the document type declaration.

The string-value of comment is the content of the comment not including the opening <!-- or the closing -->.

A comment node does not have an expanded-name.

5.7 Text Nodes

Character data is grouped into text nodes. As much character data as possible is grouped into each text node: a text node never has an immediately following or preceding sibling that is a text node. The string-value of a text node is the character data. A text node always has at least one character of data.

Each character within a CDATA section is treated as character data. Thus, <![CDATA[<]]> in the source document will treated the same as <. Both will result in a single < character in a text node in the tree. Thus, a CDATA section is treated as if the <![CDATA[and]]> were removed and every occurrence of < and & were replaced by < and & respectively.

> **NOTE:** When a text node that contains a < character is written out as XML, the < character must be escaped by, for example, using <, or including it in a CDATA section.

Characters inside comments, processing instructions and attribute values do not produce text nodes. Line-endings in external entities are normalized to

#xA as specified in the XML Recommendation [XML].

A text node does not have an expanded-name.

6 Conformance

XPath is intended primarily as a component that can be used by other specifications. Therefore, XPath relies on specifications that use XPath (such as [XPointer] and [XSLT]) to specify criteria for conformance of implementations of XPath and does not define any conformance criteria for independent implementations of XPath.

A References

A.1 Normative References

IEEE 754
Institute of Electrical and Electronics Engineers. *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std 754-1985.

RFC2396
T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*. IETF RFC 2396. See http://www.ietf.org/rfc/rfc2396.txt.

XML
World Wide Web Consortium. *Extensible Markup Language (XML) 1.0.* W3C Recommendation. See http://www.w3.org/TR/1998/REC-xml-19980210

XML Names
World Wide Web Consortium. *Namespaces in XML.* W3C Recommendation. See http://www.w3.org/TR/REC-xml-names

A.2 Other References

Character Model
World Wide Web Consortium. *Character Model for the World Wide Web.* W3C Working Draft. See http://www.w3.org/TR/WD-charmod

DOM
World Wide Web Consortium. *Document Object Model (DOM) Level 1 Specification.* W3C Recommendation. See http://www.w3.org/TR/REC-DOM-Level-1

JLS

J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. See http://java.sun.com/docs/books/jls/index.html.

ISO/IEC 10646

ISO (International Organization for Standardization). *ISO/IEC 10646-1:1993, Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane*. International Standard. See http://www.iso.ch/cate/d18741.html.

TEI

C.M. Sperberg-McQueen, L. Burnard *Guidelines for Electronic Text Encoding and Interchange*. See http://etext.virginia.edu/TEI.html.

Unicode

Unicode Consortium. *The Unicode Standard*. See http://www.unicode.org/unicode/standard/standard.html.

XML Infoset

World Wide Web Consortium. *XML Information Set.* W3C Working Draft. See http://www.w3.org/TR/xml-infoset

XPointer

World Wide Web Consortium. *XML Pointer Language (XPointer).* W3C Working Draft. See http://www.w3.org/TR/WD-xptr

XQL

J. Robie, J. Lapp, D. Schach. *XML Query Language (XQL)*. See http://www.w3.org/TandS/QL/QL98/pp/xql.html

XSLT

World Wide Web Consortium. *XSL Transformations (XSLT).* W3C Recommendation. See http://www.w3.org/TR/xslt

B XML Information Set Mapping (Non-Normative)

The nodes in the XPath data model can be derived from the information items provided by the XML Information Set [XML Infoset] as follows:

> **NOTE:** A new version of the XML Information Set Working Draft, which will replace the May 17 version, was close to completion at the time when the preparation of this version of XPath was completed and was expected to be released at the same time or shortly after the release of this version of XPath. The mapping is given for this new version of the XML Information Set Working Draft. If the new version of the XML Information Set Working has not yet been released, W3C members may consult the internal Working Group version http://www.w3.org/XML/Group/1999/09/WD-xml-infoset-19990915.html (members only).

- The root node comes from the document information item. The children

of the root node come from the *children* and *children - comments* properties.

- An element node comes from an element information item. The children of an element node come from the *children* and *children - comments* properties. The attributes of an element node come from the *attributes* property. The namespaces of an element node come from the *in-scope namespaces* property. The local part of the expanded-name of the element node comes from the *local name* property. The namespace URI of the expanded-name of the element node comes from the *namespace URI* property. The unique ID of the element node comes from the *children* property of the attribute information item in the *attributes* property that has an *attribute type* property equal to `ID`.

- An attribute node comes from an attribute information item. The local part of the expanded-name of the attribute node comes from the *local name* property. The namespace URI of the expanded-name of the attribute node comes from the *namespace URI* property. The string-value of the node comes from concatenating the *character code* property of each member of the *children* property.

- A text node comes from a sequence of one or more consecutive character information items. The string-value of the node comes from concatenating the *character code* property of each of the character information items.

- A processing instruction node comes from a processing instruction information item. The local part of the expanded-name of the node comes from the *target* property. (The namespace URI part of the expanded-name of the node is null.) The string-value of the node comes from the *content* property. There are no processing instruction nodes for processing instruction items that are children of document type declaration information item.

- A comment node comes from a comment information item. The string-value of the node comes from the *content* property. There are no comment nodes for comment information items that are children of document type declaration information item.

- A namespace node comes from a namespace declaration information item. The local part of the expanded-name of the node comes from the *prefix* property. (The namespace URI part of the expanded-name of the node is null.) The string-value of the node comes from the *namespace*

URI property.

J2EE
JDBC Technology

Downloads
- Early Access
- Tools

Reference
- API Specifications
- Documentation
- FAQs
- Code Samples & Apps
- Technical Articles & Tips
- Industry Support

Community
- Books & Authors
- Code Certification
- Forums
- Bug Database

Learning
- Tutorials & Code Camps
- Online Sessions & Courses
- Instructor-Led Courses
- Quizzes

Regional Sites - 日本語

JDBC technology is an API (included in both J2SE and J2EE releases) that provides cross-DBMS connectivity to a wide range of SQL databases and access to other tabular data sources, such as spreadsheets or flat files. With a JDBC technology-enabled driver, you can connect all corporate data even in a heterogeneous environment. » Read More

Notes for Developers
Interested in certifying your JDBC driver or finding a JDBC driver for use with J2EE compatible products? Check out the JDBC certification program. Our driver database is now searchable by supported JDBC API version, driver type, DBMS support, and features.

See also our page covering the relationship of JDBC to a related technology: Java Data Objects (JDO).

What's New

7 April 2004
NEW **JDBC RowSet Implementations 1.0** Available through the Java Community Process (JCP) site, this spec defines the APIs for implementations of the Rowset interface for passing tabular data between distributed tiers and components. Also defines the XML Schema that will be used to represent the Rowset objects that are passed between components.
» Read more

10 June 2003
JDBC 4.0 Specification - JSR 221 Filed The JDBC 4.0 API specification seeks to improve Java application access to SQL data stores by the provision of ease-of-development focused features and improvements at both the utility and API level. Using new Java language features planned for JSR-175, generics defined in JSR-014, a set of JDBC utility classes, SQL-savvy developers will be able to more easily access SQL data sources while still benefiting from the full power of the JDBC API.
» Read more

8 April 2004
NEW **JDBC Optional Package for CDC/Foundation Profile**
The JDBC API for CDC / FP Optional Package defines a subset of the JDBC 3.0 API that can be used in conjunction with the Java 2 Micro Edition (J2ME) Connected Device Configuration / Foundation Profile (CDC / FP).

9 May 2002
JDBC API 3.0 Specification - Final Release The JDBC API v. 3.0 is included in J2SE 1.4. The JDBC Optional Package will be packaged with the core API and be included as part of J2SE. Other major changes in the new version of the JDBC specification include: Connection Pool configuration enhancements; Statement pooling for Pooled Connections; and a description of the migration path from the JDBC SPI (Service Provider Interface) to the Connector Architecture.

24 October 2002

Related Links

Popular Downloads
- J2SE 1.5.0 Beta 1
- J2SE 1.4.2
- Java Web Services Developer Pack 1.3

Technical Topics
- Performance
- Web Services
- Security
- Desktop

Products and Technologies
- J2SE 1.5
- Desktop Java
- Sun Java Desktop System
- Sun Java Studio Standard IDE

Sun Resources
- java.net
- Java Upgrade Program
- New to Java Center
- Professional Certification
- Professional Training
- JavaOne Online
- Java Community Process

JDBC Connector Early Access The JDBC Connector enables any JDBC driver to be packaged as a J2EE Connector Architecture-compliant connector, and it makes it easy to plug any JDBC driver into an application server compliant with the J2EE platform.

Community

Events

2004 JavaOne Conference. San Francisco , CA Be there! Join the thousands of developers worldwide who come to the JavaOne conference each year in San Francisco to immerse themselves in Java technology, the latest innovations, the community, and the learning opportunities. » Read More

Subscribe to Newsletters. Members of Sun Developer Network can sign up to receive these (and other) newsletters. Not yet a member? **Join us!**

Enterprise Java Technologies Newsletter
Learn about new enterprise Java technologies, products, tools, and resources for developers.

Enterprise Java Technologies Tech Tips
Get expert tips, sample code solutions, and techniques for developing in the Java 2 Platform, Enterprise Edition (J2EE).
» Read More

JDBC Interest

JDBC API Mailing List Learn more about the JDBC API. To subscribe, send email to listserv@java.sun.com with this info in the body of the message: `subscribe JDBC-INTEREST yourlastname yourfirstname`.

Core Java

Java Data Objects (JDO)

Downloads
- Choose...

Reference
- Documentation
- FAQs

Community
- Bug Database
- Forums

Learning
- Tutorials & Code Camps

The Java Data Objects (JDO) API is a standard interface-based Java model abstraction of persistence, developed as Java Specification Request 12 JSR 12 under the auspices of the Java Community Process. If you are an application programmer, you can use JDO technology to directly store your Java domain model instances into the persistent store (database). » Read More

Benefits of Using JDO for Application Programming
Applications written with the JDO API are portable, and independent of the underlying database. You can focus on your domain object model and leave the details of persistence (field-by-field storage of objects) to the JDO implementation.

What's New

September 2003
JDO Specification (JSR 12) Download the final release of the spec from the Java Community Process (JCP) site. JDO 1.0.1 has been released.

Community

JDO Central
Online resource for JDO developers. Find out the latest on JDO from experts, fellow users, and vendors. Chats, news, and more.

Events
2004 JavaOne Conference. San Francisco , CA Be there! Join the thousands of developers worldwide who come to the JavaOne conference each year in San Francisco to immerse themselves in Java technology, the latest innovations, the community, and the learning opportunities. » Read More

Subscribe to Newsletters. Members of Sun Developer Network can sign up to receive these (and other) newsletters. Not yet a member? **Join us!**

Java Technology Fundamentals
New to Java? Learn the basics of the Java programming language and keep up-to-date on additions to the New-to-Java Programming Center.

Core Java Technologies Newsletter
Find out about new enterprise Java technologies, products, tools, and resources for developers.

Core Java Technologies Tech Tips
Get expert tips, sample code solutions, and techniques for developing in the Java 2 Platform, Standard Edition (J2SE).
» Read More

Related Links

Popular Downloads
- J2SE 1.5.0 Beta 1
- J2SE 1.4.2
- Java Web Services Developer Pack 1.3

Technical Topics
- Performance
- Web Services
- Security
- Desktop

Products and Technologies
- J2SE 1.5
- Desktop Java
- Sun Java Desktop System
- Sun Java Studio Standard IDE

Sun Resources
- java.net
- Java Upgrade Program
- New to Java Center
- Professional Certification
- Professional Training
- JavaOne Online
- Java Community Process

Java Technology

Java 2 Platform, Enterprise Edition (J2EE)

Downloads
- Early Access

Reference
- API Specifications
- Documentation
- FAQs
- Code Samples & Apps
- BluePrints
- Technical Articles & Tips
- White Papers
- » **See all**

Community
- Bug Database
- Books & Authors
- Newsletters
- Forums

Learning
- Tutorials & Code Camps
- Online Sessions & Courses
- Instructor-Led Courses
- Quizzes
- Course Certification

Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing component-based multitier enterprise applications. Features include Web services support and development tools (SDK). » Read More

→ Get the SDK **→ Get the Specification (PDF)**

Get J2EE Technology Training and Certification
J2EE training can lead to one of three certifications: Sun Certified Web Component Developer, Sun Certified Business Component Developer, or Sun Certified Enterprise Architect. Find out more.

What's New

May 12, 2004
Sun Java System Application Server Platform Edition 8 and NetBeans 3.6 Bundle
This bundle includes NetBeans IDE, our powerful integrated development environment for developing applications on the Java platform, the Sun Java System Application Server Platform Edition 8 FCS, and a plug-in to connect the NetBeans IDE to the Sun Java System Application Server 8.

April 27, 2004
Online Chat Transcript
View the transcript of the the April 27 online chat with guests Tony Ng and Paul Ko on the J2EE 1.4 SDK and Sun Java System Application Server Platform Edition 8.

April 26, 2004
Java Application Verification Kit (AVK) for the Enterprise 1.4 FCS
Download the latest version of the AVK with the new installation wizard and portability features for testing web services. The Java AVK 1.4 FCS is available for free.

April 26, 2004
Java Adventure Builder Reference application v1.0.1 (AVK Release)
A sample application verified by the Java Application Verification Kit (AVK) for the Enterprise for portability across all J2EE-compatible application servers

March 22, 2004
The J2EE 1.4 SDK Now Available
The J2EE 1.4 SDK contains the Sun Java System Application Server Platform Edition 8 FCS, the J2SE 1.4.2 SDK, J2EE 1.4 platform API documentation, and a slew of samples to help developers learn about the J2EE platform and technologies and prototype J2EE applications. The J2EE 1.4 SDK is free for both development and deployment.

In addition to being a fully compliant implementation of the J2EE 1.4 platform, the Sun Java System Application Server Platform Edition 8 FCS includes two new technologies for building the Web tier of J2EE applications: JavaServer Faces, which simplifies building user interfaces by providing reusable UI

Related Links

Popular Downloads
- J2EE 1.4 SDK
- J2EE 1.3 SDK
- Java Web Services Developer Pack 1.3

Compatibility & Java Verification
- Java Verification Program
- Java Application Verification Kit (AVK)
- Compatible Implementations

Products
- Sun Java System Application Server 8
- Sun Java System Application Server 7
- Sun Java Studio Enterprise
- Sun Java Studio Standard
- Sun Java System Message Queue 3.5

Technologies
- Enterprise JavaBeans
- JavaServer Faces
- JavaServer Pages
- Java Servlet

Technical Topics
- Web Services
- Performance
- Security
- Desktop

Sun Resources
- Professional Training
- Professional Certification
- Developer Tech Support

components for use in JSP pages and the JavaServer Pages Standard Tag Library (JSTL), which encapsulates core functionality common to applications that use JSP pages.

March 22, 2004

The J2EE 1.4 Tutorial

The J2EE 1.4 Tutorial is for programmers interested in developing and deploying J2EE applications on the J2EE 1.4 SDK. The tutorial describes how to develop JavaServer Faces applications for the J2EE 1.4 SDK, how to use Web services technologies, and how to write applications using the new versions of J2EE platform technologies (Servlet 2.4, JSP 2.0, EJB 2.1, JMS 1.1, J2EE Connector 1.5).

March 22, 2004

The Migration Tool for the Sun Java System Application Server

Download this tool to automate the migration of J2EE applications from other J2EE application servers to the Sun Java System Application Server Platform Edition 8.

March 3, 2004

JavaServer Faces Now Shipping

After receiving approval for JSR-127 from the Executive Committee of the Java Community Process, JavaServer Faces is now available for download. Under the terms of the license, the JavaServer Faces Reference Implementation is free to redistribute in binary form. This means you can include the JavaServer Faces Reference Implementation with your software distribution. Click here to learn more about JavaServer Faces.

February 11, 2004

J2EE Client Provisioning

J2EE Client Provisioning technology extends J2EE platform to enable the development of provisioning portals that manage and distribute applications and content to a variety of client devices. This Final Release implementation of the specification for J2EE Client Provisioning includes support for J2ME MIDP and J2SE devices. It allows developers to experiment with the technology, and includes a sample provisioning portal and detailed getting started and installation instructions.

February 5, 2004

Java Adventure Builder Reference application v1.0

Java Adventure Builder Reference application v1.0 is now available for download from the Java BluePrints Program. The Java BluePrints program provides guidelines, patterns, and code for real-world application scenarios, enabling you to build robust, scalable, and portable solutions. Java Adventure Builder Reference application is a J2EE 1.4 application that showcases how to design interoperable and portable Web services on the J2EE 1.4 platform. It also presents recommendations for the effective use of J2EE technologies such as JAX-RPC, JAXP, Servlet 2.4, JSP 2.0, JSTL and EJB. Documentation including an online version of the Java Blueprints book, Designing Web Services with the J2EE 1.4 Platform, is also available.

November 24, 2003

J2EE 1.4 Platform Specification Final Release

The Java 2 Platform, Enterprise Edition (J2EE) specification (JSR 151) for version 1.4 is now final, after receiving unanimous approval from the Executive Committee of the Java Community Process. The Final Release specifications describe a set of requirements that J2EE platform products must meet. It specifies platform architecture, application components, containers, set of standard J2EE services, interoperability requirements and platform contracts (APIs, SPIs, network protocols and deployment descriptors). New specifications in J2EE 1.4 Platform include: Java API for XML-Based RPC (JAX-RPC), SOAP with

Attachments API for Java (SAAJ), Web Services for J2EE, J2EE Management Model, J2EE Deployment API, Java Management Extensions (JMX), J2EE Authorization Contract for Containers, Java API for XML Registries (JAXR).

Platform Releases

- J2EE 1.4
- J2EE 1.3
- J2EE 1.2

Web Services Technologies

- Java API for XML Processing (JAXP)
- Java API for XML Registries (JAXR)
- Java API for XML-based RPC (JAX-RPC)
- SOAP with Attachments API for Java (SAAJ)

Component Model Technologies

- Java Servlet
- JavaServer Pages
- JavaServer Faces
- Enterprise JavaBeans
- Java Message Service
- J2EE Connector Architecture

Management Technologies

- J2EE Deployment Specification
- J2EE Management Specification
- J2EE Client Provisioning
- Java Authorization Contract for Containers

Other J2EE Technologies

- JDBC
- Java Data Objects (JDO)
- JavaMail
- Transactions

Related Technologies and Products

- Java 2 Platform, Enterprise Edition Tools
- ECPerf

Community

Events

2004 JavaOne Conference. San Francisco , CA Be there! Join the thousands of developers worldwide who come to the JavaOne conference each year in San Francisco to immerse themselves in Java technology, the latest innovations, the community, and the learning opportunities. » Read More

Subscribe to Newsletters. Members of Sun Developer Network can sign up to receive these (and other) newsletters. Not yet a member? **Join us!**

Enterprise Java Technologies Newsletter
Learn about new enterprise Java technologies, products, tools, and resources for developers.

Enterprise Java Technologies Tech Tips
Get expert tips, sample code solutions, and techniques for developing in the Java 2 Platform, Enterprise Edition (J2EE).
» Read More

Looking for an Older Release?

Sun provides some older product and technology releases as a courtesy to developers for problem resolution. These releases and products have completed the Sun "end-of-life" (EOL) process and are no longer supported under standard support contracts. However, you can access them via our archives.

A Sun Developer Network Site

XML Content Feeds

J2EE

Java Message Service (JMS)

Downloads

Reference
- Documentation
- FAQs
- Licensees

Community
- Forums
- Bug Database

Learning
- Instructor-Led Courses
- Tutorials & Code Camps
- Online Sessions & Courses

The Java Message Service (JMS) API is a messaging standard that allows application components based on the Java 2 Platform, Enterprise Edition (J2EE) to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous. » Read More

→ **Get the SDK**

→ **Get the Specification (PDF)**

What's New

May 10, 2004
Sun Java System Message Queue 3.5 Now Available for download Check out Sun's latest Enterprise Messaging software offering, available for free download.

November 20, 2003
Java Technology Courses Check out Sun's latest J2EE technology courseware offerings, especially those that lead to certification.

Community

Events
2004 JavaOne Conference. San Francisco , CA Be there! Join the thousands of developers worldwide who come to the JavaOne conference each year in San Francisco to immerse themselves in Java technology, the latest innovations, the community, and the learning opportunities. » Read More

Subscribe to Newsletters. Members of Sun Developer Network can sign up to receive these (and other) newsletters. Not yet a member? **Join us!**

Enterprise Java Technologies Newsletter
Learn about new enterprise Java technologies, products, tools, and resources for developers.

Enterprise Java Technologies Tech Tips
Get expert tips, sample code solutions, and techniques for developing in the Java 2 Platform, Enterprise Edition (J2EE).
» Read More

JMS Provider With release 1.4 of the J2EE platform, the JMS provider may be integrated with the application server using the J2EE Connector Architecture. For more information, see the Enterprise JavaBeans Specification v 2.1, and the J2EE Connector Architecture Specification v 1.5.

Related Links

Popular Downloads
- J2EE 1.4 SDK
- JMS 1.1 SDK

Products and Technologies
- Sun Java System Message Queue 3.5
- Java Transaction API
- J2EE Connector Architecture
- Enterprise JavaBeans
- JDBC Data Access API
- Java Naming and Directory Interface

Sun Resources
- Developer Technical Support
- Downloads for Application Development

Core Java

Java Remote Method Invocation (Java RMI)

Downloads

Reference
- Documentation
- White Papers

Community
- Forums
- Bug Database

Learning
- Tutorials & Code Camps

Java Remote Method Invocation (Java RMI) enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines*, possibly on different hosts. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism.

Releases of Java RMI
Java RMI is available for Java 2 Platform, Standard Edition (J2SE) and Java 2 Platform, Micro Edition (J2ME).

Community

Events
2004 JavaOne Conference. San Francisco , CA Be there! Join the thousands of developers worldwide who come to the JavaOne conference each year in San Francisco to immerse themselves in Java technology, the latest innovations, the community, and the learning opportunities. » Read More

Feedback and Participation
Got comments or questions? Write to us.

Visit our email archive of the RMI-USERS mailing list.

Related Links

Popular Downloads
- J2SE 1.5.0 Beta 1
- J2SE 1.4.2
- Java Web Services Developer Pack 1.3

Technical Topics
- Performance
- Web Services
- Security
- Desktop

Products and Technologies
- J2SE 1.5
- Desktop Java
- Sun Java Desktop System
- Sun Java Studio Standard IDE

Sun Resources
- java.net
- Java Upgrade Program
- New to Java Center
- Professional Certification
- Professional Training
- JavaOne Online
- Java Community Process

Web Services Activity

Groups · News · Documents · Technical discussion · Events

The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as ***Web services***.

The goal of the Web Services Activity is to develop a set of technologies in order to lead Web services to their full potential. The Web Services Activity Statement explains the W3C's work on this topic in more detail.

Note: The XML Protocol Activity was incorporated in the Web Services Activity in January 2002.

Groups

The Activity is currently composed of three Working Groups and one Interest Group, coordinated by one Coordination Group:

- XML Protocol Working Group.
- Web Services Description Working Group.
- Web Services Choreography Working Group.
- Semantic Web Services Interest Group.

Detailed information about the work of each of these groups can be found on their respective home pages.

News

- ***2004-04-28***: The XML Protocol Working Group published the first Working Draft of SOAP Resource Representation Header.
-

2004-04-27: The Web Services Choreography Working Group published the first Working Draft of Web Services Choreography Description Language Version 1.0.

- ***2004-03-26***: The Web Services Description Working Group published new Working Drafts of Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language and Web Services Description Language (WSDL) Version 2.0 Part 2: Message Exchange Patterns.
- ***2004-03-24***: The Web Services Choreography Working Group published the first Working Draft of WS Choreography Model Overview.
- ***2004-03-11***: The Web Services Choreography Working Group published the second Working Draft of Web Services Choreography Requirements.
- ***2004-02-13***: Handling Privacy In WSDL 2.0 was published as a Team Submission.
- ***2004-02-11***: The Web Services Architecture Working Group published the Web Services Architecture and supporting documents, the Web Services Glossary, Web Services Architecture Requirements, Web Services Architecture Usage Scenarios, Web Service Management: Service Life Cycle as Working Group Notes.
- ***2004-02-09***: The XML Protocol Working Group published XML-binary Optimized Packaging and SOAP Message Transmission Optimization Mechanism as Working Drafts.

Previous news items are available on the history page.

Documents

Documents produced by the Web Services Architecture Working Group

- Web Services Architecture (Working Group Note)
- Web Services Architecture Requirements (Working Group Note)
- Web Services Glossary (Working Group Note)
- Web Services Architecture Usage Scenarios (Working Group Note)
- Web Service Management: Service Life Cycle (Working Group Note)

Documents produced by the XML Protocol Working Group

W3C Recommendations

- SOAP Version 1.2 Part 0: Primer (Recommendation)
- SOAP Version 1.2 Part 1: Messaging Framework (Recommendation)
- SOAP Version 1.2 Part 2: Adjuncts (Recommendation)
- SOAP Version 1.2 Specification Assertions and Test Collection (Recommendation)

Other documents

- SOAP Message Transmission Optimization Mechanism (Working Draft)
- XML-binary Optimized Packaging
- SOAP Optimized Serialization Use Cases and Requirements (Working Draft)
- SOAP Resource Representation Header
- XML Protocol (XMLP) Requirements Document (Working Group Note)
- XML Protocol Abstract Model (Working Draft)
- XML Protocol Usage Scenarios (Working Group Note)
- SOAP 1.2 Attachment Feature (Last Call Working Draft; **review period ends 15 October 2002**)
- SOAP Version 1.2 Email Binding (Note)
- SOAP Version 1.2 Message Normalization (Note)

Documents produced by the Web Services Description Working Group

- Web Service Description Requirements (Last Call Working Draft; **review period ends 31 December 2002**)
- Web Service Description Usage Scenarios (Working Draft)
- Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language (Working Draft)
- Web Services Description Language (WSDL) Version 2.0 Part 2: Message Exchange Patterns (Working Draft)
- Web Services Description Language (WSDL) Version 1.2 Part 3: Bindings (Working Draft)

Documents produced by the Web Services Choreography Working Group

- Web Services Choreography Requirements 1.0 (Working Draft)
- WS Choreography Model Overview (Working Draft)
- Web Services Choreography Description Language Version 1.0

Technical discussion

General technical discussions and announcements about Web services happen on the www-ws publicly archived mailing list, by sending email to **www-ws@w3.org**.

The www-ws archive can be searched for:

Search Help!

For specific Working Group discussions, please refer to the Groups' home pages:

Mailing list	Purpose
www-ws@w3.org	General Web services discussions
www-ws-arch@w3.org	Discussion of the Web Services Architecture and other documents produced by Web Services Architecture Working Group
xml-dist-app@w3.org	Discussion list for the XML Protocol Working Group
www-ws-desc@w3.org	Discussion list for the Web Services Description Working Group
public-ws-chor@w3.org	Discussion list for the Web Services Choreography Working Group
www-ws-cg@w3.org	Public list for the Web Services Coordination Group
public-sws-ig@w3.org	Discussion list for the Semantic Web Services Interest Group

Events

See also the news section.

As a result of the increased interest in Web services, W3C and others have recently organized several related activities including:

- ***October 2003***: Semantic Web Services Interest Group launched
- June 2003: SOAP Version 1.2 released as a W3C Recommendation (press release, testimonials, changes and benefits, FAQ)
- January 2003: Creation of the Web Services Choreography Working Group.
- August 2002: W3C acknowledges WSCI submission as a W3C Note: Web Service Choreography Interface (WSCI) 1.0.
- January 2002: Creation of the Web Services Activity and of the Web Services Architecture and Web Services Description Working Groups, and the Web Services Coordination Group.
- April 11-12, 2001: W3C holds a Workshop on Web services in San Jose, CA.
- March 2001: W3C acknowledges WSDL submission as a W3C Note: Web Services Description Language (WSDL) 1.1 and creates the www-ws mailing list for Web services discussions.
- September 2000: Creation of the XML Protocol Working Group.
- May 15-19, 2000: W3C organizes a panel on XML Protocols at WWW9 in Amsterdam.
- May 2000: W3C acknowledges SOAP submission as a W3C Note: Simple Object Access Protocol (SOAP) 1.1.
- March 2000: W3C provides an initial comparison of the various initiatives already in the marketplace: XML Protocol Comparisons
- March 2000: Michael Condry, Sun, organizes a B2BXML BOF at IETF 47 in Adelaide, Australia, 26-31
- February/March 2000: W3C organizes an XML protocols BOF at XTech 2000
- February 2000: W3C provides an interim plan for XML protocol work
- December 1999: W3C creates the xml-dist-app@w3.org mailing list for discussion of XML protocol related issues.

Hugo Haas <hugo@w3.org>, Activity Lead

$Id: Overview.html,v 1.127 2004/04/29 11:44:46 hugo Exp $

HyperText Markup Language (HTML) Home Page

This is W3C's home page for the HTML Activity. Here you will find pointers to our specifications for HTML/XHTML, guidelines on how to use HTML/XHTML to the best effect, and pointers to related work at W3C. When W3C decides to become involved in an area of Web technology or policy, it initiates an activity in that area. HTML is one of many Activities currently being pursued. You can learn more about the HTML Activity from the HTML Activity Statement.

news | Recommendations | public drafts | test suites | tutorials | slides | guidelines | validation | articles | translations | charter | working group | roadmap | XForms | forums | HTML Tidy | related work | HTML 4.0/3.2/2.0 | historical

NEWS

26 April 2004: The first public release of the test suite for XHTML-Print is now available. Please send comments to www-html-testsuite@w3.org (archive).

18 February 2004: A Working Draft of Modularization of XHTML 1.0 - Second Edition, a revision of the W3C Recommendation Modularization of XHTML, has been published for community review. This document clarifies and makes corrections based on nearly three years of use by the community. It also incorporated an implementation of the abstract modules using XML Schemas, previously published as Modularization of XHTML in XML Schema. The HTML WG expects to advance this specification to Proposed Edited Recommendation after incorporating feedback on this Working Draft. Please send error reports to www-html-editor@w3.org (archive).

20 January 2004: The XHTML-Print specification has been published as a Candidate Recommendation. XHTML-Print is designed to be

appropriate for printing from mobile devices to low-cost printers that might not have a full-page buffer and that generally print from top-to-bottom and left-to-right with the paper in a portrait orientation. XHTML-Print is also targeted at printing in environments where it is not feasible or desirable to install a printer-specific driver and where some variability in the formatting of the output is acceptable. Please send implementation feedback to www-html-editor@w3.org (archive).

14 October 2003: **The World Wide Web Consortium today released XML Events specification as a Recommendation.** The XML Events module defined in this specification provides XML languages with the ability to uniformly integrate event listeners and associated event handlers with DOM2 event interfaces. The specification has been reviewed by the W3C Membership, who favor its adoption by industry.

3 October 2003: The second Last Call Working Draft of Modularization of XHTML in XML Schema has been published. It is being re-submitted for Last Call because of substantial changes in the way the Schemas are implemented to ease their use in non-XHTML context. The Last Call review period ends 14 November 2003. Please send Last Call comments to www-html-editor@w3.org (archive).

23 September 2003: W3C Launched the HTML Patent Advisory Group (PAG) to study issues for HTML-related specifications raised by the US court case of Eolas v. Microsoft and US Patent 5,838,906. Public discussion takes place on the public-web-plugins mailing list. The FAQ on US Patent 5,838,906 and the W3C is available.

(Past News)

What is HTML?

HTML is the *lingua franca* for publishing hypertext on the World Wide Web. It is a non-proprietary format based upon SGML, and can be created and processed by a wide range of tools, from simple plain text editors - you type it in from scratch- to sophisticated WYSIWYG authoring tools. HTML uses tags such as <h1> and </h1> to structure text into headings, paragraphs, lists, hypertext links etc. Here is a 10-minute guide for newcomers to HTML. W3C's statement of direction for HTML is given on the HTML Activity Statement. See also the page on our work on the next generation of Web forms, and the section on Web history.

What is XHTML?

The Extensible HyperText Markup Language (XHTML™) is a family of current and future document types and modules that reproduce, subset, and extend HTML, reformulated in XML. XHTML Family document types are all XML-based, and ultimately are designed to work in conjunction with XML-based user agents. XHTML is the successor of HTML, and a series of specifications has been developed for XHTML.

Mission of the HTML Working Group

The mission of the HTML Working Group (members only) is to develop the next generation of HTML as a suite of XML tag sets with a clean migration path from HTML 4. Some of the expected benefits include: reduced authoring costs, an improved match to database & workflow applications, a modular solution to the increasingly disparate capabilities of browsers, and the ability to cleanly integrate HTML with other XML applications. For further information, see the Charter for the HTML Working Group.

Note. *The HTML Working Group Charter has been renewed in August 2002.*

Recommendations

W3C produces what are known as "Recommendations". These are specifications, developed by W3C working groups, and then reviewed by Members of the Consortium. A W3C Recommendation indicates that consensus has been reached among the Consortium Members that a specification is appropriate for widespread use.

XHTML 1.0 | HTML 4.01 | XHTML Basic | Modularization of XHTML | XHTML 1.1 | XML Events

XHTML 1.0

XHTML 1.0 is the W3C's first Recommendation for XHTML, following on from earlier work on HTML 4.01, HTML 4.0, HTML 3.2 and HTML 2.0. With a wealth of features, XHTML 1.0 is a reformulation of HTML 4.01 in XML, and combines the strength of HTML 4 with the power of XML.

XHTML 1.0 is the first major change to HTML since HTML 4.0 was released in 1997. It brings the rigor of XML to Web pages and is the keystone in W3C's work to create standards that provide richer Web pages on an ever increasing range of browser platforms including cell phones, televisions, cars, wallet sized wireless communicators, kiosks, and desktops.

XHTML 1.0 is the first step and the HTML Working Group is busy on the next. XHTML 1.0 reformulates HTML as an XML application. This makes it easier to process and easier to maintain. XHTML 1.0 borrows elements and attributes from W3C's earlier work on HTML 4, and can be interpreted by existing browsers, by following a few simple guidelines. This allows you to start using XHTML now!

You can roll over your old HTML documents into XHTML using an Open Source HTML Tidy utility. This tool also cleans up markup errors, removes clutter and prettifies the markup making it easier to maintain.

Three "flavors" of XHTML 1.0:

XHTML 1.0 is specified in three "flavors". You specify which of these variants you are using by inserting a line at the beginning of the document. For example, the HTML for this document starts with a line which says that it is using XHTML 1.0 Strict. Thus, if you want to validate the document, the tool used knows which variant you are using. Each variant has its own DTD - Document Type Definition - which sets out the rules and regulations for using HTML in a succinct and definitive manner.

- **XHTML 1.0 Strict** - Use this when you want really clean structural mark-up, free of any markup associated with layout. Use this together with W3C's Cascading Style Sheet language (CSS) to get the font, color, and layout effects you want.

- **XHTML 1.0 Transitional** - Many people writing Web pages for the general public to access might want to use this flavor of XHTML 1.0. The idea is to take advantage of XHTML features including style sheets but nonetheless to make small adjustments to your markup for the benefit of those viewing your pages with older browsers which can't understand style sheets. These include using the `body` element with `bgcolor`, `text` and `link` attributes.

- **XHTML 1.0 Frameset** - Use this when you want to use Frames to partition the browser window into two or more frames.

The complete XHTML 1.0 specification is available in English in several formats, including HTML, PostScript and PDF. See also the list of translations produced by volunteers.

HTML 4.01

HTML 4.01 is a revision of the HTML 4.0 Recommendation first released on 18th December 1997. The revision fixes minor errors that have been found since then. The XHTML 1.0 spec relies on HTML 4.01 for the meanings of XHTML elements and attributes. This allowed us to reduce the size of the XHTML 1.0 spec very considerably.

XHTML Basic

XHTML Basic is the second Recommendation in a series of XHTML specifications.

The XHTML Basic document type includes the minimal set of modules required to be an XHTML Host Language document type, and in addition it includes images, forms, basic tables, and object support. It is designed for Web clients that do not support the full set of XHTML features; for example, Web clients such as mobile phones, PDAs, pagers, and settop boxes. The document type is rich enough for content authoring.

XHTML Basic is designed as a common base that may be extended. For example, an event module that is more generic than the traditional HTML 4 event system could be added or it could be extended by additional modules from XHTML Modularization such as the Scripting Module. The goal of XHTML Basic is to serve as a common language supported by various kinds of user agents.

The document type definition is implemented using XHTML modules as defined in "*Modularization of XHTML*".

The complete XHTML Basic specification is available in English in several formats, including HTML, plain text, PostScript and PDF. See also the list of translations produced by volunteers.

Modularization of XHTML

Modularization of XHTML is the third Recommendation in a series of XHTML specifications.

This Recommendation specifies an abstract modularization of XHTML and an implementation of the abstraction using XML Document Type Definitions (DTDs). This modularization provides a means for subsetting and extending XHTML, a feature needed for extending XHTML's reach onto emerging platforms.

Modularization of XHTML will make it easier to combine with markup tags for things like vector graphics, multimedia, math, electronic commerce and more. Content providers will find it easier to produce content for a wide range of

platforms, with better assurances as to how the content is rendered.

The modular design reflects the realization that a one-size-fits-all approach will no longer work in a world where browsers vary enormously in their capabilities. A browser in a cellphone can't offer the same experience as a top of the range multimedia desktop machine. The cellphone doesn't even have the memory to load the page designed for the desktop browser.

See also an overview of XHTML Modularization.

XHTML 1.1 - Module-based XHTML

This Recommendation defines a new XHTML document type that is based upon the module framework and modules defined in Modularization of XHTML. The purpose of this document type is to serve as the basis for future extended XHTML 'family' document types, and to provide a consistent, forward-looking document type cleanly separated from the deprecated, legacy functionality of HTML 4 that was brought forward into the XHTML 1.0 document types.

This document type is essentially a reformulation of XHTML 1.0 Strict using XHTML Modules. This means that many facilities available in other XHTML Family document types (e.g., XHTML Frames) are not available in this document type. These other facilities are available through modules defined in Modularization of XHTML, and document authors are free to define document types based upon XHTML 1.1 that use these facilities (see Modularization of XHTML for information on creating new document types).

What is the difference between XHTML 1.0, XHTML Basic and XHTML 1.1?

The first step was to reformulate HTML 4 in XML, resulting in XHTML 1.0. By following the HTML Compatibility Guidelines set forth in Appendix C of the XHTML 1.0 specification, XHTML 1.0 documents could be compatible with existing HTML user agents.

The next step is to modularize the elements and attributes into convenient collections for use in documents that combine XHTML with other tag sets. The modules are defined in Modularization of XHTML. XHTML Basic is an example of fairly minimal build of these modules and is targeted at mobile applications.

XHTML 1.1 is an example of a larger build of the modules, avoiding many of the presentation features. While XHTML 1.1 looks very similar to XHTML 1.0 Strict, it is designed to serve as the basis for future extended XHTML Family document types, and its modular design makes it easier to add other modules as needed or integrate itself into other markup languages. XHTML 1.1 plus MathML 2.0

document type is an example of such XHTML Family document type.

XML Events

Note: This specification was renamed from "XHTML Events".

The XML Events module defined in this specification provides XML languages with the ability to uniformly integrate event listeners and associated event handlers with Document Object Model (DOM) Level 2 event interfaces. The result is to provide an interoperable way of associating behaviors with document-level markup.

Previous Versions of HTML

HTML 4.0

First released as a W3C Recommendation on 18 December 1997. A second release was issued on 24 April 1998 with changes limited to editorial corrections. **This specification has now been superseded by HTML 4.01.**

HTML 3.2

W3C's first Recommendation for HTML which represented the consensus on HTML features for 1996. HTML 3.2 added widely-deployed features such as tables, applets, text-flow around images, superscripts and subscripts, while providing backwards compatibility with the existing HTML 2.0 Standard.

HTML 2.0

HTML 2.0 (RFC 1866) was developed by the IETF's HTML Working Group, which closed in 1996. It set the standard for core HTML features based upon current practice in 1994. Note that with the release of RFC 2854, RFC 1866 has been obsoleted and its current status is **HISTORIC**.

ISO HTML

ISO/IEC 15445:2000 is a subset of HTML 4, standardized by ISO/IEC. It takes a more rigorous stance for instance, an h3 element can't occur after an h1 element unless there is an intervening h2 element. Roger Price and David Abrahamson have written a user's guide to ISO HTML.

Other Public Drafts

We would like to hear from you via email. Please send your comments to: www-html@w3.org (archive). Don't forget to include **XHTML** in the subject line.

HTML Working Group Roadmap

This describes the timeline for deliverables of the HTML working group. It used to be a W3C NOTE but has now been moved to the MarkUp area for easier maintenance.

Modularization of XHTML in XML Schema

The purpose of this document is to describe a modularization framework for languages within the XHTML Namespace using XML Schema. This document provides a complete set of XML Schema modules for XHTML. In addition to the schema modules themselves, the framework presented here describes a means of further extending and modifying XHTML.

An XHTML + MathML + SVG Profile

An XHTML+MathML+SVG profile is a profile that combines XHTML 1.1, MathML 2.0 and SVG 1.1 together. This profile enables mixing XHTML, MathML and SVG in the same document using XML namespaces mechanism, while allowing validation of such a mixed-namespace document.

This specification is a joint work with the SVG Working Group, with the help from the Math WG.

XHTML 2.0

XHTML 2.0 is a markup language intended for rich, portable web-based applications. While the ancestry of XHTML 2.0 comes from HTML 4, XHTML 1.0, and XHTML 1.1, it is *not* intended to be backward compatible with its earlier versions. Application developers familiar with its earlier ancestors will be comfortable working with XHTML 2.0.

XHTML 2 is a member of the XHTML Family of markup languages. It is an XHTML Host Language as defined in Modularization of XHTML. As such, it is made up of a set of XHTML Modules that together describe the elements and attributes of the language, and their content model. XHTML 2.0 updates many of the modules defined in Modularization of XHTML, and includes the updated versions of all those modules and their semantics. XHTML 2.0 also uses modules from Ruby, XML Events, and XForms.

XFrames

XFrames is an XML application for composing documents together, replacing

HTML Frames. XFrames is *not* a part of XHTML per se, that allows similar functionality to HTML Frames, with fewer usability problems, principally by making the content of the frameset visible in its URI.

XHTML 1.0 in XML Schema

This document describes *informative* XML Schemas for XHTML 1.0. These Schemas are still work in progress, and this document *does not* change the normative definition of XHTML 1.0.

HLink

The HLink module defined in this specification provides XHTML Family Members with the ability to specify which attributes of elements represent Hyperlinks, and how those hyperlinks should be traversed, and extends XLink use to a wider class of languages than those restricted to the syntactic style allowed by XLink.

XHTML-Print

XHTML-Print is member of the family of XHTML Languages defined by the *Modularization of XHTML*. It is designed to be appropriate for printing from mobile devices to low-cost printers that might not have a full-page buffer and that generally print from top-to-bottom and left-to-right with the paper in a portrait orientation. XHTML-Print is also targeted at printing in environments where it is not feasible or desirable to install a printer-specific driver and where some variability in the formatting of the output is acceptable.

Useful information for HTML/XHTML authors

Tutorials

- *Getting started with HTML* by Dave Raggett is a short introduction to writing HTML, including tutorials on advanced features.
- *Adding a touch of style* by Dave Raggett is a short guide to styling your Web pages.
- *XHTML Modules and Markup Languages - How to create XHTML Family modules and markup languages for fun and profit* by Shane McCarron explains how to create XHTML Family modules and markup languages, based on Modularization of XHTML.

Slides on XHTML

You may also be interested in the following slides on XHTML:

- XHTML: The Extensible Hypertext Markup Language by Dave Raggett, at W3C LA event in Stockholm, 24 March 1999.
- W3C HTML Activity by Dave Raggett, as part of WWW8 W3C Track, 12 May 1999
- W3C Work on XHTML by Dave Raggett, at XML '99, 6 December 1999. The presentation describes the work being done by W3C on XHTML.
- The XHTML Family (in Japanese) by Masayasu Ishikawa, at SFC Open Research Forum 2001, 21 September 2001.
- XForms, XHTML and Device Independence by Steven Pemberton, at W3C.DE-Arbeitstreffen: Cross Media Publishing, 11 April 2002.
- XHTML Family by Masayasu Ishikawa, as part of WWW2002 W3C Track, 9 May 2002. Slides are available in XHTML or HTML (XHTML version needs XHTML+MathML+SVG+Ruby support).
- XHTML 2.0 and XForms by Steven Pemberton, as part of WWW2003 W3C Track, 21 May 2003.
- W3C's Horizontal Activities Usage: XHTML Family Case Study by Steven Pemberton, WWW2003 W3C Track, 23 May 2003.
- XHTML and XForms by Steven Pemberton, at Zomersessie van NGI Limburg: XHTML2 en XForms, state of the art en stage-ervaringen bij het W3C, 3 July 2003.

Guidelines for authoring

Here are some rough guidelines for HTML authors. If you use these, you are more likely to end up with pages that are easy to maintain, look acceptable to users regardless of the browser they are using, and can be accessed by the many Web users with disabilities. Meanwhile W3C have produced some more formal guidelines for authors. Have a look at the detailed Web Content Accessibility Guidelines 1.0.

1. **A question of style sheets.** For most people the look of a document - the color, the font, the margins - are as important as the textual content of the document itself. But make no mistake! HTML is not designed to be used to control these aspects of document layout. What you should do is to use HTML to mark up headings, paragraphs, lists, hypertext links, and other structural parts of your document, and then add a style sheet to specify layout separately, just as you might do in a conventional Desk Top Publishing Package. That way, not only is there a better chance of all browsers displaying your document properly, but also, if you want to change such things as the font or color, it's really simple to do so. See the Touch of style.

2. **FONT tag considered harmful!** Many filters from word-processing packages, and also some HTML authoring tools, generate HTML code which is completely contrary to the design goals of the language. What they do is to look at a document almost purely from the point of view of layout, and then mimic that layout in HTML by doing tricks with FONT, BR and ` ` (non-breaking spaces). HTML documents are supposed to be structured around items such as paragraphs, headings and lists. Yet some of these documents barely have a paragraph tag in sight!

   The problem comes when the content of pages needs to be updated, or given a new layout, or re-cast in XML (which is now to be the new mark-up language). With proper use of HTML, such operations are not difficult, but with a muddle of non-structural tags it's quite a different matter; maintenance tasks become impractical. To correct pages suffering from injudicious use of FONT, try the HTML Tidy program, which will do its best to put things right and generate better and more manageable HTML.

3. **Make your pages readable by those with disabilities.** The Web is a tremendously useful tool for the visually impaired or blind user, but bear in mind that these users rely on speech synthesizers or Braille readers to render the text. Sloppy mark-up, or mark-up which doesn't have the layout defined in a separate style sheet, is hard for such software to deal with. Wherever possible, use a style sheet for the presentational aspects of your pages, using HTML purely for structural mark-up.

   Also, remember to include descriptions with each image, and try to avoid server-side image maps. For tables, you should include a summary of the table's structure, and remember to associate table data with relevant headers. This will give non-visual browsers a chance to help orient people as they move from one cell to the next. For forms, remember to include labels for form fields.

Do look at the accessibility guidelines for a more detailed account of how to make your Web pages really accessible.

W3C Markup Validation Service

To further promote the reliability and fidelity of communications on the Web, W3C has introduced the W3C Markup Validation Service at `http://validator.w3.org/`.

Content providers can use this service to validate their Web pages against the HTML and XHTML Recommendations, thereby ensuring the maximum possible audience for their Web pages. It also supports XHTML Family document types

such as XHTML+MathML and XHTML+MathML+SVG, and also other markup vocabularies such as SVG.

Software developers who write HTML and XHTML editing tools can ensure interoperability with other Web software by verifying that the output of their tool complies with the W3C Recommendations for HTML and XHTML.

HTML Tidy

HTML Tidy is a stand-alone tool for checking and pretty-printing HTML that is in many cases able to fix up mark-up errors, and also offers a means to convert existing HTML content into well-formed XML, for delivery as XHTML. HTML Tidy was originally written by Dave Raggett, and it is now maintained as an open source project at SourceForge by a group of volunteers.

There is an archived public mailing list html-tidy@w3.org. Please send bug reports / suggestions on HTML Tidy to this mailing list.

Discussion Forums

Changes to HTML necessitate obtaining a consensus from a broad range of organizations. If you have a great idea, it will take time to convince others! Here are some of the places where discussion on HTML takes place:

comp.infosystems.www.authoring.html
: A USENET newsgroup where HTML authoring issues are discussed. "How To" questions should be addressed here. Note that many issues related to forms and CGI, image maps, transparent gifs, etc. are covered in the WWW FAQ.

www-html@w3.org
: A technical discussion list. If you have a proposal for a change to HTML/XHTML, you might start a discussion here to see what other developers think of it.
  - how to subscribe
  - archives from 1994 to present
  - (We're working on moving the old archives to W3C. Stay tuned!)

www-html-editor@w3.org
: This is a list to report errors / send review comments on HTML/XHTML specifications. *This is NOT a discussion list.* Anyone may send comments without subscription, although you'll be requested to give explicit approval to include your message in our publicly-readable mailing list archive at your first post. To subscribe, send subscription request to www-html-editor-request@w3.org. For more information, see how to subscribe.

W3C HTML Working Group (members only)

The HTML WG is open to W3C Members and invited experts. The Group's mission is to develop the next generation of HTML as a suite of XML tag sets with a clean migration path from HTML 4. Some of the expected benefits include: reduced authoring costs, an improved match to database & workflow applications, a modular solution to the increasingly disparate capabilities of browsers, and the ability to cleanly integrate HTML with other XML applications. The Group is chaired by Steven Pemberton.

Current Working Group participants include:

- America Online, Inc. (AOL)
- CWI
- HP
- Matsushita Electric Industrial Co., Ltd. (MEI)
- Microsoft Corporation
- Opera Software
- Oracle Corporation
- SAP AG
- Sun Microsystems, Ltd.

w3c-translators@w3.org

This is a mailing list for people working on translations of W3C specifications such as the HTML/XHTML Recommendations. To subscribe, send an email to w3c-translators-request@w3.org with the word "subscribe" in the subject line; (include the word "unsubscribe" if you want to unsubscribe.) The archive for the list is accessible online.

IETF MHTML WG (closed)

Developed RFC 2557 - "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML). J. Palme et al. March 1989.

IETF HTML Working Group (closed)

The HTML working group of the IETF, closed in 1996.

Web Conferences

The next international conference dedicated to the Web is WWW2004, to be held in New York city, USA, in 2004. The last was WWW2003, which was held in Budapest, Hungary, 20-24 May 2003.

Related W3C Work

XML

XML is the universal format for structured documents and data on the Web. It allows you to define your own mark-up formats when HTML is not a good fit. XML is being used increasingly for data; for instance, W3C's

metadata format RDF.

Style Sheets

W3C's Cascading Style Sheets language (CSS) provides a simple means to style HTML pages, allowing you to control visual and aural characteristics; for instance, fonts, margins, line-spacing, borders, colors, layers and more. W3C is also working on a new style sheet language written in XML called XSL, which provides a means to transform XML documents into HTML.

Document Object Model

Provides ways for scripts to manipulate HTML using a set of methods and data types defined independently of particular programming languages or computer platforms. It forms the basis for dynamic effects in Web pages, but can also be exploited in HTML editors and other tools by extensions for manipulating HTML content.

Internationalization

HTML 4 provides a number of features for use with a wide variety of languages and writing systems. For instance, mixed language text, and right-to-left and mixed direction text. HTML 4 is formally based upon Unicode, but allows you to store and transmit documents in a variety of character encodings. Further work is envisaged for handling vertical text and phonetic annotations for Kanji (Ruby).

Access for People with Disabilities

HTML 4 includes many features for improved access by people with disabilities. W3C's Web Accessibility Initiative is working on providing effective guidelines for making your pages accessible to all, not just those using graphical browsers.

XForms

Forms are a very widely used feature in web pages. W3C is working on the design of the next generation of web forms with a view to separating the presentation, data and logic, as a means to allowing the same forms to be used with widely differing presentations.

Mathematics

Work on representing mathematics on the Web has focused on ways to handle the presentation of mathematical expressions and also the intended meaning. The MathML language is an application of XML, which, while not suited to hand-editing, is easy to process by machine.

Contacts

- Masayasu ISHIKAWA is the HTML Activity Lead and the Team Contact for the HTML Working Group

news | Recommendations | public drafts | test suites | tutorials | slides | guidelines | validation | articles | translations | charter | working group | roadmap | XForms | forums | HTML Tidy | related work | HTML 4.0/3.2/2.0 | historical

---

This page was last modified on: $Date: 2004/05/06 05:58:05 $

J2EE

JavaServer Faces

Downloads

Reference
- API Specifications
- Documentation
- FAQs
- Code Samples & Apps

Community
- Forums
- Bug Database

JavaServer Faces technology simplifies building user interfaces for JavaServer applications. Developers of various skill levels can quickly build web applications by: assembling reusable UI components in a page; connecting these components to an application data source; and wiring client-generated events to server-side event handlers. » Read More

What's New

April 2, 2004
JavaServer Faces 1.0 SCSL release The source code is now available for JavaServer Faces 1.0 release.

March 22, 2004
J2EE 1.4 SDK now includes JavaServer Faces support The J2EE 1.4 SDK contains the Sun Java System Application Server Platform Edition 8 FCS featuring support for JavaServer Faces technology and is free for development, deployment, and redistribution. Get the SDK now at http://java.sun.com/j2ee/1.4/download-sdk.html.

March 22, 2004
JavaServer Faces in The J2EE 1.4 Tutorial JavaServer Faces technology is now documented in The J2EE 1.4 Tutorial.

March 3, 2004
JavaServer Faces Now Shipping After receiving approval for JSR-127 from the Executive Committee of the Java Community Process, JavaServer Faces is now available for download. Under the terms of the license, the JavaServer Faces Reference Implementation is free to redistribute in binary form. This means you can include the JavaServer Faces Reference Implementation with your software distribution.

Community

JavaServer Faces Forum Drop in to the JavaServer Faces forum to discuss how to build web applications using JavaServerFaces technology.

Java Specification Request 127 This specification request defines an architecture and APIs that simplify the creation and maintenance of Java Server application GUIs.

Events
2004 JavaOne Conference. San Francisco , CA Be there! Join the thousands of developers worldwide who come to the JavaOne conference each year in San Francisco to immerse themselves in Java technology, the latest innovations, the community, and the learning opportunities. » Read More

Subscribe to Newsletters. Members of Sun Developer Network can sign up to receive these (and other) newsletters. Not yet a member? **Join us!**
Enterprise Java Technologies Newsletter

Related Links

Popular Downloads
- JavaServer Faces Technology
- Sun Java Studio Creator Early Access release
- J2EE 1.4 SDK
- Java Web Services Developer Pack 1.3
- Sun Java Studio Standard IDE
- Sun Java System Application Server

Technical Topics
- Web Services

Products and Technologies
- Java Servlet
- JavaServer Pages

Sun Resources
- Developer Technical Support
- Professional Training
- Professional Certification
- java.net

Learn about new enterprise Java technologies, products, tools, and resources for developers.

Enterprise Java Technologies Tech Tips

Get expert tips, sample code solutions, and techniques for developing in the Java 2 Platform, Enterprise Edition (J2EE).

» Read More

The Extensible Stylesheet Language Family (XSL)

XSL is a family of recommendations for defining XML document transformation and presentation. It consists of three parts:

XSL Transformations (XSLT)
: a language for transforming XML

the XML Path Language (XPath)
: an expression language used by XSLT to access or refer to parts of an XML document. (XPath is also used by the XML Linking specification)

XSL Formatting Objects (XSL-FO)
: an XML vocabulary for specifying formatting semantics

An XSLT stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses a formatting vocabulary, such as (X)HTML or XSL-FO. For a more detailed explanation of how XSL works, see the What Is XSL page.

For background information on style sheets, see the Web style sheets resource page. XSL is developed by the W3C XSL Working Group (members only) whose charter is to develop the next version of XSL. XSL is part of W3C's XML Activity, whose work is described in the XML Activity Statement.

Specifications

- XSLT 1.0
- XPath 1.0
- XQuery 1.0 and XPath 2.0 Data Model (WD)
- XPath 2.0 requirements
- XSLT 2.0 requirements
- XSLT 2.0
- XPath 2.0
- XSL 1.0 (aka XSL-FO)
- XSL 1.1 (WD)

Mailing Lists

- XSL-List@lists.mulberrytech.com , main list for discussion about XSL
- public-qt-comments@w3.org : comments on XSLT/XPath 2.0
- www-xpath-comments@w3.org : comments/discussion on XPath 1.0
- xsl-editors@w3.org : comments/discussion on XSLT 1.0
- www-xsl-fo@w3.org W3C list about Formatting Objects
- XSL-FO@yahoogroups.com : a YahooGoups list on FOs.

Software

- **XSLT: too many to list here. Check dmoz.org .**
- XPath 2.0 Grammar Test Applet
- XSL Formatter (Win/Linux, free evaluation versions)
- Scriptura: WYSIWYG XSL-FO stylesheet editor (Java, free evaluation)
- XSLFast (Java, free evaluation version)
- XEP (Java, free evaluation version)
- FOP (Java, open source)
- PassiveTeX (TeX, open source)
- Apoc XSL-FO , for integration with .NET [free evaluation version]
- Unicorn FOs (TeX, free Windows binaries)
- REXP early implementation based on FOP
- X-smiles (Java, open source)
- Chameleon/XML , an XML rendering system using FO
- Novosoft RTF2FO : RTF to FO converter (Java, Free evaluation version)
- jfor : FO to RTF converter (Java, Open Source)
- WH2FO : WordHTML-to-FO (Java, Open Source)
- html2fo (C, Open Source)
- FOA : XSL Authoring Tool (Java, Open Source)
- jFO Java API for FOs (evaluation version available)
- Adobe Document Server : document production framework (commercial, Win/Solaris)
- Infoprint XML Extender from IBM: document printing application (commercial, z/OS)

Translations

- XPath 1.0 (Spanish)
- XPath 1.0 (German)
- XPath1.0 (Russian)
- XPath1.0 (Russian) (2)
- XSLT1.0 (Japanese)
- XPath 1.0 (Japanese)
- XPath 1.0 (Japanese) (2)
- XPath 1.0 (French)

- XSLT 1.0 (French)
- XSLT 1.0 (Russian)
- XSL 1.0 (Russian)
- XSLT 1.0 (German)

Tutorials

- XSL-FO tutorial by RenderX
- How to Develop Stylesheets for XML to XSL-FO Transformation by Antenna House
- XSLT & XPath tutorial from TopXML
- XSL School from w3schools.
- Introduction to XSL by Miloslav Nic
- Tutorial from iX magazine (German , English)

Reference

- XSLT and XPATH: A Guide to XML Transformations by John Robert Gardner and Zarella L. Rendon
- An introduction to XSL Formatting Objects , a book by Dave Pawson, available on-line.
- Chapter 15 of the XML Bible by Elliotte Rusty Harold is about Formatting Objects is available online.
- Miloslav Nic has published a complete FO reference
- The XSL FAQ . A very complete repository of XSL material.
- VBXML reference on XSLT and XPath .
- XSLT Programmer's Reference (2nd edition) by Michael Kay
- XSLT and XPath On The Edge, Unlimited Edition by Jeni Tennison
- XSLT Quickly by Bob DuCharme
- XSLT by Doug Tidwell
- Practical Transformations Using XSLT and XPath , Practical Formatting Using XSLFO and Definitive XSLT and XPath from Crane Softwrights Ltd.
- Chapter 17 of the XML Bible is dedicated to XSLT

Articles

- What Is XSL-FO and When Should I Use It? by Stephen Deach
- What's New in XSLT 2.0 by Evan Lenz
- What's New in XPath 2.0 by Evan Lenz
- Using XSL Formatting Objects by J. David Eisenberg, from xml.com.
- XSL style sheets: push or pull? by Kevin Williams
- Expand XSL with extensions by Jared Jackson
- Tip: Outputting HTML from an XSL style sheet by Nicholas Chase

Links

- Stylesheet Central , a collection of over 100 examples of XML and XSLT stylesheets, arranged into categories
- EXSLT , a community initiative to provide extensions to XSLT
- The XSLT language code library by TopXML
- A few useful XSL tools and documents
- XSL Resources at Oasis
- jeremie.com's XSL pages
- xslt.com , resource list for XSLT
- XML-Web , XSLT site in German
- XSL Resources Plattform (in German)

News

2004-05-11 : Oxygen XML Editor adds an XSLT Debugger

The latest release of Oxygen XML Editor improves the XSLT editing and transformation support and adds a full featured XSLT Debugger. Now complex transformations can be executed involving multiple stylesheets and macros can be used when configuring scenarios for a greater reuse. When editing an XSLT a schema/DTD for the transformation result can be set and the content completion will offer the elements and attributes from that target schema where appropriate. The XSLT Debugger supports both Xalan and Saxon as XSLT engines and provides full debugger options: step into, step over, run to cursor, run to end, pause, stop, breakpoints, call stack, trace history, XPath expression watch, variables view, etc. Please find more details about oXygen XML editor in general at www.oxygenxml.com and about the XSLT Debugger features at www.oxygenxml.com/features/XSLT-Debugger.html .

2004-04-30 : RenderX Announces New Version - XEP.NET Released

RenderX has released XEP.NET , a Visual J#/.NET port of XEP, RenderX's XSL Formatting engine. It consists of 100% manageable code, no Java or native libraries are required to run it. Its functionality and XSL FO support level are identical to RenderX's Java version. The XEP.NET core is wrapped in an API that exposes standard .NET interfaces for XML processing: XmlReader and XmlWriter. This public API is a .NET class library component that can be used from .NET programming language, such as Visual C#.NET, Visual Basic.NET, and Visual J#.NET. Additionally, the software includes a .NET class library for MSXML integration, with C# source codes included. This integration component allows use of MSXML SAX parser and transformation APIs in addition to .NET system interfaces. A trial version is available for download from the RenderX developer's site (http://shop.xattic.com) and trial developer software can be requested from sales@renderx.com.

2004-03-24 : Antenna House XSL Formatter V3.1

Antenna House is proud to release this milestone in the development of XSL Formatter, Version 3.1 as of March 24, 2004.~XSL Formatter V3.1 offers all the features (GUI, support for over 50 languages, PDF generation), W3C compliance, extensions and formatting capabilities of our previous Version 2.5 plus the significantly enhanced performance, unlimited document size and SVG support of V3.0. In addition V3.1 offers vector support for EMF and WMF, CMYK support, new output capabilities and numerous other enhancements. With V3.1 a customer can now have all the advantages of V3 and V2.5 in one. See product page for and pointer to free evaluation download.

2004-02-11 : RTF2FO 3.3.3

Novosoft Inc., is pleased to announce the new 3.3.3 version of RTF2FO converter. RTF2FO 3.3.3 enhances and streamlines process of converting RTF documents to XML ones in line with the W3C specifications for XSL FO formatting semantics. The new version is available for evaluation at http://www.rtf2fo.com/download.html

2003-11-14 : Scriptura 2.1

Scriptura 2.1. Inventive Designers releases Scriptura 2.1, the cross-platform document design and generation solution based on XSLT and XSL-FO. It has a WYSIWYG design tool and an engine. The XSL-FO formatter used in the engine is no longer based on Apache FOP, but is written from scratch by Inventive Designers. The new features in this release are: support for bulleted and numbered lists, 'break-before' and 'break-after' properties, extended bar code options and improved number and currency formatting. A free trial version is available for download.

2003-11-12 : Last Call working drafts

New working drafts of XSLT 2.0, XPath 2.0 and associated XML Query specifications published, most in Last Call, see the TR page

2003-10-15 : Xinc Beta

Xinc Beta release. Lunasil Ltd is pleased to announce the Beta release of Xinc, an XSL-FO processor. Xinc was designed to be fast, multithreaded and memory efficient. A Swing based XSL-FO viewer allows you to view and print XSL-FO files as well as generate PDF files with the click of a button. Xinc can be used as a server component via its Java API. Xinc can also be used in a Microsoft server environment by using its COM interface. New features include hyphenation, basic-link, PDF output, memory/speed optimizations and a simple COM interface.

2003-10-01 : XSL Formatter 3.0

XSL Formatter V3.0 Release. Antenna House , Inc is pleased to inform you that our XSL-FO processor [XSL Formatter V3.0] is now available. XSL Formatter is a software to format XML

documents for production-quality printing and output to PDF. Antenna House has been providing version V2 of the same product since January, 2002 in the global market, and XSL Formatter was rated as one of the best quality product at the XML2002, XML 2003 conferences held in Europe.

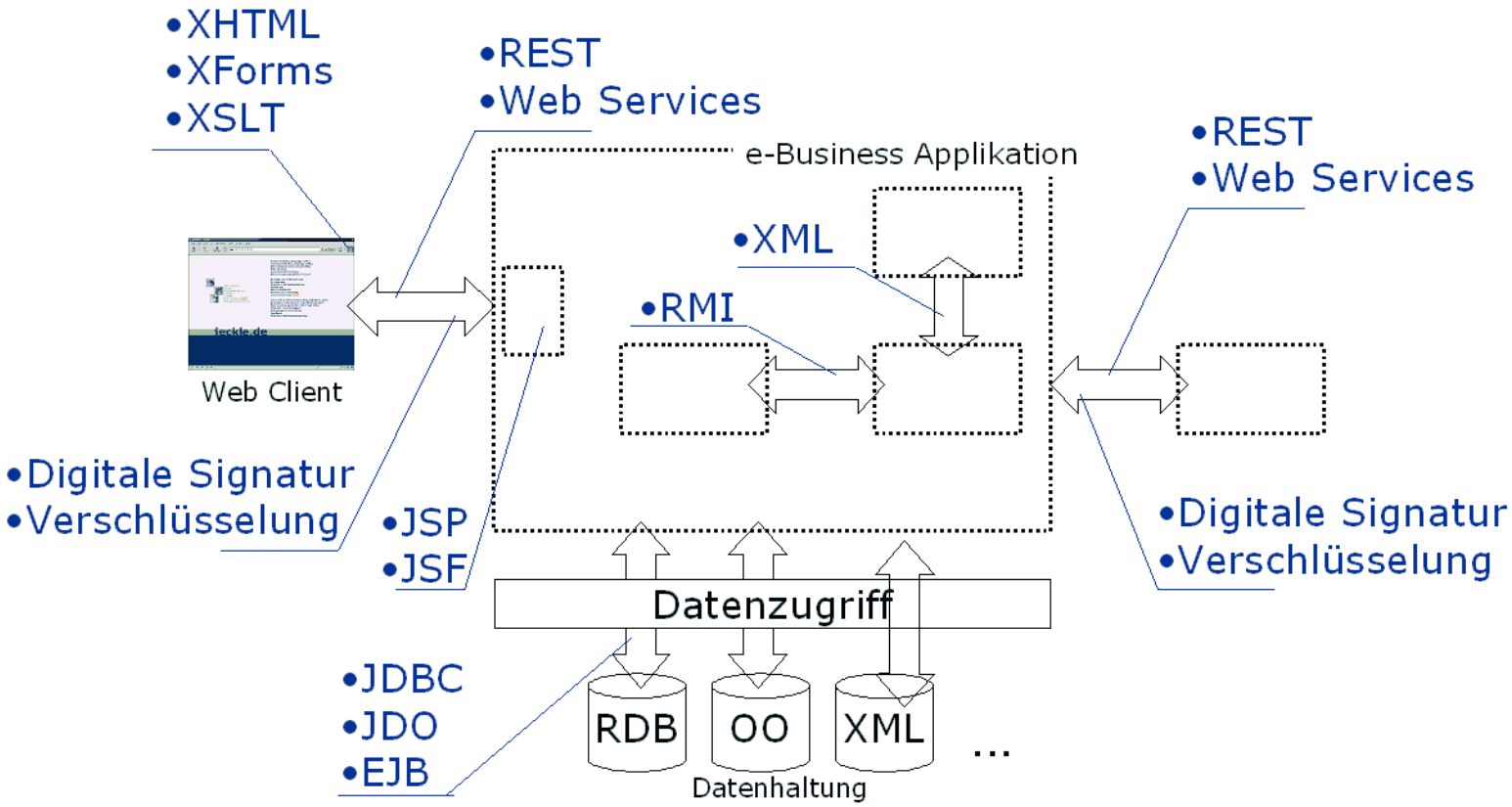
2003-08-22 : New working draftw

New working drafts of XSLT 2.0, XPath 2.0 and associated XML Query specifications published, some in Last Call, see the TR page

2003-08-07 : XSL Formatter 3.0 Beta

XSL Formatter V3.0 Released. Antenna House, Inc is pleased to inform you that the Beta version of our XSL-FO processor [XSL Formatter V3.0] is now available. Building on over 4 years of experience developing XSL-FO software Antenna House has completely written from scratch an entirely new Formatter that offers significant enhancements and provides a solid foundation on which to continue to move forward.

Liam Quin (liam@w3.org), Team Contact for the XSL Working Group
$Id: Overview.html,v 1.352 2004/05/11 09:52:39 liam Exp $.
This page was generated using XSLT. The XML source is also available for viewing on an XSLT-enabled browser.


•XHTML
•XForms
•XSLT
•REST
•Web Services
e-Business Applikation
•REST
•Web Services
•XML
•RMI
Web Client
•Digitale Signatur
•Verschlüsselung
•JSP
•JSF
•Digitale Signatur
•Verschlüsselung
Datenzugriff
•JDBC
•JDO
•EJB
RDB
OO
XML
...
Datenhaltung

REC-xml-19980210

Extensible Markup Language (XML) 1.0

W3C Recommendation 10-February-1998

This version:
http://www.w3.org/TR/1998/REC-xml-19980210
http://www.w3.org/TR/1998/REC-xml-19980210.xml
http://www.w3.org/TR/1998/REC-xml-19980210.html
http://www.w3.org/TR/1998/REC-xml-19980210.pdf
http://www.w3.org/TR/1998/REC-xml-19980210.ps
Latest version:
http://www.w3.org/TR/REC-xml
Previous version:
http://www.w3.org/TR/PR-xml-971208
Editors:
Tim Bray (Textuality and Netscape) <tbray@textuality.com>
Jean Paoli (Microsoft) <jeanpa@microsoft.com>
C. M. Sperberg-McQueen (University of Illinois at Chicago) <cmsmcq@uic.edu>

Abstract

The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

Status of this document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.
This document specifies a syntax created by subsetting an existing, widely used international text processing standard (Standard Generalized Markup Language, ISO 8879:1986(E) as amended and corrected) for use on the World Wide Web. It is a product of the W3C XML Activity, details of which can be found at http://www.w3.org/XML. A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.
This specification uses the term URI, which is defined by [Berners-Lee et al.], a work in progress expected to update [IETF RFC1738] and [IETF RFC1808].
The list of known errors in this specification is available at http://www.w3.org/XML/xml-19980210-errata.
Please report errors in this document to xml-editor@w3.org.

Extensible Markup Language (XML) 1.0

Table of Contents

Appendices

---

1. Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.
XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.
A software module called an **XML processor** is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the **application**. This specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C. The membership of the XML Working Group is given in an appendix. Dan Connolly served as the WG's contact with the W3C.
The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

This specification, together with associated standards (Unicode and ISO/IEC 10646 for characters, Internet RFC 1766 for language identification tags, ISO 639 for language name codes, and ISO 3166 for country name codes), provides all the information necessary to understand XML Version 1.0 and construct computer programs to process it. This version of the XML specification may be distributed freely, as long as all text and legal notices remain intact.

1.2 Terminology

The terminology used to describe XML documents is defined in the body of this specification. The terms defined in the following list are used in building those definitions and in describing the actions of an XML processor:

may
Conforming documents and XML processors are permitted to but need not behave as described.
must
Conforming documents and XML processors are required to behave as described; otherwise they are in error.

error
: A violation of the rules of this specification; results are undefined. Conforming software may detect and report an error and may recover from it.

fatal error
: An error which a conforming XML processor must detect and report to the application. After encountering a fatal error, the processor may continue processing the data to search for further errors and may report such errors to the application. In order to support correction of errors, the processor may make unprocessed data from the document (with intermingled character data and markup) available to the application. Once a fatal error is detected, however, the processor must not continue normal processing (i.e., it must not continue to pass character data and information about the document's logical structure to the application in the normal way).

at user option
: Conforming software may or must (depending on the modal verb in the sentence) behave as described; if it does, it must provide users a means to enable or disable the behavior described.

validity constraint
: A rule which applies to all valid XML documents. Violations of validity constraints are errors; they must, at user option, be reported by validating XML processors.

well-formedness constraint
: A rule which applies to all well-formed XML documents. Violations of well-formedness constraints are fatal errors.

match
: (Of strings or names:) Two strings or names being compared must be identical. Characters with multiple possible representations in ISO/IEC 10646 (e.g. characters with both precomposed and base+diacritic forms) match only if they have the same representation in both strings. At user option, processors may normalize such characters to some canonical form. No case folding is performed. (Of strings and rules in the grammar:) A string matches a grammatical production if it belongs to the language generated by that production. (Of content and content models:) An element matches its declaration when it conforms in the fashion described in the constraint "Element Valid".

for compatibility
: A feature of XML included solely to ensure that XML remains compatible with SGML.

for interoperability
: A non-binding recommendation included to increase the chances that XML documents can be processed by the existing installed base of SGML processors which predate the WebSGML Adaptations Annex to ISO 8879.

2. Documents

A data object is an **XML document** if it is well-formed, as defined in this specification. A well-formed XML document may in addition be valid if it meets certain further constraints.

Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The logical and physical structures must nest properly, as described in "4.3.2 Well-Formed Parsed Entities".

2.1 Well-Formed XML Documents

A textual object is a well-formed XML document if:

1. Taken as a whole, it matches the production labeled `document`.
2. It meets all the well-formedness constraints given in this specification.
3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed.

Document

```
[1] document :: prolog element Misc*
             =
```

Matching the `document` production implies that:

1. It contains one or more elements.
2. There is exactly one element, called the **root**, or document element, no part of which appears in the content of any other element. For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

As a consequence of this, for each non-root element `C` in the document, there is one other element `P` in the document such that `C` is in the content of `P`, but is not in the content of any other element that is in the content of `P`. `P` is referred to as the **parent** of `C`, and `C` as a **child** of `P`.

2.2 Characters

A parsed entity contains **text**, a sequence of characters, which may represent markup or character data. A **character** is an atomic unit of text as specified by ISO/IEC 10646 [ISO/IEC 10646]. Legal characters are tab, carriage return, line feed, and the legal graphic characters of Unicode and ISO/IEC 10646. The use of "compatibility characters", as defined in section 6.8 of [Unicode], is discouraged.

Character Range

```
[2] Char :: #x9 | #xA | #xD | [#x20-          / any Unicode character,
         = #xD7FF] | [#xE000-#xFFFD]         * excluding the surrogate
           | [#x10000-#x10FFFF]                blocks, FFFE, and FFFF. */
```

The mechanism for encoding character code points into bit patterns may vary from entity to entity. All XML processors must accept the UTF-8 and UTF-16 encodings of 10646; the mechanisms for signaling which of the two is in use, or for bringing other encodings into play, are discussed later, in "4.3.3 Character Encoding in Entities".

2.3 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.
S (white space) consists of one or more space (#x20) characters, carriage returns, line feeds, or tabs.

White Space

```
[3] S :: (#x20 | #x9 | #xD | #xA)+
      =
```

Characters are classified for convenience as letters, digits, or other characters. Letters consist of an alphabetic or syllabic base character possibly followed by one or more combining characters, or of an ideographic character. Full definitions of the specific characters in each class are given in "B. Character Classes".

A **Name** is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters. Names beginning with the string "`xml`", or any string which would match `(('X'|'x') ('M'|'m') ('L'|'l'))`, are reserved for standardization in this or future versions of this specification.

Note: The colon character within XML names is reserved for experimentation with name spaces. Its meaning is expected to be standardized at some future point, at which point those documents using the colon for experimental purposes may need to be updated. (There is no guarantee that any name-space mechanism adopted for XML will in fact use the colon as a name-space delimiter.) In practice, this means that authors should not use the colon in XML names except as part of name-space experiments, but that XML processors should accept the colon as a name character.

An `Nmtoken` (name token) is any mixture of name characters.

Names and Tokens

```
[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar
                 | Extender
[5]     Name ::= (Letter | '_' | ':') (NameChar)*
[6]    Names ::= Name (S Name)*
[7]  Nmtoken ::= (NameChar)+
[8] Nmtokens ::= Nmtoken (S Nmtoken)*
```

Literal data is any quoted string not containing the quotation mark used as a delimiter for that string. Literals are used for specifying the content of internal entities (EntityValue), the values of attributes (AttValue), and external identifiers (SystemLiteral). Note that a SystemLiteral can be parsed without scanning for markup.

Literals

```
 [9]   EntityValue ::= '"' ([^%&"] | PEReference | Reference)* '"'
                     |  "'" ([^%&'] | PEReference | Reference)* "'"
[10]      AttValue ::= '"' ([^<&"] | Reference)* '"'
                     |  "'" ([^<&'] | Reference)* "'"
[11] SystemLiteral ::= ('"' [^"]* '"') | ("'" [^']* "'")
[12]  PubidLiteral ::= '"' PubidChar* '"' | "'" (PubidChar - "'")* "'"
[13]     PubidChar ::= #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]
```

2.4 Character Data and Markup

Text consists of intermingled character data and markup. **Markup** takes the form of start-tags, end-tags, empty-element tags, entity references, character references, comments, CDATA section delimiters, document type declarations, and processing instructions.

All text that is not markup constitutes the **character data** of the document.

The ampersand character (&) and the left angle bracket (<) may appear in their literal form *only* when used as markup delimiters, or within a comment, a processing instruction, or a CDATA section. They are also legal within the literal entity value of an internal entity declaration; see "4.3.2 Well-Formed Parsed Entities". If they are needed elsewhere, they must be escaped using either numeric character references or the strings "`&`" and "`<`" respectively. The right angle bracket (>) may be represented using the string "`>`", and must, for compatibility, be escaped using "`>`" or a character reference when it appears in the string "`]]>`" in content, when that string is not marking the end of a CDATA section.

In the content of elements, character data is any string of characters which does not contain the start-delimiter of any markup. In a CDATA section, character data is any string of characters not including the CDATA-section-close delimiter, "`]]>`".

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') may be represented as "`&apos;`", and the double-quote character (") as "`&quot;`".

Character Data

```
[14] CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
```

2.5 Comments

Comments may appear anywhere in a document outside other markup; in addition, they may appear within the document type declaration at places allowed by the grammar. They are not part of the document's character data; an XML processor may, but need not, make it possible for an application to retrieve the text of comments. For compatibility, the string "`--`" (double-hyphen) must not occur within comments.

Comments

```
[15] Comment :: '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
             =
```

An example of a comment:

```
<!-- declarations for <head> & <body> -->
```

2.6 Processing Instructions

Processing instructions (PIs) allow documents to contain instructions for applications.

Processing Instructions

```
[16]       PI :: '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
              =
[17] PITarget :: Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
              =
```

PIs are not part of the document's character data, but must be passed through to the application. The PI begins with a target (`PITarget`) used to identify the application to which the instruction is directed. The target names "`XML`", "`xml`", and so on are reserved for standardization in this or future versions of this specification. The XML Notation mechanism may be used for formal declaration of PI targets.

2.7 CDATA Sections

CDATA sections may occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup. CDATA sections begin with the string "`<![CDATA[`" and end with the string "`]]>`":

CDATA Sections

```
[18]  CDSect :: CDStart CData CDEnd
             =
[19] CDStart :: '<![CDATA['
             =
[20]   CData :: (Char* - (Char* ']]>'
             = Char*))
[21]   CDEnd :: ']]>'
             =
```

Within a CDATA section, only the `CDEnd` string is recognized as markup, so that left angle brackets and ampersands may occur in their literal form; they need not (and cannot) be escaped using "`<`" and "`&`". CDATA sections cannot nest.
An example of a CDATA section, in which "`<greeting>`" and "`</greeting>`" are recognized as character data, not markup:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

2.8 Prolog and Document Type Declaration

XML documents may, and should, begin with an **XML declaration** which specifies the version of XML being used. For example, the following is a complete XML document, well-formed but not valid:

```
<?xml version="1.0"?>
<greeting>Hello, world!</greeting>
```

and so is this:

```
<greeting>Hello, world!</greeting>
```

The version number "1.0" should be used to indicate conformance to this version of this specification; it is an error for a document to use the value "1.0" if it does not conform to this version of this specification. It is the intent of the XML working group to give later versions of this specification numbers other than "1.0", but this intent does not indicate a commitment to produce any future versions of XML, nor if any are produced, to use any particular numbering scheme. Since future versions are not ruled out, this construct is provided as a means to allow the possibility of automatic version recognition, should it become necessary. Processors may signal an error if they receive documents labeled with versions they do not support.

The function of the markup in an XML document is to describe its storage and logical structure and to associate attribute-value pairs with its logical structures. XML provides a mechanism, the document type declaration, to define constraints on the logical structure and to support the use of predefined storage units. An XML document is **valid** if it has an associated document type declaration and if the document complies with the constraints expressed in it.

The document type declaration must appear before the first element in the document.

Prolog

```
[22]      prolog :: XMLDecl? Misc* (doctypedecl Misc*)?
                  =
[23]     XMLDecl :: '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
                  =
[24] VersionInfo :: S 'version' Eq (' VersionNum ' | " VersionNum ")
                  =
[25]          Eq :: S? '=' S?
                  =
[26]  VersionNum :: ([a-zA-Z0-9_.:] | '-')+
                  =
[27]        Misc :: Comment | PI |  S
                  =
```

The XML **document type declaration** contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or **DTD**. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. The DTD for a document consists of both subsets taken together.

A **markup declaration** is an element type declaration, an attribute-list declaration, an entity declaration, or a notation declaration. These declarations may be contained in whole or in part within parameter entities, as described in the well-formedness and validity constraints below. For fuller information, see "4. Physical Structures".

Document Type Definition

```
[28] doctypedecl :: '<!DOCTYPE' S Name (S                  [ VC: Root Element
                 =  ExternalID)? S?                          Type ]
                    ('[' (markupdecl | PEReference
                    | S)* ']' S?)? '>'
[29]  markupdecl :: elementdecl | AttlistDecl              [ VC: Proper
                 =  | EntityDecl | NotationDecl              Declaration/PE
                    | PI | Comment                           Nesting ]
                                                           [ WFC: PEs in Internal
                                                             Subset ]
```

The markup declarations may be made up in whole or in part of the replacement text of parameter entities. The productions later in this specification for individual nonterminals (`elementdecl`, `AttlistDecl`, and so on) describe the declarations *after* all the parameter entities have been included.

Validity Constraint: Root Element Type

The `Name` in the document type declaration must match the element type of the root element.

Validity Constraint: Proper Declaration/PE Nesting

Parameter-entity replacement text must be properly nested with markup declarations. That is to say, if either the first character or the last character of a markup declaration (`markupdecl` above) is contained in the replacement text for a parameter-entity reference, both must be contained in the same replacement text.

Well-Formedness Constraint: PEs in Internal Subset

In the internal DTD subset, parameter-entity references can occur only where markup declarations can occur, not within markup declarations. (This does not apply to references that occur in external parameter entities or to the external subset.)

Like the internal subset, the external subset and any external parameter entities referred to in the DTD must consist of a series of complete markup declarations of the types allowed by the non-terminal symbol `markupdecl`, interspersed with white space or parameter-entity references. However, portions of the contents of the external subset or of external parameter entities may conditionally be ignored by using the conditional section construct; this is not allowed in the internal subset.

External Subset

```
[30]     extSubset :: TextDecl? extSubsetDecl
                    =
[31] extSubsetDecl :: ( markupdecl | conditionalSect | PEReference | S )*
                    =
```

The external subset and external parameter entities also differ from the internal subset in that in them, parameter-entity references are permitted *within* markup declarations, not only *between* markup declarations.

An example of an XML document with a document type declaration:

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

The system identifier "`hello.dtd`" gives the URI of a DTD for the document.

The declarations can also be given locally, as in this example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

If both the external and internal subsets are used, the internal subset is considered to occur before the external subset. This has the effect that entity and attribute-list declarations in the internal subset take precedence over those in the external subset.

2.9 Standalone Document Declaration

Markup declarations can affect the content of the document, as passed from an XML processor to an application; examples are attribute defaults and entity declarations. The standalone document declaration, which may appear as a component of the XML declaration, signals whether or not there are such declarations which appear external to the document entity.

Standalone Document Declaration

```
[32] SDDecl ::= S 'standalone' Eq                    [ VC: Standalone Document
                ((''' ('yes' | 'no') ''')                Declaration ]
                | ('"' ('yes' | 'no') '"'))
```

In a standalone document declaration, the value "`yes`" indicates that there are no markup declarations external to the

document entity (either in the DTD external subset, or in an external parameter entity referenced from the internal subset) which affect the information passed from the XML processor to the application. The value "no" indicates that there are or may be such external markup declarations. Note that the standalone document declaration only denotes the presence of external *declarations*; the presence, in a document, of references to external *entities*, when those entities are internally declared, does not change its standalone status.
If there are no external markup declarations, the standalone document declaration has no meaning. If there are external markup declarations but there is no standalone document declaration, the value "no" is assumed.
Any XML document for which `standalone="no"` holds can be converted algorithmically to a standalone document, which may be desirable for some network delivery applications.
Validity Constraint: Standalone Document Declaration
The standalone document declaration must have the value "no" if any external markup declarations contain declarations of:

- attributes with default values, if elements to which these attributes apply appear in the document without specifications of values for these attributes, or
- entities (other than `amp`, `lt`, `gt`, `apos`, `quot`), if references to those entities appear in the document, or
- attributes with values subject to normalization, where the attribute appears in the document with a value which will change as a result of normalization, or
- element types with element content, if white space occurs directly within any instance of those types.

An example XML declaration with a standalone document declaration:

```
<?xml version="1.0" standalone='yes'?>
```

2.10 White Space Handling

In editing XML documents, it is often convenient to use "white space" (spaces, tabs, and blank lines, denoted by the nonterminal S in this specification) to set apart the markup for greater readability. Such white space is typically not intended for inclusion in the delivered version of the document. On the other hand, "significant" white space that should be preserved in the delivered version is common, for example in poetry and source code.
An XML processor must always pass all characters in a document that are not markup through to the application. A validating XML processor must also inform the application which of these characters constitute white space appearing in element content.
A special attribute named `xml:space` may be attached to an element to signal an intention that in that element, white space should be preserved by applications. In valid documents, this attribute, like any other, must be declared if it is used. When declared, it must be given as an enumerated type whose only possible values are "`default`" and "`preserve`". For example:

```
    <!ATTLIST poem   xml:space (default|preserve) 'preserve'>
```

The value "`default`" signals that applications' default white-space processing modes are acceptable for this element; the value "`preserve`" indicates the intent that applications preserve all the white space. This declared intent is considered to apply to all elements within the content of the element where it is specified, unless overriden with another instance of the `xml:space` attribute.
The root element of any document is considered to have signaled no intentions as regards application space handling, unless it provides a value for this attribute or the attribute is declared with a default value.

2.11 End-of-Line Handling

XML parsed entities are often stored in computer files which, for editing convenience, are organized into lines. These lines are typically separated by some combination of the characters carriage-return (#xD) and line-feed (#xA).
To simplify the tasks of applications, wherever an external parsed entity or the literal entity value of an internal parsed entity contains either the literal two-character sequence "#xD#xA" or a standalone literal #xD, an XML processor must pass to the application the single character #xA. (This behavior can conveniently be produced by normalizing all line breaks to #xA on input, before parsing.)

2.12 Language Identification

In document processing, it is often useful to identify the natural or formal language in which the content is written. A special attribute named `xml:lang` may be inserted in documents to specify the language used in the contents and attribute values of any element in an XML document. In valid documents, this attribute, like any other, must be declared if it is used. The values of the attribute are language identifiers as defined by [IETF RFC 1766], "Tags for the Identification of Languages":

Language Identification

```
[33] LanguageID :: Langcode ('-' Subcode)*
                =
[34]   Langcode :: ISO639Code |  IanaCode |  UserCode
                =
[35] ISO639Code :: ([a-z] | [A-Z]) ([a-z] | [A-Z])
                =
[36]   IanaCode :: ('i' | 'I') '-' ([a-z] | [A-Z])+
                =
[37]   UserCode :: ('x' | 'X') '-' ([a-z] | [A-Z])+
                =
[38]    Subcode :: ([a-z] | [A-Z])+
                =
```

The Langcode may be any of the following:

- a two-letter language code as defined by [ISO 639], "Codes for the representation of names of languages"
- a language identifier registered with the Internet Assigned Numbers Authority [IANA]; these begin with the prefix "`i-`" (or "`I-`")
- a language identifier assigned by the user, or agreed on between parties in private use; these must begin with the prefix "`x-`" or "`X-`" in order to ensure that they do not conflict with names later standardized or registered with IANA

There may be any number of Subcode segments; if the first subcode segment exists and the Subcode consists of two letters, then it must be a country code from [ISO 3166], "Codes for the representation of names of countries." If the first subcode consists of more than two letters, it must be a subcode for the language in question registered with IANA, unless the Langcode begins with the prefix "`x-`" or "`X-`".

It is customary to give the language code in lower case, and the country code (if any) in upper case. Note that these values, unlike other names in XML documents, are case insensitive.

For example:

```
<p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
  <l>Habe nun, ach! Philosophie,</l>
  <l>Juristerei, und Medizin</l>
  <l>und leider auch Theologie</l>
  <l>durchaus studiert mit heißem Bemüh'n.</l>
  </sp>
```

The intent declared with `xml:lang` is considered to apply to all attributes and content of the element where it is specified, unless overridden with an instance of `xml:lang` on another element within that content.

A simple declaration for `xml:lang` might take the form

```
xml:lang  NMTOKEN  #IMPLIED
```

but specific default values may also be given, if appropriate. In a collection of French poems for English students, with glosses and notes in English, the xml:lang attribute might be declared this way:

```
    <!ATTLIST poem   xml:lang NMTOKEN 'fr'>
    <!ATTLIST gloss  xml:lang NMTOKEN 'en'>
    <!ATTLIST note   xml:lang NMTOKEN 'en'>
```

3. Logical Structures

Each XML document contains one or more **elements**, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, sometimes called its "generic identifier" (GI), and may have a set of attribute specifications. Each attribute specification has a name and a value.

Element

```
[39] element ::= EmptyElemTag
               | STag content ETag [ WFC: Element Type Match ]
                                   [ VC: Element Valid ]
```

This specification does not constrain the semantics, use, or (beyond syntax) names of the element types and attributes, except that names beginning with a match to `(('X'|'x')('M'|'m')('L'|'l'))` are reserved for standardization in this or future versions of this specification.

Well-Formedness Constraint: Element Type Match
The `Name` in an element's end-tag must match the element type in the start-tag.

Validity Constraint: Element Valid
An element is valid if there is a declaration matching `elementdecl` where the `Name` matches the element type, and one of the following holds:

1. The declaration matches `EMPTY` and the element has no content.
2. The declaration matches `children` and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space (characters matching the nonterminal S) between each pair of child elements.
3. The declaration matches `Mixed` and the content consists of character data and child elements whose types match names in the content model.
4. The declaration matches `ANY`, and the types of any child elements have been declared.

3.1 Start-Tags, End-Tags, and Empty-Element Tags

The beginning of every non-empty XML element is marked by a **start-tag**.

Start-tag

```
[40]      STag ::= '<' Name (S Attribute)*    [ WFC: Unique Att Spec ]
                 = S? '>'
[41] Attribute ::= Name Eq AttValue           [ VC: Attribute Value Type ]
                 =
                                              [ WFC: No External Entity
                                                References ]
                                              [ WFC: No < in Attribute
                                                Values ]
```

The `Name` in the start- and end-tags gives the element's **type**. The `Name`-`AttValue` pairs are referred to as the **attribute specifications** of the element, with the `Name` in each pair referred to as the **attribute name** and the content of the `AttValue` (the text between the `'` or `"` delimiters) as the **attribute value**.

Well-Formedness Constraint: Unique Att Spec
No attribute name may appear more than once in the same start-tag or empty-element tag.

Validity Constraint: Attribute Value Type
The attribute must have been declared; the value must be of the type declared for it. (For attribute types, see "3.3 Attribute-List Declarations".)

Well-Formedness Constraint: No External Entity References
Attribute values cannot contain direct or indirect entity references to external entities.

Well-Formedness Constraint: No < in Attribute Values

The replacement text of any entity referred to directly or indirectly in an attribute value (other than "<") must not contain a <.

An example of a start-tag:

```
<termdef id="dt-dog" term="dog">
```

The end of every element that begins with a start-tag must be marked by an **end-tag** containing a name that echoes the element's type as given in the start-tag:

End-tag

```
[42] ETag ::= '</' Name S? '>'
```

An example of an end-tag:

```
</termdef>
```

The text between the start-tag and end-tag is called the element's **content**:

Content of Elements

```
[43] content ::= (element | CharData | Reference | CDSect | PI | Comment)*
```

If an element is **empty**, it must be represented either by a start-tag immediately followed by an end-tag or by an empty-element tag. An **empty-element tag** takes a special form:

Tags for Empty Elements

```
[44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>' [ WFC: Unique Att Spec ]
```

Empty-element tags may be used for any element which has no content, whether or not it is declared using the keyword EMPTY. For interoperability, the empty-element tag must be used, and can only be used, for elements which are declared EMPTY.

Examples of empty elements:

```
<IMG align="left"
 src="http://www.w3.org/Icons/WWW/w3c_home" />
<br></br>
<br/>
```

3.2 Element Type Declarations

The element structure of an XML document may, for validation purposes, be constrained using element type and attribute-list declarations. An element type declaration constrains the element's content.

Element type declarations often constrain which element types can appear as children of the element. At user option, an XML processor may issue a warning when a declaration mentions an element type for which no declaration is provided, but this is not an error.

An **element type declaration** takes the form:

Element Type Declaration

```
[45] elementdecl ::= '<!ELEMENT' S Name S                [ VC: Unique Element Type
                     contentspec S? '>'                    Declaration ]
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed
                   | children
```

where the Name gives the element type being declared.

Validity Constraint: Unique Element Type Declaration

No element type may be declared more than once.

Examples of element type declarations:

```
<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA|emph)* >
<!ELEMENT %name.para; %content.para; >
<!ELEMENT container ANY>
```

3.2.1 Element Content

An element type has **element content** when elements of that type must contain only child elements (no character data), optionally separated by white space (characters matching the nonterminal S). In this case, the constraint includes a content model, a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear. The grammar is built on content particles (cps), which consist of names, choice lists of content particles, or sequence lists of content particles:

Element-content Models

```
[47] children :: (choice | seq) ('?' | '*'
             =  | '+')?
[48]       cp :: (Name | choice | seq) ('?'
             =  | '*' | '+')?
[49]   choice :: '(' S? cp ( S? '|' S? cp )*   [ VC: Proper Group/PE
             =  S? ')'                           Nesting ]
[50]      seq :: '(' S? cp ( S? ',' S? cp )*   [ VC: Proper Group/PE
             =  S? ')'                           Nesting ]
```

where each Name is the type of an element which may appear as a child. Any content particle in a choice list may appear in the element content at the location where the choice list appears in the grammar; content particles occurring in a sequence list must each appear in the element content in the order given in the list. The optional character following a name or list governs whether the element or the content particles in the list may occur one or more (+), zero or more (*), or zero or one times (?). The absence of such an operator means that the element or content particle must appear exactly once. This syntax and meaning are identical to those used in the productions in this specification.
The content of an element matches a content model if and only if it is possible to trace out a path through the content model, obeying the sequence, choice, and repetition operators and matching each element in the content against an element type in the content model. For compatibility, it is an error if an element in the document can match more than one occurrence of an element type in the content model. For more information, see "E. Deterministic Content Models".

Validity Constraint: Proper Group/PE Nesting

Parameter-entity replacement text must be properly nested with parenthetized groups. That is to say, if either of the opening or closing parentheses in a choice, seq, or Mixed construct is contained in the replacement text for a parameter entity, both must be contained in the same replacement text. For interoperability, if a parameter-entity reference appears in a choice, seq, or Mixed construct, its replacement text should not be empty, and neither the first nor last non-blank character of the replacement text should be a connector (| or ,).

Examples of element-content models:

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
<!ELEMENT dictionary-body (%div.mix; | %dict.mix;)*>
```

3.2.2 Mixed Content

An element type has **mixed content** when elements of that type may contain character data, optionally interspersed with child elements. In this case, the types of the child elements may be constrained, but not their order or their number of occurrences:

Mixed-content Declaration

```
[51] Mixed :: '(' S? '#PCDATA' (S? '|' S? Name)*
          = S? ')*'
             | '(' S? '#PCDATA' S? ')'            [ VC: Proper Group/PE
                                                    Nesting ]
                                                  [ VC: No Duplicate Types ]
```

where the Names give the types of elements that may appear as children.

Validity Constraint: No Duplicate Types

The same name must not appear more than once in a single mixed-content declaration.

Examples of mixed content declarations:

```
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >
<!ELEMENT b (#PCDATA)>
```

3.3 Attribute-List Declarations

Attributes are used to associate name-value pairs with elements. Attribute specifications may appear only within start-tags and empty-element tags; thus, the productions used to recognize them appear in "3.1 Start-Tags, End-Tags, and Empty-Element Tags". Attribute-list declarations may be used:

- To define the set of attributes pertaining to a given element type.
- To establish type constraints for these attributes.
- To provide default values for attributes.

Attribute-list declarations specify the name, data type, and default value (if any) of each attribute associated with a given element type:

Attribute-list Declaration

```
[52] AttlistDecl :: '<!ATTLIST' S Name AttDef* S? '>'
                 =
[53]      AttDef :: S Name S AttType S DefaultDecl
                 =
```

The Name in the AttlistDecl rule is the type of an element. At user option, an XML processor may issue a warning if attributes are declared for an element type not itself declared, but this is not an error. The Name in the AttDef rule is the name of the attribute.

When more than one AttlistDecl is provided for a given element type, the contents of all those provided are merged. When more than one definition is provided for the same attribute of a given element type, the first declaration is binding and later declarations are ignored. For interoperability, writers of DTDs may choose to provide at most one attribute-list declaration for a given element type, at most one attribute definition for a given attribute name, and at least one attribute definition in each attribute-list declaration. For interoperability, an XML processor may at user option issue a warning when more than one attribute-list declaration is provided for a given element type, or more than one attribute definition is provided for a given attribute, but this is not an error.

3.3.1 Attribute Types

XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The string type may take any literal string as a value; the tokenized types have varying lexical and semantic constraints, as noted:

Attribute Types

```
[54]       AttType :: StringType | TokenizedType
                   =  | EnumeratedType
[55]    StringType :: 'CDATA'
                   =
[56] TokenizedType :: 'ID'                              [ VC: ID ]
                   =
                                                        [ VC: One ID per
                                                          Element Type ]
                                                        [ VC: ID Attribute
                                                          Default ]
                      | 'IDREF'                         [ VC: IDREF ]
                      | 'IDREFS'                        [ VC: IDREF ]
                      | 'ENTITY'                        [ VC: Entity Name ]
                      | 'ENTITIES'                      [ VC: Entity Name ]
                      | 'NMTOKEN'                       [ VC: Name Token ]
                      | 'NMTOKENS'                      [ VC: Name Token ]
```

Validity Constraint: ID

Values of type `ID` must match the `Name` production. A name must not appear more than once in an XML document as a value of this type; i.e., ID values must uniquely identify the elements which bear them.

Validity Constraint: One ID per Element Type

No element type may have more than one ID attribute specified.

Validity Constraint: ID Attribute Default

An ID attribute must have a declared default of `#IMPLIED` or `#REQUIRED`.

Validity Constraint: IDREF

Values of type `IDREF` must match the `Name` production, and values of type `IDREFS` must match `Names`; each `Name` must match the value of an ID attribute on some element in the XML document; i.e. `IDREF` values must match the value of some ID attribute.

Validity Constraint: Entity Name

Values of type `ENTITY` must match the `Name` production, values of type `ENTITIES` must match `Names`; each `Name` must match the name of an unparsed entity declared in the DTD.

Validity Constraint: Name Token

Values of type `NMTOKEN` must match the `Nmtoken` production; values of type `NMTOKENS` must match Nmtokens.

Enumerated attributes can take one of a list of values provided in the declaration. There are two kinds of enumerated types:

Enumerated Attribute Types

```
[57] EnumeratedType :: NotationType
                    =  | Enumeration
[58]   NotationType :: 'NOTATION' S '(' S? Name    [ VC: Notation
                    =  (S? '|' S? Name)* S? ')'      Attributes ]
[59]    Enumeration :: '(' S? Nmtoken (S? '|' S?   [ VC: Enumeration ]
                    =  Nmtoken)* S? ')'
```

A `NOTATION` attribute identifies a notation, declared in the DTD with associated system and/or public identifiers, to be used in interpreting the element to which the attribute is attached.

Validity Constraint: Notation Attributes

Values of this type must match one of the notation names included in the declaration; all notation names in the declaration must be declared.

Validity Constraint: Enumeration

Values of this type must match one of the `Nmtoken` tokens in the declaration.

For interoperability, the same `Nmtoken` should not occur more than once in the enumerated attribute types of a single element type.

3.3.2 Attribute Defaults

An attribute declaration provides information on whether the attribute's presence is required, and if not, how an XML processor should react if a declared attribute is absent in a document.

Attribute Defaults

```
[60] DefaultDecl :: '#REQUIRED' | '#IMPLIED'
                 =
                    | (('#FIXED' S)? AttValue) [ VC: Required Attribute ]
                                               [ VC: Attribute Default
                                                 Legal ]
                                               [ WFC: No < in Attribute
                                                 Values ]
                                               [ VC: Fixed Attribute
                                                 Default ]
```

In an attribute declaration, `#REQUIRED` means that the attribute must always be provided, `#IMPLIED` that no default value is provided. If the declaration is neither `#REQUIRED` nor `#IMPLIED`, then the `AttValue` value contains the declared **default** value; the `#FIXED` keyword states that the attribute must always have the default value. If a default value is declared, when an XML processor encounters an omitted attribute, it is to behave as though the attribute were present with the declared default value.

Validity Constraint: Required Attribute

If the default declaration is the keyword `#REQUIRED`, then the attribute must be specified for all elements of the type in the attribute-list declaration.

Validity Constraint: Attribute Default Legal

The declared default value must meet the lexical constraints of the declared attribute type.

Validity Constraint: Fixed Attribute Default

If an attribute has a default value declared with the `#FIXED` keyword, instances of that attribute must match the default value.

Examples of attribute-list declarations:

```
<!ATTLIST termdef
          id      ID      #REQUIRED
          name    CDATA   #IMPLIED>
<!ATTLIST list
          type    (bullets|ordered|glossary)  "ordered">
<!ATTLIST form
          method  CDATA   #FIXED "POST">
```

3.3.3 Attribute-Value Normalization

Before the value of an attribute is passed to the application or checked for validity, the XML processor must normalize it as follows:

- a character reference is processed by appending the referenced character to the attribute value
- an entity reference is processed by recursively processing the replacement text of the entity
- a whitespace character (#x20, #xD, #xA, #x9) is processed by appending #x20 to the normalized value, except that only a single #x20 is appended for a "#xD#xA" sequence that is part of an external parsed entity or the literal entity value of an internal parsed entity
- other characters are processed by appending them to the normalized value

If the declared value is not CDATA, then the XML processor must further process the normalized attribute value by discarding any leading and trailing space (#x20) characters, and by replacing sequences of space (#x20) characters by a single space (#x20) character.

All attributes for which no declaration has been read should be treated by a non-validating parser as if declared `CDATA`.

3.4 Conditional Sections

Conditional sections are portions of the document type declaration external subset which are included in, or excluded from, the logical structure of the DTD based on the keyword which governs them.

Conditional Section

```
[61]    conditionalSect :: includeSect | ignoreSect
                        =
[62]        includeSect :: '<![' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'
                        =
[63]         ignoreSect :: '<![' S? 'IGNORE' S? '[' ignoreSectContents* ']]>'
                        =
[64] ignoreSectContents :: Ignore ('<![' ignoreSectContents ']]>' Ignore)*
                        =
[65]             Ignore :: Char* - (Char* ('<![' | ']]>') Char*)
                        =
```

Like the internal and external DTD subsets, a conditional section may contain one or more complete declarations, comments, processing instructions, or nested conditional sections, intermingled with white space.
If the keyword of the conditional section is `INCLUDE`, then the contents of the conditional section are part of the DTD. If the keyword of the conditional section is `IGNORE`, then the contents of the conditional section are not logically part of the DTD. Note that for reliable parsing, the contents of even ignored conditional sections must be read in order to detect nested conditional sections and ensure that the end of the outermost (ignored) conditional section is properly detected. If a conditional section with a keyword of `INCLUDE` occurs within a larger conditional section with a keyword of `IGNORE`, both the outer and the inner conditional sections are ignored.
If the keyword of the conditional section is a parameter-entity reference, the parameter entity must be replaced by its content before the processor decides whether to include or ignore the conditional section.
An example:

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >

<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

4. Physical Structures

An XML document may consist of one or many storage units. These are called **entities**; they all have **content** and are all (except for the document entity, see below, and the external DTD subset) identified by **name**. Each XML document has one entity called the document entity, which serves as the starting point for the XML processor and may contain the whole document.
Entities may be either parsed or unparsed. A **parsed entity's** contents are referred to as its replacement text; this text is considered an integral part of the document.
An **unparsed entity** is a resource whose contents may or may not be text, and if text, may not be XML. Each unparsed entity has an associated notation, identified by name. Beyond a requirement that an XML processor make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.
Parsed entities are invoked by name using entity references; unparsed entities by name, given in the value of `ENTITY` or `ENTITIES` attributes.
General entities are entities for use within the document content. In this specification, general entities are sometimes referred to with the unqualified term *entity* when this leads to no ambiguity. Parameter entities are parsed entities for use within the DTD. These two types of entities use different forms of reference and are recognized in different contexts. Furthermore, they occupy different namespaces; a parameter entity and a general entity with the same name

are two distinct entities.

4.1 Character and Entity References

A **character reference** refers to a specific character in the ISO/IEC 10646 character set, for example one not directly accessible from available input devices.

Character Reference

```
[66] CharRef ::= '&#' [0-9]+ ';'
                 | '&#x' [0-9a-fA-F]+ ';' [ WFC: Legal Character ]
```

Well-Formedness Constraint: Legal Character

Characters referred to using character references must match the production for Char.

If the character reference begins with "&#x", the digits and letters up to the terminating ; provide a hexadecimal representation of the character's code point in ISO/IEC 10646. If it begins just with "&#", the digits up to the terminating ; provide a decimal representation of the character's code point.

An **entity reference** refers to the content of a named entity. References to parsed general entities use ampersand (&) and semicolon (;) as delimiters. **Parameter-entity references** use percent-sign (%) and semicolon (;) as delimiters.

Entity Reference

```
[67]   Reference ::= EntityRef | CharRef
[68]   EntityRef ::= '&' Name ';'          [ WFC: Entity Declared ]
                                           [ VC: Entity Declared ]
                                           [ WFC: Parsed Entity ]
                                           [ WFC: No Recursion ]
[69] PEReference ::= '%' Name ';'          [ VC: Entity Declared ]
                                           [ WFC: No Recursion ]
                                           [ WFC: In DTD ]
```

Well-Formedness Constraint: Entity Declared

In a document without any DTD, a document with only an internal DTD subset which contains no parameter entity references, or a document with "`standalone='yes'`", the Name given in the entity reference must match that in an entity declaration, except that well-formed documents need not declare any of the following entities: `amp`, `lt`, `gt`, `apos`, `quot`. The declaration of a parameter entity must precede any reference to it. Similarly, the declaration of a general entity must precede any reference to it which appears in a default value in an attribute-list declaration. Note that if entities are declared in the external subset or in external parameter entities, a non-validating processor is not obligated to read and process their declarations; for such documents, the rule that an entity must be declared is a well-formedness constraint only if standalone='yes'.

Validity Constraint: Entity Declared

In a document with an external subset or external parameter entities with "`standalone='no'`", the Name given in the entity reference must match that in an entity declaration. For interoperability, valid documents should declare the entities `amp`, `lt`, `gt`, `apos`, `quot`, in the form specified in "4.6 Predefined Entities". The declaration of a parameter entity must precede any reference to it. Similarly, the declaration of a general entity must precede any reference to it which appears in a default value in an attribute-list declaration.

Well-Formedness Constraint: Parsed Entity

An entity reference must not contain the name of an unparsed entity. Unparsed entities may be referred to only in attribute values declared to be of type `ENTITY` or `ENTITIES`.

Well-Formedness Constraint: No Recursion

A parsed entity must not contain a recursive reference to itself, either directly or indirectly.

Well-Formedness Constraint: In DTD

Parameter-entity references may only appear in the DTD.

Examples of character and entity references:

```
Type <key>less-than</key> (&#x3C;) to save options.
This document was prepared on &docdate; and
is classified &security-level;.
```

Example of a parameter-entity reference:

```
<!-- declare the parameter entity "ISOLat2"... -->
<!ENTITY % ISOLat2
         SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ... now reference it. -->
%ISOLat2;
```

4.2 Entity Declarations

Entities are declared thus:

Entity Declaration

```
[70] EntityDecl :: GEDecl | PEDecl
                =
[71]     GEDecl :: '<!ENTITY' S Name S EntityDef S? '>'
                =
[72]     PEDecl :: '<!ENTITY' S '%' S Name S PEDef S? '>'
                =
[73]  EntityDef :: EntityValue | (ExternalID NDataDecl?)
                =
[74]      PEDef :: EntityValue | ExternalID
                =
```

The `Name` identifies the entity in an entity reference or, in the case of an unparsed entity, in the value of an `ENTITY` or `ENTITIES` attribute. If the same entity is declared more than once, the first declaration encountered is binding; at user option, an XML processor may issue a warning if entities are declared multiple times.

4.2.1 Internal Entities

If the entity definition is an `EntityValue`, the defined entity is called an **internal entity**. There is no separate physical storage object, and the content of the entity is given in the declaration. Note that some processing of entity and character references in the literal entity value may be required to produce the correct replacement text: see "4.5 Construction of Internal Entity Replacement Text".

An internal entity is a parsed entity.

Example of an internal entity declaration:

```
<!ENTITY Pub-Status "This is a pre-release of the
 specification.">
```

4.2.2 External Entities

If the entity is not internal, it is an **external entity**, declared as follows:

External Entity Declaration

```
[75] ExternalID :: 'SYSTEM' S SystemLiteral
                =
                   | 'PUBLIC' S PubidLiteral S
                   SystemLiteral
[76]  NDataDecl :: S 'NDATA' S Name                    [ VC: Notation Declared ]
                =
```

If the NDataDecl is present, this is a general unparsed entity; otherwise it is a parsed entity.

Validity Constraint: Notation Declared

The Name must match the declared name of a notation.

The SystemLiteral is called the entity's **system identifier**. It is a URI, which may be used to retrieve the entity. Note that the hash mark (#) and fragment identifier frequently used with URIs are not, formally, part of the URI itself; an XML processor may signal an error if a fragment identifier is given as part of a system identifier. Unless otherwise provided by information outside the scope of this specification (e.g. a special XML element type defined by a particular DTD, or a processing instruction defined by a particular application specification), relative URIs are relative to the location of the resource within which the entity declaration occurs. A URI might thus be relative to the document entity, to the entity containing the external DTD subset, or to some other external parameter entity.

An XML processor should handle a non-ASCII character in a URI by representing the character in UTF-8 as one or more bytes, and then escaping these bytes with the URI escaping mechanism (i.e., by converting each byte to %HH, where HH is the hexadecimal notation of the byte value).

In addition to a system identifier, an external identifier may include a **public identifier**. An XML processor attempting to retrieve the entity's content may use the public identifier to try to generate an alternative URI. If the processor is unable to do so, it must use the URI specified in the system literal. Before a match is attempted, all strings of white space in the public identifier must be normalized to single space characters (#x20), and leading and trailing white space must be removed.

Examples of external entity declarations:

```
<!ENTITY open-hatch
         SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
         SYSTEM "../grafix/OpenHatch.gif"
         NDATA gif >
```

4.3 Parsed Entities

4.3.1 The Text Declaration

External parsed entities may each begin with a **text declaration**.

Text Declaration

```
[77] TextDecl :: '<?xml' VersionInfo? EncodingDecl S? '?>'
              =
```

The text declaration must be provided literally, not by reference to a parsed entity. No text declaration may appear at any position other than the beginning of an external parsed entity.

4.3.2 Well-Formed Parsed Entities

The document entity is well-formed if it matches the production labeled document. An external general parsed entity is well-formed if it matches the production labeled extParsedEnt. An external parameter entity is well-formed if it matches the production labeled extPE.

Well-Formed External Parsed Entity

```
[78] extParsedEnt :: TextDecl? content
                  =
[79]        extPE :: TextDecl? extSubsetDecl
                  =
```

An internal general parsed entity is well-formed if its replacement text matches the production labeled `content`.
All internal parameter entities are well-formed by definition.
A consequence of well-formedness in entities is that the logical and physical structures in an XML document are properly nested; no start-tag, end-tag, empty-element tag, element, comment, processing instruction, character reference, or entity reference can begin in one entity and end in another.

4.3.3 Character Encoding in Entities

Each external parsed entity in an XML document may use a different encoding for its characters. All XML processors must be able to read entities in either UTF-8 or UTF-16.
Entities encoded in UTF-16 must begin with the Byte Order Mark described by ISO/IEC 10646 Annex E and Unicode Appendix B (the ZERO WIDTH NO-BREAK SPACE character, #xFEFF). This is an encoding signature, not part of either the markup or the character data of the XML document. XML processors must be able to use this character to differentiate between UTF-8 and UTF-16 encoded documents.
Although an XML processor is required to read only entities in the UTF-8 and UTF-16 encodings, it is recognized that other encodings are used around the world, and it may be desired for XML processors to read entities that use them. Parsed entities which are stored in an encoding other than UTF-8 or UTF-16 must begin with a text declaration containing an encoding declaration:

Encoding Declaration

```
[80] EncodingDecl :: S 'encoding' Eq ('"'
                  = EncName '"' |  "'"
                    EncName "'" )
[81]      EncName :: [A-Za-z] ([A-Za-z0-9._]     / Encoding name contains
                  = | '-')*                      * only Latin characters */
```

In the document entity, the encoding declaration is part of the XML declaration. The `EncName` is the name of the encoding used.
In an encoding declaration, the values "`UTF-8`", "`UTF-16`", "`ISO-10646-UCS-2`", and "`ISO-10646-UCS-4`" should be used for the various encodings and transformations of Unicode / ISO/IEC 10646, the values "`ISO-8859-1`", "`ISO-8859-2`", ... "`ISO-8859-9`" should be used for the parts of ISO 8859, and the values "`ISO-2022-JP`", "`Shift_JIS`", and "`EUC-JP`" should be used for the various encoded forms of JIS X-0208-1997. XML processors may recognize other encodings; it is recommended that character encodings registered (as *charsets*) with the Internet Assigned Numbers Authority [IANA], other than those just listed, should be referred to using their registered names. Note that these registered names are defined to be case-insensitive, so processors wishing to match against them should do so in a case-insensitive way.
In the absence of information provided by an external transport protocol (e.g. HTTP or MIME), it is an error for an entity including an encoding declaration to be presented to the XML processor in an encoding other than that named in the declaration, for an encoding declaration to occur other than at the beginning of an external entity, or for an entity which begins with neither a Byte Order Mark nor an encoding declaration to use an encoding other than UTF-8. Note that since ASCII is a subset of UTF-8, ordinary ASCII entities do not strictly need an encoding declaration.
It is a fatal error when an XML processor encounters an entity with an encoding that it is unable to process.
Examples of encoding declarations:

```
<?xml encoding='UTF-8'?>
<?xml encoding='EUC-JP'?>
```

4.4 XML Processor Treatment of Entities and References

The table below summarizes the contexts in which character references, entity references, and invocations of unparsed entities might appear and the required behavior of an XML processor in each case. The labels in the leftmost column describe the recognition context:

Reference in Content
as a reference anywhere after the start-tag and before the end-tag of an element; corresponds to the nonterminal `content`.

Reference in Attribute Value
as a reference within either the value of an attribute in a start-tag, or a default value in an attribute declaration; corresponds to the nonterminal `AttValue`.

Occurs as Attribute Value
as a `Name`, not a reference, appearing either as the value of an attribute which has been declared as type `ENTITY`, or as one of the space-separated tokens in the value of an attribute which has been declared as type `ENTITIES`.

Reference in Entity Value
as a reference within a parameter or internal entity's literal entity value in the entity's declaration; corresponds to the nonterminal `EntityValue`.

Reference in DTD
as a reference within either the internal or external subsets of the DTD, but outside of an `EntityValue` or `AttValue`.

	Entity Type				
	Parameter	Internal General	External Parsed General	Unparsed	Character
Reference in Content	Not recognized	Included	Included if validating	Forbidden	Included
Reference in Attribute Value	Not recognized	Included in literal	Forbidden	Forbidden	Included
Occurs as Attribute Value	Not recognized	Forbidden	Forbidden	Notify	Not recognized
Reference in EntityValue	Included in literal	Bypassed	Bypassed	Forbidden	Included
Reference in DTD	Included as PE	Forbidden	Forbidden	Forbidden	Forbidden

4.4.1 Not Recognized

Outside the DTD, the % character has no special significance; thus, what would be parameter entity references in the DTD are not recognized as markup in `content`. Similarly, the names of unparsed entities are not recognized except when they appear in the value of an appropriately declared attribute.

4.4.2 Included

An entity is **included** when its replacement text is retrieved and processed, in place of the reference itself, as though it were part of the document at the location the reference was recognized. The replacement text may contain both character data and (except for parameter entities) markup, which must be recognized in the usual way, except that the replacement text of entities used to escape markup delimiters (the entities `amp`, `lt`, `gt`, `apos`, `quot`) is always treated as data. (The string "`AT&T;`" expands to "`AT&T;`" and the remaining ampersand is not recognized as an entity-reference delimiter.) A character reference is **included** when the indicated character is processed in place of the reference itself.

4.4.3 Included If Validating

When an XML processor recognizes a reference to a parsed entity, in order to validate the document, the processor must include its replacement text. If the entity is external, and the processor is not attempting to validate the XML

document, the processor may, but need not, include the entity's replacement text. If a non-validating parser does not include the replacement text, it must inform the application that it recognized, but did not read, the entity.
This rule is based on the recognition that the automatic inclusion provided by the SGML and XML entity mechanism, primarily designed to support modularity in authoring, is not necessarily appropriate for other applications, in particular document browsing. Browsers, for example, when encountering an external parsed entity reference, might choose to provide a visual indication of the entity's presence and retrieve it for display only on demand.

4.4.4 Forbidden

The following are forbidden, and constitute fatal errors:

- the appearance of a reference to an unparsed entity.
- the appearance of any character or general-entity reference in the DTD except within an `EntityValue` or `AttValue`.
- a reference to an external entity in an attribute value.

4.4.5 Included in Literal

When an entity reference appears in an attribute value, or a parameter entity reference appears in a literal entity value, its replacement text is processed in place of the reference itself as though it were part of the document at the location the reference was recognized, except that a single or double quote character in the replacement text is always treated as a normal data character and will not terminate the literal. For example, this is well-formed:

```
<!ENTITY % YN '"Yes"' >
<!ENTITY WhatHeSaid "He said &YN;" >
```

while this is not:

```
<!ENTITY EndAttr "27'" >
<element attribute='a-&EndAttr;>
```

4.4.6 Notify

When the name of an unparsed entity appears as a token in the value of an attribute of declared type `ENTITY` or `ENTITIES`, a validating processor must inform the application of the system and public (if any) identifiers for both the entity and its associated notation.

4.4.7 Bypassed

When a general entity reference appears in the `EntityValue` in an entity declaration, it is bypassed and left as is.

4.4.8 Included as PE

Just as with external parsed entities, parameter entities need only be included if validating. When a parameter-entity reference is recognized in the DTD and included, its replacement text is enlarged by the attachment of one leading and one following space (#x20) character; the intent is to constrain the replacement text of parameter entities to contain an integral number of grammatical tokens in the DTD.

4.5 Construction of Internal Entity Replacement Text

In discussing the treatment of internal entities, it is useful to distinguish two forms of the entity's value. The **literal entity value** is the quoted string actually present in the entity declaration, corresponding to the non-terminal `EntityValue`. The **replacement text** is the content of the entity, after replacement of character references and parameter-entity references.
The literal entity value as given in an internal entity declaration (`EntityValue`) may contain character, parameter-entity, and general-entity references. Such references must be contained entirely within the literal entity value. The

actual replacement text that is included as described above must contain the *replacement text* of any parameter entities referred to, and must contain the character referred to, in place of any character references in the literal entity value; however, general-entity references must be left as-is, unexpanded. For example, given the following declarations:

```
<!ENTITY % pub    "&#xc9;ditions Gallimard" >
<!ENTITY   rights "All rights reserved" >
<!ENTITY   book   "La Peste: Albert Camus,
&#xA9; 1947 %pub;. &rights;" >
```

then the replacement text for the entity "`book`" is:

```
La Peste: Albert Camus,
© 1947 Éditions Gallimard. &rights;
```

The general-entity reference "`&rights;`" would be expanded should the reference "`&book;`" appear in the document's content or an attribute value.
These simple rules may have complex interactions; for a detailed discussion of a difficult example, see "D. Expansion of Entity and Character References".

4.6 Predefined Entities

Entity and character references can both be used to **escape** the left angle bracket, ampersand, and other delimiters. A set of general entities (`amp`, `lt`, `gt`, `apos`, `quot`) is specified for this purpose. Numeric character references may also be used; they are expanded immediately when recognized and must be treated as character data, so the numeric character references "`&#60;`" and "`&#38;`" may be used to escape < and & when they occur in character data.
All XML processors must recognize these entities whether they are declared or not. For interoperability, valid XML documents should declare these entities, like any others, before using them. If the entities in question are declared, they must be declared as internal entities whose replacement text is the single character being escaped or a character reference to that character, as shown below.

```
<!ENTITY lt     "&#38;#60;">
<!ENTITY gt     "&#62;">
<!ENTITY amp    "&#38;#38;">
<!ENTITY apos   "&#39;">
<!ENTITY quot   "&#34;">
```

Note that the < and & characters in the declarations of "`lt`" and "`amp`" are doubly escaped to meet the requirement that entity replacement be well-formed.

4.7 Notation Declarations

Notations identify by name the format of unparsed entities, the format of elements which bear a notation attribute, or the application to which a processing instruction is addressed.
Notation declarations provide a name for the notation, for use in entity and attribute-list declarations and in attribute specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.

Notation Declarations

```
[82] NotationDecl ::= '<!NOTATION' S Name S (ExternalID |  PublicID) S? '>'

[83]     PublicID ::= 'PUBLIC' S PubidLiteral
```

XML processors must provide applications with the name and external identifier(s) of any notation declared and referred to in an attribute value, attribute definition, or entity declaration. They may additionally resolve the external identifier into the system identifier, file name, or other information needed to allow the application to call a processor for data in the notation described. (It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.)

4.8 Document Entity

The **document entity** serves as the root of the entity tree and a starting-point for an XML processor. This specification does not specify how the document entity is to be located by an XML processor; unlike other entities, the document entity has no name and might well appear on a processor input stream without any identification at all.

5. Conformance

5.1 Validating and Non-Validating Processors

Conforming XML processors fall into two classes: validating and non-validating.
Validating and non-validating processors alike must report violations of this specification's well-formedness constraints in the content of the document entity and any other parsed entities that they read.
Validating processors must report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification. To accomplish this, validating XML processors must read and process the entire DTD and all external parsed entities referenced in the document.
Non-validating processors are required to check only the document entity, including the entire internal DTD subset, for well-formedness. While they are not required to check the document for validity, they are required to **process** all the declarations they read in the internal DTD subset and in any parameter entity that they read, up to the first reference to a parameter entity that they do *not* read; that is to say, they must use the information in those declarations to normalize attribute values, include the replacement text of internal entities, and supply default attribute values. They must not process entity declarations or attribute-list declarations encountered after a reference to a parameter entity that is not read, since the entity may have contained overriding declarations.

5.2 Using XML Processors

The behavior of a validating XML processor is highly predictable; it must read every piece of a document and report all well-formedness and validity violations. Less is required of a non-validating processor; it need not read any part of the document other than the document entity. This has two effects that may be important to users of XML processors:

- Certain well-formedness errors, specifically those that require reading external entities, may not be detected by a non-validating processor. Examples include the constraints entitled Entity Declared, Parsed Entity, and No Recursion, as well as some of the cases described as forbidden in "4.4 XML Processor Treatment of Entities and References".
- The information passed from the processor to the application may vary, depending on whether the processor reads parameter and external entities. For example, a non-validating processor may not normalize attribute values, include the replacement text of internal entities, or supply default attribute values, where doing so depends on having read declarations in external or parameter entities.

For maximum reliability in interoperating between different XML processors, applications which use non-validating processors should not rely on any behaviors not required of such processors. Applications which require facilities such as the use of default attributes or internal entities which are declared in external entities should use validating XML processors.

6. Notation

The formal grammar of XML is given in this specification using a simple Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one symbol, in the form

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are defined by a regular expression, or with an initial lower case letter otherwise. Literal strings are quoted.
Within the expression on the right-hand side of a rule, the following expressions are used to match strings of one or more characters:

#xN
where N is a hexadecimal integer, the expression matches the character in ISO/IEC 10646 whose canonical (UCS-4) code value, when interpreted as an unsigned binary number, has the value indicated. The number of leading zeros in the #xN form is insignificant; the number of leading zeros in the corresponding code value is governed by the character encoding in use and is not significant for XML.

[a-zA-Z], [#xN-#xN]
matches any character with a value in the range(s) indicated (inclusive).

[^a-z], [^#xN-#xN]
matches any character with a value *outside* the range indicated.

[^abc], [^#xN#xN#xN]
matches any character with a value not among the characters given.

"string"
matches a literal string matching that given inside the double quotes.

'string'
matches a literal string matching that given inside the single quotes.

These symbols may be combined to match more complex patterns as follows, where A and B represent simple expressions:

(expression)
expression is treated as a unit and may be combined as described in this list.

A?
matches A or nothing; optional A.

A B
matches A followed by B.

A | B
matches A or B but not both.

A - B
matches any string that matches A but does not match B.

A+
matches one or more occurrences of A.

A*
matches zero or more occurrences of A.

Other notations used in the productions are:

/* ... */
comment.

[wfc: ...]
well-formedness constraint; this identifies by name a constraint on well-formed documents associated with a production.

[vc: ...]
validity constraint; this identifies by name a constraint on valid documents associated with a production.

Appendices

A. References

A.1 Normative References

IANA
(Internet Assigned Numbers Authority) *Official Names for Character Sets*, ed. Keld Simonsen et al. See ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets.

IETF RFC 1766
IETF (Internet Engineering Task Force). *RFC 1766: Tags for the Identification of Languages*, ed. H. Alvestrand. 1995.

ISO 639

(International Organization for Standardization). *ISO 639:1988 (E). Code for the representation of names of languages.* [Geneva]: International Organization for Standardization, 1988.

ISO 3166

(International Organization for Standardization). *ISO 3166-1:1997 (E). Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes* [Geneva]: International Organization for Standardization, 1997.

ISO/IEC 10646

ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).

Unicode

The Unicode Consortium. *The Unicode Standard, Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press, 1996.

A.2 Other References

Aho/Ullman

Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Reading: Addison-Wesley, 1986, rpt. corr. 1988.

Berners-Lee et al.

Berners-Lee, T., R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax and Semantics*. 1997. (Work in progress; see updates to RFC1738.)

Brüggemann-Klein

Brüggemann-Klein, Anne. *Regular Expressions into Finite Automata*. Extended abstract in I. Simon, Hrsg., LATIN 1992, S. 97-98. Springer-Verlag, Berlin 1992. Full Version in Theoretical Computer Science 120: 197-213, 1993.

Brüggemann-Klein and Wood

Brüggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages*. Universität Freiburg, Institut für Informatik, Bericht 38, Oktober 1991.

Clark

James Clark. Comparison of SGML and XML. See http://www.w3.org/TR/NOTE-sgml-xml-971215.

IETF RFC1738

IETF (Internet Engineering Task Force). *RFC 1738: Uniform Resource Locators (URL)*, ed. T. Berners-Lee, L. Masinter, M. McCahill. 1994.

IETF RFC1808

IETF (Internet Engineering Task Force). *RFC 1808: Relative Uniform Resource Locators*, ed. R. Fielding. 1995.

IETF RFC2141

IETF (Internet Engineering Task Force). *RFC 2141: URN Syntax*, ed. R. Moats. 1997.

ISO 8879

ISO (International Organization for Standardization). *ISO 8879:1986(E). Information processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML).* First edition -- 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO/IEC 10744

ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology -- Hypermedia/Time-based Structuring Language (HyTime).* [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe.* [Geneva]: International Organization for Standardization, 1996.

B. Character Classes

Following the characteristics defined in the Unicode standard, characters are classed as base characters (among others, these contain the alphabetic characters of the Latin alphabet, without diacritics), ideographic characters, and combining characters (among others, this class contains most diacritics); these classes combine to form the class of letters. Digits and extenders are also distinguished.

Characters

```
[84]        Letter :: BaseChar | Ideographic
                   =
[85]      BaseChar :: [#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6]
                   =  | [#x00D8-#x00F6] | [#x00F8-#x00FF] | [#x0100-#x0131]
                      | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E]
                      | [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5]
                      | [#x01FA-#x0217] | [#x0250-#x02A8] | [#x02BB-#x02C1]
                      | #x0386 | [#x0388-#x038A] | #x038C | [#x038E-#x03A1]
                      | [#x03A3-#x03CE] | [#x03D0-#x03D6] | #x03DA | #x03DC
                      | #x03DE | #x03E0 | [#x03E2-#x03F3] | [#x0401-#x040C]
                      | [#x040E-#x044F] | [#x0451-#x045C] | [#x045E-#x0481]
                      | [#x0490-#x04C4] | [#x04C7-#x04C8] | [#x04CB-#x04CC]
                      | [#x04D0-#x04EB] | [#x04EE-#x04F5] | [#x04F8-#x04F9]
                      | [#x0531-#x0556] | #x0559 | [#x0561-#x0586] | [#x05D0-
                     #x05EA] | [#x05F0-#x05F2] | [#x0621-#x063A] | [#x0641-
                     #x064A] | [#x0671-#x06B7] | [#x06BA-#x06BE] | [#x06C0-
                     #x06CE] | [#x06D0-#x06D3] | #x06D5 | [#x06E5-#x06E6]
                      | [#x0905-#x0939] | #x093D | [#x0958-#x0961] | [#x0985-
                     #x098C] | [#x098F-#x0990] | [#x0993-#x09A8] | [#x09AA-
                     #x09B0] | #x09B2 | [#x09B6-#x09B9] | [#x09DC-#x09DD]
                      | [#x09DF-#x09E1] | [#x09F0-#x09F1] | [#x0A05-#x0A0A]
                      | [#x0A0F-#x0A10] | [#x0A13-#x0A28] | [#x0A2A-#x0A30]
                      | [#x0A32-#x0A33] | [#x0A35-#x0A36] | [#x0A38-#x0A39]
                      | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74] | [#x0A85-
                     #x0A8B] | #x0A8D | [#x0A8F-#x0A91] | [#x0A93-#x0AA8]
                      | [#x0AAA-#x0AB0] | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9]
                      | #x0ABD | #x0AE0 | [#x0B05-#x0B0C] | [#x0B0F-#x0B10]
                      | [#x0B13-#x0B28] | [#x0B2A-#x0B30] | [#x0B32-#x0B33]
                      | [#x0B36-#x0B39] | #x0B3D | [#x0B5C-#x0B5D] | [#x0B5F-
                     #x0B61] | [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-
                     #x0B95] | [#x0B99-#x0B9A] | #x0B9C | [#x0B9E-#x0B9F]
                      | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA] | [#x0BAE-#x0BB5]
                      | [#x0BB7-#x0BB9] | [#x0C05-#x0C0C] | [#x0C0E-#x0C10]
                      | [#x0C12-#x0C28] | [#x0C2A-#x0C33] | [#x0C35-#x0C39]
                      | [#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90]
                      | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3] | [#x0CB5-#x0CB9]
                      | #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C] | [#x0D0E-
                     #x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39] | [#x0D60-
                     #x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33]
                      | [#x0E40-#x0E45] | [#x0E81-#x0E82] | #x0E84 | [#x0E87-
                     #x0E88] | #x0E8A | #x0E8D | [#x0E94-#x0E97] | [#x0E99-
                     #x0E9F] | [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7 | [#x0EAA-
                     #x0EAB] | [#x0EAD-#x0EAE] | #x0EB0 | [#x0EB2-#x0EB3]
                      | #x0EBD | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-
                     #x0F69] | [#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100
                      | [#x1102-#x1103] | [#x1105-#x1107] | #x1109 | [#x110B-
                     #x110C] | [#x110E-#x1112] | #x113C | #x113E | #x1140
                      | #x114C | #x114E | #x1150 | [#x1154-#x1155] | #x1159
                      | [#x115F-#x1161] | #x1163 | #x1165 | #x1167 | #x1169
                      | [#x116D-#x116E] | [#x1172-#x1173] | #x1175 | #x119E
                      | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8]
                      | #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9
                      | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15]
                      | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D]
                      | [#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D | [#x1F5F-
                     #x1F7D] | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE
                      | [#x1FC2-#x1FC4] | [#x1FC6-#x1FCC] | [#x1FD0-#x1FD3]
                      | [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC] | [#x1FF2-#x1FF4]
                      | [#x1FF6-#x1FFC] | #x2126 | [#x212A-#x212B] | #x212E
```

```
                      | [#x2180-#x2182] | [#x3041-#x3094] | [#x30A1-#x30FA]
                      | [#x3105-#x312C] | [#xAC00-#xD7A3]
[86]   Ideographic :: [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]
                   =
[87] CombiningChar :: [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486]
                   =  | [#x0591-#x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD]
                      | #x05BF | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652]
                      | #x0670 | [#x06D6-#x06DC] | [#x06DD-#x06DF] | [#x06E0-
                      #x06E4] | [#x06E7-#x06E8] | [#x06EA-#x06ED] | [#x0901-
                      #x0903] | #x093C | [#x093E-#x094C] | #x094D | [#x0951-
                      #x0954] | [#x0962-#x0963] | [#x0981-#x0983] | #x09BC
                      | #x09BE | #x09BF | [#x09C0-#x09C4] | [#x09C7-#x09C8]
                      | [#x09CB-#x09CD] | #x09D7 | [#x09E2-#x09E3] | #x0A02
                      | #x0A3C | #x0A3E | #x0A3F | [#x0A40-#x0A42] | [#x0A47-
                      #x0A48] | [#x0A4B-#x0A4D] | [#x0A70-#x0A71] | [#x0A81-
                      #x0A83] | #x0ABC | [#x0ABE-#x0AC5] | [#x0AC7-#x0AC9]
                      | [#x0ACB-#x0ACD] | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-
                      #x0B43] | [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-
                      #x0B57] | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-
                      #x0BC8] | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03]
                      | [#x0C3E-#x0C44] | [#x0C46-#x0C48] | [#x0C4A-#x0C4D]
                      | [#x0C55-#x0C56] | [#x0C82-#x0C83] | [#x0CBE-#x0CC4]
                      | [#x0CC6-#x0CC8] | [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6]
                      | [#x0D02-#x0D03] | [#x0D3E-#x0D43] | [#x0D46-#x0D48]
                      | [#x0D4A-#x0D4D] | #x0D57 | #x0E31 | [#x0E34-#x0E3A]
                      | [#x0E47-#x0E4E] | #x0EB1 | [#x0EB4-#x0EB9] | [#x0EBB-
                      #x0EBC] | [#x0EC8-#x0ECD] | [#x0F18-#x0F19] | #x0F35
                      | #x0F37 | #x0F39 | #x0F3E | #x0F3F | [#x0F71-#x0F84]
                      | [#x0F86-#x0F8B] | [#x0F90-#x0F95] | #x0F97 | [#x0F99-
                      #x0FAD] | [#x0FB1-#x0FB7] | #x0FB9 | [#x20D0-#x20DC]
                      | #x20E1 | [#x302A-#x302F] | #x3099 | #x309A
[88]         Digit :: [#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9]
                   =  | [#x0966-#x096F] | [#x09E6-#x09EF] | [#x0A66-#x0A6F]
                      | [#x0AE6-#x0AEF] | [#x0B66-#x0B6F] | [#x0BE7-#x0BEF]
                      | [#x0C66-#x0C6F] | [#x0CE6-#x0CEF] | [#x0D66-#x0D6F]
                      | [#x0E50-#x0E59] | [#x0ED0-#x0ED9] | [#x0F20-#x0F29]
[89]      Extender :: #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46
                   =  | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E]
                      | [#x30FC-#x30FE]
```

The character classes defined here can be derived from the Unicode character database as follows:

- Name start characters must have one of the categories Ll, Lu, Lo, Lt, Nl.
- Name characters other than Name-start characters must have one of the categories Mc, Me, Mn, Lm, or Nd.
- Characters in the compatibility area (i.e. with character code greater than #xF900 and less than #xFFFE) are not allowed in XML names.
- Characters which have a font or compatibility decomposition (i.e. those with a "compatibility formatting tag" in field 5 of the database -- marked by field 5 beginning with a "<") are not allowed.
- The following characters are treated as name-start characters rather than name characters, because the property file classifies them as Alphabetic: [#x02BB-#x02C1], #x0559, #x06E5, #x06E6.
- Characters #x20DD-#x20E0 are excluded (in accordance with Unicode, section 5.14).
- Character #x00B7 is classified as an extender, because the property list so identifies it.
- Character #x0387 is added as a name character, because #x00B7 is its canonical equivalent.
- Characters ':' and '_' are allowed as name-start characters.
- Characters '-' and '.' are allowed as name characters.

C. XML and SGML (Non-Normative)

XML is designed to be a subset of SGML, in that every valid XML document should also be a conformant SGML document. For a detailed comparison of the additional restrictions that XML places on documents beyond those of SGML, see [Clark].

D. Expansion of Entity and Character References (Non-Normative)

This appendix contains some examples illustrating the sequence of entity- and character-reference recognition and expansion, as specified in "4.4 XML Processor Treatment of Entities and References".

If the DTD contains the declaration

```
<!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>" >
```

then the XML processor will recognize the character references when it parses the entity declaration, and resolve them before storing the following string as the value of the entity "example":

```
<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>
```

A reference in the document to "&example;" will cause the text to be reparsed, at which time the start- and end-tags of the "p" element will be recognized and the three references will be recognized and expanded, resulting in a "p" element with the following content (all data, no delimiters or markup):

```
An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;).
```

A more complex example will illustrate the rules and their effects fully. In the following example, the line numbers are solely for reference.

```
1 <?xml version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60;!ENTITY tricky "error-prone" >' >
6 %xx;
7 ]>
8 <test>This sample shows a &tricky; method.</test>
```

This produces the following:

- in line 4, the reference to character 37 is expanded immediately, and the parameter entity "xx" is stored in the symbol table with the value "%zz;". Since the replacement text is not rescanned, the reference to parameter entity "zz" is not recognized. (And it would be an error if it were, since "zz" is not yet declared.)
- in line 5, the character reference "<" is expanded immediately and the parameter entity "zz" is stored with the replacement text "<!ENTITY tricky "error-prone" >", which is a well-formed entity declaration.
- in line 6, the reference to "xx" is recognized, and the replacement text of "xx" (namely "%zz;") is parsed. The reference to "zz" is recognized in its turn, and its replacement text ("<!ENTITY tricky "error-prone" >") is parsed. The general entity "tricky" has now been declared, with the replacement text "error-prone".
- in line 8, the reference to the general entity "tricky" is recognized, and it is expanded, so the full content of the "test" element is the self-describing (and ungrammatical) string *This sample shows a error-prone method.*

E. Deterministic Content Models (Non-Normative)

For compatibility, it is required that content models in element type declarations be deterministic.

SGML requires deterministic content models (it calls them "unambiguous"); XML processors built using SGML systems may flag non-deterministic content models as errors.

For example, the content model `((b, c) | (b, d))` is non-deterministic, because given an initial `b` the parser cannot know which `b` in the model is being matched without looking ahead to see which element follows the `b`. In this case, the two references to `b` can be collapsed into a single reference, making the model read `(b, (c | d))`. An initial `b` now clearly matches only a single name in the content model. The parser doesn't need to look ahead to see what follows; either `c` or `d` would be accepted.
More formally: a finite state automaton may be constructed from the content model using the standard algorithms, e. g. algorithm 3.5 in section 3.9 of Aho, Sethi, and Ullman [Aho/Ullman]. In many such algorithms, a follow set is constructed for each position in the regular expression (i.e., each leaf node in the syntax tree for the regular expression); if any position has a follow set in which more than one following position is labeled with the same element type name, then the content model is in error and may be reported as an error.
Algorithms exist which allow many but not all non-deterministic content models to be reduced automatically to equivalent deterministic models; see Brüggemann-Klein 1991 [Brüggemann-Klein].

F. Autodetection of Character Encodings (Non-Normative)

The XML encoding declaration functions as an internal label on each entity, indicating which character encoding is in use. Before an XML processor can read the internal label, however, it apparently has to know what character encoding is in use--which is what the internal label is trying to indicate. In the general case, this is a hopeless situation. It is not entirely hopeless in XML, however, because XML limits the general case in two ways: each implementation is assumed to support only a finite set of character encodings, and the XML encoding declaration is restricted in position and content in order to make it feasible to autodetect the character encoding in use in each entity in normal cases. Also, in many cases other sources of information are available in addition to the XML data stream itself. Two cases may be distinguished, depending on whether the XML entity is presented to the processor without, or with, any accompanying (external) information. We consider the first case first.
Because each XML entity not in UTF-8 or UTF-16 format *must* begin with an XML encoding declaration, in which the first characters must be '`<?xml`', any conforming processor can detect, after two to four octets of input, which of the following cases apply. In reading this list, it may help to know that in UCS-4, '<' is "`#x0000003C`" and '?' is "`#x0000003F`", and the Byte Order Mark required of UTF-16 data streams is "`#xFEFF`".

- `00 00 00 3C`: UCS-4, big-endian machine (1234 order)
- `3C 00 00 00`: UCS-4, little-endian machine (4321 order)
- `00 00 3C 00`: UCS-4, unusual octet order (2143)
- `00 3C 00 00`: UCS-4, unusual octet order (3412)
- `FE FF`: UTF-16, big-endian
- `FF FE`: UTF-16, little-endian
- `00 3C 00 3F`: UTF-16, big-endian, no Byte Order Mark (and thus, strictly speaking, in error)
- `3C 00 3F 00`: UTF-16, little-endian, no Byte Order Mark (and thus, strictly speaking, in error)
- `3C 3F 78 6D`: UTF-8, ISO 646, ASCII, some part of ISO 8859, Shift-JIS, EUC, or any other 7-bit, 8-bit, or mixed-width encoding which ensures that the characters of ASCII have their normal positions, width, and values; the actual encoding declaration must be read to detect which of these applies, but since all of these encodings use the same bit patterns for the ASCII characters, the encoding declaration itself may be read reliably
- `4C 6F A7 94`: EBCDIC (in some flavor; the full encoding declaration must be read to tell which code page is in use)
- other: UTF-8 without an encoding declaration, or else the data stream is corrupt, fragmentary, or enclosed in a wrapper of some kind

This level of autodetection is enough to read the XML encoding declaration and parse the character-encoding identifier, which is still necessary to distinguish the individual members of each family of encodings (e.g. to tell UTF-8 from 8859, and the parts of 8859 from each other, or to distinguish the specific EBCDIC code page in use, and so on).
Because the contents of the encoding declaration are restricted to ASCII characters, a processor can reliably read the entire encoding declaration as soon as it has detected which family of encodings is in use. Since in practice, all widely used character encodings fall into one of the categories above, the XML encoding declaration allows reasonably reliable in-band labeling of character encodings, even when external sources of information at the operating-system or transport-protocol level are unreliable.
Once the processor has detected the character encoding in use, it can act appropriately, whether by invoking a separate input routine for each case, or by calling the proper conversion function on each character of input.
Like any self-labeling system, the XML encoding declaration will not work if any software changes the entity's

character set or encoding without updating the encoding declaration. Implementors of character-encoding routines should be careful to ensure the accuracy of the internal and external information used to label the entity.
The second possible case occurs when the XML entity is accompanied by encoding information, as in some file systems and some network protocols. When multiple sources of information are available, their relative priority and the preferred method of handling conflict should be specified as part of the higher-level protocol used to deliver XML. Rules for the relative priority of the internal label and the MIME-type label in an external header, for example, should be part of the RFC document defining the text/xml and application/xml MIME types. In the interests of interoperability, however, the following rules are recommended.

- If an XML entity is in a file, the Byte-Order Mark and encoding-declaration PI are used (if present) to determine the character encoding. All other heuristics and sources of information are solely for error recovery.
- If an XML entity is delivered with a MIME type of text/xml, then the `charset` parameter on the MIME type determines the character encoding method; all other heuristics and sources of information are solely for error recovery.
- If an XML entity is delivered with a MIME type of application/xml, then the Byte-Order Mark and encoding-declaration PI are used (if present) to determine the character encoding. All other heuristics and sources of information are solely for error recovery.

These rules apply only in the absence of protocol-level documentation; in particular, when the MIME types text/xml and application/xml are defined, the recommendations of the relevant RFC will supersede these rules.

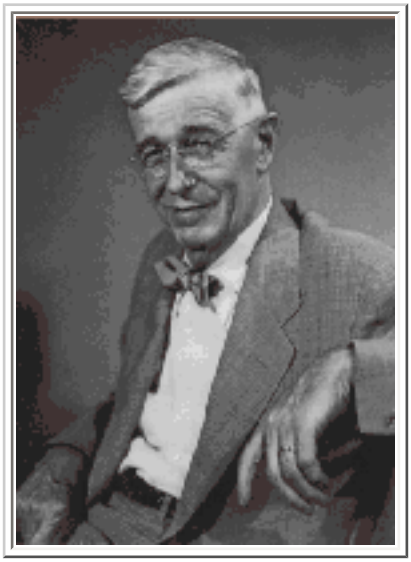
G. W3C XML Working Group (Non-Normative)

This specification was prepared and approved for publication by the W3C XML Working Group (WG). WG approval of this specification does not necessarily imply that all WG members voted for its approval. The current and former members of the XML WG are:
Jon Bosak, Sun (Chair); James Clark (Technical Lead); Tim Bray, Textuality and Netscape (XML Co-editor); Jean Paoli, Microsoft (XML Co-editor); C. M. Sperberg-McQueen, U. of Ill. (XML Co-editor); Dan Connolly, W3C (W3C Liaison); Paula Angerstein, Texcel; Steve DeRose, INSO; Dave Hollander, HP; Eliot Kimber, ISOGEN; Eve Maler, ArborText; Tom Magliery, NCSA; Murray Maloney, Muzmo and Grif; Makoto Murata, Fuji Xerox Information Systems; Joel Nava, Adobe; Conleth O'Connell, Vignette; Peter Sharpe, SoftQuad; John Tigue, DataChannel

Top

As We May Think

by **Vannevar Bush**



This article was originally published in the **July 1945** issue of **The Atlantic Monthly**. It is reproduced here **with** their permission.

HTML version by Denys Duchier, University of Ottawa, April 1994. Updated August 1995, Simon Fraser University. - please email comments and corrections to duchier@cs.sfu.ca - an ASCII version is also available - both have been donated to Project Gutenberg.

As Director of the Office of Scientific Research and Development, Dr. Vannevar Bush has coördinated the activities of some six thousand leading American scientists in the application of science to warfare. In this significant article he holds up an incentive for scientists when the fighting has ceased. He urges that men of science should then turn to the massive task of making more accessible our bewildering store of knowledge. For many years inventions have extended man's physical powers rather than the powers of his mind. Trip hammers that multiply the fists, microscopes that sharpen the eye, and engines of destruction and detection are new results, but the end results, of modern science. Now, says Dr. Bush, instruments are at hand which, if properly developed, will give man access to and command over the inherited knowledge of the ages. The perfection of these pacific instruments should be the first objective of our scientists as

they emerge from their war work. Like Emerson's famous address of 1837 on ``The American Scholar,'' this paper by Dr. Bush calls for a new relationship between thinking man and the sum of our knowledge.
- The Editor

This has not been a scientist's war; it has been a war in which all have had a part. The scientists, burying their old professional competition in the demand of a common cause, have shared greatly and learned much. It has been exhilarating to work in effective partnership. Now, for many, this appears to be approaching an end. What are the scientists to do next?

For the biologists, and particularly for the medical scientists, there can be little indecision, for their war work has hardly required them to leave the old paths. Many indeed have been able to carry on their war research in their familiar peacetime laboratories. Their objectives remain much the same.

It is the physicists who have been thrown most violently off stride, who have left academic pursuits for the making of strange destructive gadgets, who have had to devise new methods for their unanticipated assignments. They have done their part on the devices that made it possible to turn back the enemy. They have worked in combined effort with the physicists of our allies. They have felt within themselves the stir of achievement. They have been part of a great team. Now, as peace approaches, one asks where they will find objectives worthy of their best.

1

Of what lasting benefit has been man's use of science and of the new instruments which his research brought into existence? First, they have increased his control of his material environment. They have improved his food, his clothing, his shelter; they have increased his security and released him partly from the bondage of bare existence. They have given him increased knowledge of his own biological processes so that he has had a progressive freedom from disease and an increased span of life. They are illuminating the interactions of his physiological and psychological functions, giving the promise of an improved mental health.

Science has provided the swiftest communication between individuals; it has provided a record of ideas and has enabled man to manipulate and to make extracts from that record so that knowledge evolves and endures throughout the life of a race rather than that of an individual.

There is a growing mountain of research. But there is increased evidence that we are being bogged down today as specialization extends. The investigator is staggered by the findings and conclusions of thousands of other workers - conclusions which he cannot find time to grasp, much less to remember, as they appear. Yet specialization becomes increasingly necessary for progress, and the effort to bridge between disciplines is correspondingly superficial.

Professionally our methods of transmitting and reviewing the results of research are generations old

and by now are totally inadequate for their purpose. If the aggregate time spent in writing scholarly works and in reading them could be evaluated, the ratio between these amounts of time might well be startling. Those who conscientiously attempt to keep abreast of current thought, even in restricted fields, by close and continuous reading might well shy away from an examination calculated to show how much of the previous month's efforts could be produced on call. Mendel's concept of the laws of genetics was lost to the world for a generation because his publication did not reach the few who were capable of grasping and extending it; and this sort of catastrophe is undoubtedly being repeated all about us, as truly significant attainments become lost in the mass of the inconsequential.

The difficulty seems to be, not so much that we publish unduly in view of the extent and variety of present-day interests, but rather that publication has been extended far beyond our present ability to make real use of the record. The summation of human experience is being expanded at a prodigious rate, and the means we use for threading through the consequent maze to the momentarily important item is the same as was used in the days of square-rigged ships.

But there are signs of a change as new and powerful instrumentalities come into use. Photocells capable of seeing things in a physical sense, advanced photography which can record what is seen or even what is not, thermionic tubes capable of controlling potent forces under the guidance of less power than a mosquito uses to vibrate his wings, cathode ray tubes rendering visible an occurrence so brief that by comparison a microsecond is a long time, relay combinations which will carry out involved sequences of movements more reliably than any human operator and thousand of times as fast - there are plenty of mechanical aids with which to effect a transformation in scientific records.

Two centuries ago Leibnitz invented a calculating machine which embodied most of the essential features of recent keyboard devices, but it could not then come into use. The economics of the situation were against it: the labor involved in constructing it, before the days of mass production, exceeded the labor to be saved by its use, since all it could accomplish could be duplicated by sufficient use of pencil and paper. Moreover, it would have been subject to frequent breakdown, so that it could not have been depended upon; for at that time and long after, complexity and unreliability were synonymous.

Babbage, even with remarkably generous support for his time, could not produce his great arithmetical machine. His idea was sound enough, but construction and maintenance costs were then too heavy. Had a Pharaoh been given detailed and explicit designs of an automobile, and had he understood them completely, it would have taxed the resources of his kingdom to have fashioned the thousands of parts for a single car, and that car would have broken down on the first trip to Giza.

Machines with interchangeable parts can now be constructed with great economy of effort. In spite of much complexity, they perform reliably. Witness the humble typewriter, or the movie camera, or the automobile. Electrical contacts have ceased to stick when thoroughly understood. Note the automatic telephone exchange, which has hundreds of thousands of such contacts, and yet is reliable. A spider web of metal, sealed in a thin glass container, a wire heated to brilliant glow, in short, the thermionic tube of radio sets, is made by the hundred million, tossed about in packages, plugged into sockets - and it works! Its gossamer parts, the precise location and alignment involved in its construction, would have occupied a master craftsman of the guild for months; now it is built for thirty cents. The

world has arrived at an age of cheap complex devices of great reliability; and something is bound to come of it.

2

A record, if it is to be useful to science, must be continuously extended, it must be stored, and above all it must be consulted. Today we make the record conventionally by writing and photography, followed by printing; but we also record on film, on wax disks, and on magnetic wires. Even if utterly new recording procedures do not appear, these present ones are certainly in the process of modification and extension.

Certainly progress in photography is not going to stop. Faster material and lenses, more automatic cameras, finer-grained sensitive compounds to allow an extension of the minicamera idea, are all imminent. Let us project this trend ahead to a logical, if not inevitable, outcome. The camera hound of the future wears on his forehead a lump a little larger than a walnut. It takes pictures 3 millimeters square, later to be projected or enlarged, which after all involves only a factor of 10 beyond present practice. The lens is of universal focus, down to any distance accommodated by the unaided eye, simply because it is of short focal length. There is a built-in photocell on the walnut such as we now have on at least one camera, which automatically adjusts exposure for a wide range of illumination. There is film in the walnut for a hundred exposures, and the spring for operating its shutter and shifting its film is wound once for all when the film clip is inserted. It produces its result in full color. It may well be stereoscopic, and record with spaced glass eyes, for striking improvements in stereoscopic technique are just around the corner.

The cord which trips its shutter may reach down a man's sleeve within easy reach of his fingers. A quick squeeze, and the picture is taken. On a pair of ordinary glasses is a square of fine lines near the top of one lens, where it is out of the way of ordinary vision. When an object appears in that square, it is lined up for its picture. As the scientist of the future moves about the laboratory or the field, every time he looks at something worthy of the record, he trips the shutter and in it goes, without even an audible click. Is this all fantastic? The only fantastic thing about it is the idea of making as many pictures as would result from its use.

Will there be dry photography? It is already here in two forms. When Brady made his Civil War pictures, the plate had to be wet at the time of exposure. Now it has to be wet during development instead. In the future perhaps it need not be wetted at all. There have long been films impregnated with diazo dyes which form a picture without development, so that it is already there as soon as the camera has been operated. An exposure to ammonia gas destroys the unexposed dye, and the picture can then be taken out into the light and examined. The process is now slow, but someone may speed it up, and it has no grain difficulties such as now keep photographic researchers busy. Often it would be advantageous to be able to snap the camera and to look at the picture immediately.

Another process now in use is also slow, and more or less clumsy. For fifty years impregnated papers have been used which turn dark at every point where an electrical contact touches them, by reason of the chemical change thus produced in an iodine compound included in the paper. They have been

used to make records, for a pointer moving across them can leave a trail behind. If the electrical potential on the pointer is varied as it moves, the line becomes light or dark in accordance with the potential.

This scheme is now used in facsimile transmission. The pointer draws a set of closely spaced lines across the paper one after another. As it moves, its potential is varied in accordance with a varying current received over wires from a distant station, where these variations are produced by a photocell which is similarly scanning a picture. At every instant the darkness of the line being drawn is made equal to the darkness of the point on the picture being observed by the photocell. Thus, when the whole picture has been covered, a replica appears at the receiving end.

A scene itself can be just as well looked over line by line by the photocell in this way as can a photograph of the scene. This whole apparatus constitutes a camera, with the added feature, which can be dispensed with if desired, of making its picture at a distance. It is slow, and the picture is poor in detail. Still, it does give another process of dry photography, in which the picture is finished as soon as it is taken.

It would be a brave man who could predict that such a process will always remain clumsy, slow, and faulty in detail. Television equipment today transmits sixteen reasonably good images a second, and it involves only two essential differences from the process described above. For one, the record is made by a moving beam of electrons rather than a moving pointer, for the reason that an electron beam can sweep across the picture very rapidly indeed. The other difference involves merely the use of a screen which glows momentarily when the electrons hit, rather than a chemically treated paper or film which is permanently altered. This speed is necessary in television, for motion pictures rather than stills are the object.

Use chemically treated film in place of the glowing screen, allow the apparatus to transmit one picture rather than a succession, and a rapid camera for dry photography results. The treated film needs to be far faster in action than present examples, but it probably could be. More serious is the objection that this scheme would involve putting the film inside a vacuum chamber, for electron beams behave normally only in such a rarefied environment. This difficulty could be avoided by allowing the electron beam to play on one side of a partition, and by pressing the film against the other side, if this partition were such as to allow the electrons to go through perpendicular to its surface, and to prevent them from spreading out sideways. Such partitions, in crude form, could certainly be constructed, and they will hardly hold up the general development.

Like dry photography, microphotography still has a long way to go. The basic scheme of reducing the size of the record, and examining it by projection rather than directly, has possibilities too great to be ignored. The combination of optical projection and photographic reduction is already producing some results in microfilm for scholarly purposes, and the potentialities are highly suggestive. Today, with microfilm, reductions by a linear factor of 20 can be employed and still produce full clarity when the material is re-enlarged for examination. The limits are set by the graininess of the film, the excellence of the optical system, and the efficiency of the light sources employed. All of these are rapidly improving.

Assume a linear ratio of 100 for future use. Consider film of the same thickness as paper, although thinner film will certainly be usable. Even under these conditions there would be a total factor of 10,000 between the bulk of the ordinary record on books, and its microfilm replica. The *Encyclopoedia Britannica* could be reduced to the volume of a matchbox. A library of a million volumes could be compressed into one end of a desk. If the human race has produced since the invention of movable type a total record, in the form of magazines, newspapers, books, tracts, advertising blurbs, correspondence, having a volume corresponding to a billion books, the whole affair, assembled and compressed, could be lugged off in a moving van. Mere compression, of course, is not enough; one needs not only to make and store a record but also to be able to consult it, and this aspect of the matter comes later. Even the modern great library is not generally consulted; it is nibbled by a few.

Compression is important, however, when it comes to costs. The material for the microfilm *Britannica* would cost a nickel, and it could be mailed anywhere for a cent. What would it cost to print a million copies? To print a sheet of newspaper, in a large edition, costs a small fraction of a cent. The entire material of the *Britannica* in reduced microfilm form would go on a sheet eight and one-half by eleven inches. Once it is available, with the photographic reproduction methods of the future, duplicates in large quantities could probably be turned out for a cent apiece beyond the cost of materials. The preparation of the original copy? That introduces the next aspect of the subject.

3

To make the record, we now push a pencil or tap a typewriter. Then comes the process of digestion and correction, followed by an intricate process of typesetting, printing, and distribution. To consider the first stage of the procedure, will the author of the future cease writing by hand or typewriter and talk directly to the record? He does so indirectly, by talking to a stenographer or a wax cylinder; but the elements are all present if he wishes to have his talk directly produce a typed record. All he needs to do is to take advantage of existing mechanisms and to alter his language.

At a recent World Fair a machine called a Voder was shown. A girl stroked its keys and it emitted recognizable speech. No human vocal cords entered in the procedure at any point; the keys simply combined some electrically produced vibrations and passed these on to a loud-speaker. In the Bell Laboratories there is the converse of this machine, called a Vocoder. The loudspeaker is replaced by a microphone, which picks up sound. Speak to it, and the corresponding keys move. This may be one element of the postulated system.

The other element is found in the stenotype, that somewhat disconcerting device encountered usually at public meetings. A girl strokes its keys languidly and looks about the room and sometimes at the speaker with a disquieting gaze. From it emerges a typed strip which records in a phonetically simplified language a record of what the speaker is supposed to have said. Later this strip is retyped into ordinary language, for in its nascent form it is intelligible only to the initiated. Combine these two elements, let the Vocoder run the stenotype, and the result is a machine which types when talked to.

Our present languages are not especially adapted to this sort of mechanization, it is true. It is strange that the inventors of universal languages have not seized upon the idea of producing one which better fitted the technique for transmitting and recording speech. Mechanization may yet force the issue, especially in the scientific field; whereupon scientific jargon would become still less intelligible to the layman.

One can now picture a future investigator in his laboratory. His hands are free, and he is not anchored. As he moves about and observes, he photographs and comments. Time is automatically recorded to tie the two records together. If he goes into the field, he may be connected by radio to his recorder. As he ponders over his notes in the evening, he again talks his comments into the record. His typed record, as well as his photographs, may both be in miniature, so that he projects them for examination.

Much needs to occur, however, between the collection of data and observations, the extraction of parallel material from the existing record, and the final insertion of new material into the general body of the common record. For mature thought there is no mechanical substitute. But creative thought and essentially repetitive thought are very different things. For the latter there are, and may be, powerful mechanical aids.

Adding a column of figures is a repetitive thought process, and it was long ago properly relegated to the machine. True, the machine is sometimes controlled by the keyboard, and thought of a sort enters in reading the figures and poking the corresponding keys, but even this is avoidable. Machines have been made which will read typed figures by photocells and then depress the corresponding keys; these are combinations of photocells for scanning the type, electric circuits for sorting the consequent variations, and relay circuits for interpreting the result into the action of solenoids to pull the keys down.

All this complication is needed because of the clumsy way in which we have learned to write figures. If we recorded them positionally, simply by the configuration of a set of dots on a card, the automatic reading mechanism would become comparatively simple. In fact, if the dots are holes, we have the punched-card machine long ago produced by Hollorith for the purposes of the census, and now used throughout business. Some types of complex businesses could hardly operate without these machines.

Adding is only one operation. To perform arithmetical computation involves also subtraction, multiplication, and division, and in addition some method for temporary storage of results, removal from storage for further manipulation, and recording of final results by printing. Machines for these purposes are now of two types: keyboard machines for accounting and the like, manually controlled for the insertion of data, and usually automatically controlled as far as the sequence of operations is concerned; and punched-card machines in which separate operations are usually delegated to a series of machines, and the cards then transferred bodily from one to another. Both forms are very useful; but as far as complex computations are concerned, both are still embryo.

Rapid electrical counting appeared soon after the physicists found it desirable to count cosmic rays. For their own purposes the physicists promptly constructed thermionic-tube equipment capable of counting electrical impulses at the rate of 100,000 a second. The advanced arithmetical machines of

the future will be electrical in nature, and they will perform at 100 times present speeds, or more.

Moreover, they will be far more versatile than present commercial machines, so that they may readily be adapted for a wide variety of operations. They will be controlled by a control card or film, they will select their own data and manipulate it in accordance with the instructions thus inserted, they will perform complex arithmetical computations at exceedingly high speeds, and they will record results in such form as to be readily available for distribution or for later further manipulation. Such machines will have enormous appetites. One of them will take instructions and data from a roomful of girls armed with simple keyboard punches, and will deliver sheets of computed results every few minutes. There will always be plenty of things to compute in the detailed affairs of millions of people doing complicated things.

4

The repetitive processes of thought are not confined, however, to matters of arithmetic and statistics. In fact, every time one combines and records facts in accordance with established logical processes, the creative aspect of thinking is concerned only with the selection of the data and the process to be employed, and the manipulation thereafter is repetitive in nature and hence a fit matter to be relegated to the machines. Not so much has been done along these lines, beyond the bounds of arithmetic, as might be done, primarily because of the economics of the situation. The needs of business, and the extensive market obviously waiting, assured the advent of mass-produced arithmetical machines just as soon as production methods were sufficiently advanced.

With machines for advanced analysis no such situation existed; for there was and is no extensive market; the users of advanced methods of manipulating data are a very small part of the population. There are, however, machines for solving differential equations - and functional and integral equations, for that matter. There are many special machines, such as the harmonic synthesizer which predicts the tides. There will be many more, appearing certainly first in the hands of the scientist and in small numbers.

If scientific reasoning were limited to the logical processes of arithmetic, we should not get far in our understanding of the physical world. One might as well attempt to grasp the game of poker entirely by the use of the mathematics of probability. The abacus, with its beads string on parallel wires, led the Arabs to positional numeration and the concept of zero many centuries before the rest of the world; and it was a useful tool - so useful that it still exists.

It is a far cry from the abacus to the modern keyboard accounting machine. It will be an equal step to the arithmetical machine of the future. But even this new machine will not take the scientist where he needs to go. Relief must be secured from laborious detailed manipulation of higher mathematics as well, if the users of it are to free their brains for something more than repetitive detailed transformations in accordance with established rules. A mathematician is not a man who can readily manipulate figures; often he cannot. He is not even a man who can readily perform the transformation of equations by the use of calculus. He is primarily an individual who is skilled in the use of symbolic logic on a high plane, and especially he is a man of intuitive judgment in the choice of the manipulative processes he employs.

All else he should be able to turn over to his mechanism, just as confidently as he turns over the propelling of his car to the intricate mechanism under the hood. Only then will mathematics be practically effective in bringing the growing knowledge of atomistics to the useful solution of the advanced problems of chemistry, metallurgy, and biology. For this reason there will come more machines to handle advanced mathematics for the scientist. Some of them will be sufficiently bizarre to suit the most fastidious connoisseur of the present artifacts of civilization.

5

The scientist, however, is not the only person who manipulates data and examines the world about him by the use of logical processes, although he sometimes preserves this appearance by adopting into the fold anyone who becomes logical, much in the manner in which a British labor leader is elevated to knighthood. Whenever logical processes of thought are employed - that is, whenever thought for a time runs along an accepted groove - there is an opportunity for the machine. Formal logic used to be a keen instrument in the hands of the teacher in his trying of students' souls. It is readily possible to construct a machine which will manipulate premises in accordance with formal logic, simply by the clever use of relay circuits. Put a set of premises into such a device and turn the crank, and it will readily pass out conclusion after conclusion, all in accordance with logical law, and with no more slips than would be expected of a keyboard adding machine.

Logic can become enormously difficult, and it would undoubtedly be well to produce more assurance in its use. The machines for higher analysis have usually been equation solvers. Ideas are beginning to appear for equation transformers, which will rearrange the relationship expressed by an equation in accordance with strict and rather advanced logic. Progress is inhibited by the exceedingly crude way in which mathematicians express their relationships. They employ a symbolism which grew like Topsy and has little consistency; a strange fact in that most logical field.

A new symbolism, probably positional, must apparently precede the reduction of mathematical transformations to machine processes. Then, on beyond the strict logic of the mathematician, lies the application of logic in everyday affairs. We may some day click off arguments on a machine with the same assurance that we now enter sales on a cash register. But the machine of logic will not look like a cash register, even a streamlined model.

So much for the manipulation of ideas and their insertion into the record. Thus far we seem to be worse off than before - for we can enormously extend the record; yet even in its present bulk we can hardly consult it. This is a much larger matter than merely the extraction of data for the purposes of scientific research; it involves the entire process by which man profits by his inheritance of acquired knowledge. The prime action of use is selection, and here we are halting indeed. There may be millions of fine thoughts, and the account of the experience on which they are based, all encased within stone walls of acceptable architectural form; but if the scholar can get at only one a week by diligent search, his syntheses are not likely to keep up with the current scene.

Selection, in this broad sense, is a stone adze in the hands of a cabinetmaker. Yet, in a narrow sense

and in other areas, something has already been done mechanically on selection. The personnel officer of a factory drops a stack of a few thousand employee cards into a selecting machine, sets a code in accordance with an established convention, and produces in a short time a list of all employees who live in Trenton and know Spanish. Even such devices are much too slow when it comes, for example, to matching a set of fingerprints with one of five millions on file. Selection devices of this sort will soon be speeded up from their present rate of reviewing data at a few hundred a minute. By the use of photocells and microfilm they will survey items at the rate of thousands a second, and will print out duplicates of those selected.

This process, however, is simple selection: it proceeds by examining in turn every one of a large set of items, and by picking out those which have certain specified characteristics. There is another form of selection best illustrated by the automatic telephone exchange. You dial a number and the machine selects and connects just one of a million possible stations. It does not run over them all. It pays attention only to a class given by a first digit, and so on; and thus proceeds rapidly and almost unerringly to the selected station. It requires a few seconds to make the selection, although the process could be speeded up if increased speed were economically warranted. If necessary, it could be made extremely fast by substituting thermionic-tube switching for mechanical switching, so that the full selection could be made in one-hundredth of a second. No one would wish to spend the money necessary to make this change in the telephone system, but the general idea is applicable elsewhere.

Take the prosaic problem of the great department store. Every time a charge sale is made, there are a number of things to be done.. The inventory needs to be revised, the salesman needs to be given credit for the sale, the general accounts need an entry, and, most important, the customer needs to be charged. A central records device has been developed in which much of this work is done conveniently. The salesman places on a stand the customer's identification card, his own card, and the card taken from the article sold - all punched cards. When he pulls a lever, contacts are made through the holes, machinery at a central point makes the necessary computations and entries, and the proper receipt is printed for the salesman to pass to the customer.

But there may be ten thousand charge customers doing business with the store, and before the full operation can be completed someone has to select the right card and insert it at the central office. Now rapid selection can slide just the proper card into position in an instant or two, and return it afterward. Another difficulty occurs, however. Someone must read a total on the card, so that the machine can add its computed item to it. Conceivably the cards might be of the dry photography type I have described. Existing totals could then be read by photocell, and the new total entered by an electron beam.

The cards may be in miniature, so that they occupy little space. They must move quickly. They need not be transferred far, but merely into position so that the photocell and recorder can operate on them. Positional dots can enter the data. At the end of the month a machine can readily be made to read these and to print an ordinary bill. With tube selection, in which no mechanical parts are involved in the switches, little time need be occupied in bringing the correct card into use - a second should suffice for the entire operation. The whole record on the card may be made by magnetic dots on a steel sheet if desired, instead of dots to be observed optically, following the scheme by which Poulsen long ago put speech on a magnetic wire. This method has the advantage of simplicity and ease of

erasure. By using photography, however, one can arrange to project the record in enlarged form, and at a distance by using the process common in television equipment.

One can consider rapid selection of this form, and distant projection for other purposes. To be able to key one sheet of a million before an operator in a second or two, with the possibility of then adding notes thereto, is suggestive in many ways. It might even be of use in libraries, but that is another story. At any rate, there are now some interesting combinations possible. One might, for example, speak to a microphone, in the manner described in connection with the speech-controlled typewriter, and thus make his selections. It would certainly beat the usual file clerk.

6

The real heart of the matter of selection, however, goes deeper than a lag in the adoption of mechanisms by libraries, or a lack of development of devices for their use. Our ineptitude in getting at the record is largely caused by the artificiality of systems of indexing. When data of any sort are placed in storage, they are filed alphabetically or numerically, and information is found (when it is) by tracing it down from subclass to subclass. It can be in only one place, unless duplicates are used; one has to have rules as to which path will locate it, and the rules are cumbersome. Having found one item, moreover, one has to emerge from the system and re-enter on a new path.

The human mind does not work that way. It operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain. It has other characteristics, of course; trails that are not frequently followed are prone to fade, items are not fully permanent, memory is transitory. Yet the speed of action, the intricacy of trails, the detail of mental pictures, is awe-inspiring beyond all else in nature.

Man cannot hope fully to duplicate this mental process artificially, but he certainly ought to be able to learn from it. In minor ways he may even improve, for his records have relative permanency. The first idea, however, to be drawn from the analogy concerns selection. Selection by association, rather than by indexing, may yet be mechanized. One cannot hope thus to equal the speed and flexibility with which the mind follows an associative trail, but it should be possible to beat the mind decisively in regard to the permanence and clarity of the items resurrected from storage.

Consider a future device for individual use, which is a sort of mechanized private file and library. It needs a name, and to coin one at random, ``memex'' will do. A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.

It consists of a desk, and while it can presumably be operated from a distance, it is primarily the piece of furniture at which he works. On the top are slanting translucent screens, on which material can be projected for convenient reading. There is a keyboard, and sets of buttons and levers. Otherwise it looks like an ordinary desk.

In one end is the stored material. The matter of bulk is well taken care of by improved microfilm. Only a small part of the interior of the memex is devoted to storage, the rest to mechanism. Yet if the user inserted 5000 pages of material a day it would take him hundreds of years to fill the repository, so he can be profligate and enter material freely.

Most of the memex contents are purchased on microfilm ready for insertion. Books of all sorts, pictures, current periodicals, newspapers, are thus obtained and dropped into place. Business correspondence takes the same path. And there is provision for direct entry. On the top of the memex is a transparent platen. On this are placed longhand notes, photographs, memoranda, all sort of things. When one is in place, the depression of a lever causes it to be photographed onto the next blank space in a section of the memex film, dry photography being employed.

There is, of course, provision for consultation of the record by the usual scheme of indexing. If the user wishes to consult a certain book, he taps its code on the keyboard, and the title page of the book promptly appears before him, projected onto one of his viewing positions. Frequently-used codes are mnemonic, so that he seldom consults his code book; but when he does, a single tap of a key projects it for his use. Moreover, he has supplemental levers. On deflecting one of these levers to the right he runs through the book before him, each page in turn being projected at a speed which just allows a recognizing glance at each. If he deflects it further to the right, he steps through the book 10 pages at a time; still further at 100 pages at a time. Deflection to the left gives him the same control backwards.

A special button transfers him immediately to the first page of the index. Any given book of his library can thus be called up and consulted with far greater facility than if it were taken from a shelf. As he has several projection positions, he can leave one item in position while he calls up another. He can add marginal notes and comments, taking advantage of one possible type of dry photography, and it could even be arranged so that he can do this by a stylus scheme, such as is now employed in the telautograph seen in railroad waiting rooms, just as though he had the physical page before him.

7

All this is conventional, except for the projection forward of present-day mechanisms and gadgetry. It affords an immediate step, however, to associative indexing, the basic idea of which is a provision whereby any item may be caused at will to select immediately and automatically another. This is the essential feature of the memex. The process of tying two items together is the important thing.

When the user is building a trail, he names it, inserts the name in his code book, and taps it out on his keyboard. Before him are the two items to be joined, projected onto adjacent viewing positions. At the bottom of each there are a number of blank code spaces, and a pointer is set to indicate one of these on each item. The user taps a single key, and the items are permanently joined. In each code space appears the code word. Out of view, but also in the code space, is inserted a set of dots for photocell viewing; and on each item these dots by their positions designate the index number of the other item.

Thereafter, at any time, when one of these items is in view, the other can be instantly recalled merely by tapping a button below the corresponding code space. Moreover, when numerous items have been thus joined together to form a trail, they can be reviewed in turn, rapidly or slowly, by deflecting a lever like that used for turning the pages of a book. It is exactly as though the physical items had been gathered together to form a new book. It is more than this, for any item can be joined into numerous trails.

The owner of the memex, let us say, is interested in the origin and properties of the bow and arrow. Specifically he is studying why the short Turkish bow was apparently superior to the English long bow in the skirmishes of the Crusades. He has dozens of possibly pertinent books and articles in his memex. First he runs through an encyclopedia, finds an interesting but sketchy article, leaves it projected. Next, in a history, he finds another pertinent item, and ties the two together. Thus he goes, building a trail of many items. Occasionally he inserts a comment of his own, either linking it into the main trail or joining it by a side trail to a particular item. When it becomes evident that the elastic properties of available materials had a great deal to do with the bow, he branches off on a side trail which takes him through textbooks on elasticity and tables of physical constants. He inserts a page of longhand analysis of his own. Thus he builds a trail of his interest through the maze of materials available to him.

And his trails do not fade. Several years later, his talk with a friend turns to the queer ways in which a people resist innovations, even of vital interest. He has an example, in the fact that the outranged Europeans still failed to adopt the Turkish bow. In fact he has a trail on it. A touch brings up the code book. Tapping a few keys projects the head of the trail. A lever runs through it at will, stopping at interesting items, going off on side excursions. It is an interesting trail, pertinent to the discussion. So he sets a reproducer in action, photographs the whole trail out, and passes it to his friend for insertion in his own memex, there to be linked into the more general trail.

8

Wholly new forms of encyclopedias will appear, ready-made with a mesh of associative trails running through them, ready to be dropped into the memex and there amplified. The lawyer has at his touch the associated opinions and decisions of his whole experience, and of the experience of friends and authorities. The patent attorney has on call the millions of issued patents, with familiar trails to every point of his client's interest. The physician, puzzled by its patient's reactions, strikes the trail established in studying an earlier similar case, and runs rapidly through analogous case histories, with side references to the classics for the pertinent anatomy and histology. The chemist, struggling with the synthesis of an organic compound, has all the chemical literature before him in his laboratory, with trails following the analogies of compounds, and side trails to their physical and chemical behavior.

The historian, with a vast chronological account of a people, parallels it with a skip trail which stops only at the salient items, and can follow at any time contemporary trails which lead him all over civilization at a particular epoch. There is a new profession of trail blazers, those who find delight in the task of establishing useful trails through the enormous mass of the common record. The

inheritance from the master becomes, not only his additions to the world's record, but for his disciples the entire scaffolding by which they were erected.

Thus science may implement the ways in which man produces, stores, and consults the record of the race. It might be striking to outline the instrumentalities of the future more spectacularly, rather than to stick closely to the methods and elements now known and undergoing rapid development, as has been done here. Technical difficulties of all sorts have been ignored, certainly, but also ignored are means as yet unknown which may come any day to accelerate technical progress as violently as did the advent of the thermionic tube. In order that the picture may not be too commonplace, by reason of sticking to present-day patterns, it may be well to mention one such possibility, not to prophesy but merely to suggest, for prophecy based on extension of the known has substance, while prophecy founded on the unknown is only a doubly involved guess.

All our steps in creating or absorbing material of the record proceed through one of the senses - the tactile when we touch keys, the oral when we speak or listen, the visual when we read. Is it not possible that some day the path may be established more directly?

We know that when the eye sees, all the consequent information is transmitted to the brain by means of electrical vibrations in the channel of the optic nerve. This is an exact analogy with the electrical vibrations which occur in the cable of a television set: they convey the picture from the photocells which see it to the radio transmitter from which it is broadcast. We know further that if we can approach that cable with the proper instruments, we do not need to touch it; we can pick up those vibrations by electrical induction and thus discover and reproduce the scene which is being transmitted, just as a telephone wire may be tapped for its message.

The impulses which flow in the arm nerves of a typist convey to her fingers the translated information which reaches her eye or ear, in order that the fingers may be caused to strike the proper keys. Might not these currents be intercepted, either in the original form in which information is conveyed to the brain, or in the marvelously metamorphosed form in which they then proceed to the hand?

By bone conduction we already introduce sounds into the nerve channels of the deaf in order that they may hear. Is it not possible that we may learn to introduce them without the present cumbersomeness of first transforming electrical vibrations to mechanical ones, which the human mechanism promptly transforms back to the electrical form? With a couple of electrodes on the skull the encephalograph now produces pen-and-ink traces which bear some relation to the electrical phenomena going on in the brain itself. True, the record is unintelligible, except as it points out certain gross misfunctioning of the cerebral mechanism; but who would now place bounds on where such a thing may lead?

In the outside world, all forms of intelligence, whether of sound or sight, have been reduced to the form of varying currents in an electric circuit in order that they may be transmitted. Inside the human frame exactly the same sort of process occurs. Must we always transform to mechanical movements in order to proceed from one electrical phenomenon to another? It is a suggestive thought, but it hardly warrants prediction without losing touch with reality and immediateness.

Presumably man's spirit should be elevated if he can better review his shady past and analyze more completely and objectively his present problems. He has built a civilization so complex that he needs to mechanize his record more fully if he is to push his experiment to its logical conclusion and not merely become bogged down part way there by overtaxing his limited memory. His excursion may be more enjoyable if he can reacquire the privilege of forgetting the manifold things he does not need to have immediately at hand, with some assurance that he can find them again if they prove important.

The applications of science have built man a well-supplied house, and are teaching him to live healthily therein. They have enabled him to throw masses of people against another with cruel weapons. They may yet allow him truly to encompass the great record and to grow in the wisdom of race experience. He may perish in conflict before he learns to wield that record for his true good. Yet, in the application of science to the needs and desires of man, it would seem to be a singularly unfortunate stage at which to terminate the process, or to lose hope as to the outcome.

duchier@cs.sfu.ca

The mission of IDEAlliance is to advance and advocate information technologies solutions and business process integration for its members with a user focus on content-graphics creation, management, production, and distribution and its data workflow across the end-to-end value chain.

Focus

TechDoc 2004 Summer Workshops
June 28-30, 2004
Marriott Crystal Gateway Hotel
Arlington, VA

Register Today! Deadline June 23rd

A Network of IDEAlliance
TechDoc™

NEWS

Arbortext Wins Award for Best New Product in 2nd Annual American Business Awards Competition
May 19, 2004

NEW Shipment and Logistics For Newspaper Inserts Specification Announced
60-Day Public Review Period Opens May 15
May 15, 2004

IDEAlliance Announces 2004 Postal Leadership Awards
May 12, 2004

UPU Approves International Address Standard
April 22, 2004

The XML.com Interview: Jeff Barr - Keynote Speaker at XML Europe 2004
March 31, 2004

More News...

EVENTS & MEETINGS

Digital Ad Lab NY - Soft-Proofing Workflow: What's the Buzz All About?
June 3, 2004

TechDoc 2004 Summer Workshops
June 28-30, 2004

Extreme Markup Languages 2004
August 2-6, 2004

SPECTRUM 2004
September 19-22, 2004

XML 2004
November 15-19, 2004

More Events...

Committee & Working Group Meetings

Addressing/Distribution Committee
June 15-16, 2004
Maple Grove, MN
Contact: Georgia Volakis

papiNet Publication Papers Implementation Group
June 29, 2004
8:30 to 2:30pm
Via Webinar
Contact: David Steinhardt

POPULAR RESOURCES

About IDEAlliance

- Company Overview

IDEAlliance Webinars

- Upcoming Webinars
- On-Demand Webinars

Industry Standards

- DISC Specifications
- GRACoL 6.0
- GRACoL Press Sheets
- Lab Reference Card
- Mail.dat User License Code
- Mail.dat 02-2 Spec
- T-Reference Card

Post-Conference News & Proceedings

- Print Distribution 2004
- XML Europe 2004
- Primex 2004

Independent Consultants Network

The Independent Consultants Network is an organization of printing & publishing industry consultants who work with IDEAlliance to provide services for members and to support IDEAlliance Working Groups, Committees and Distance Education activitites.

Content Creation, Management & Delivery

ICE Working Group

PRISM Working Group

Independent Consultants Network

TechDoc Network

News Summit Network

Digital Advertising & Production Workflow

DISC Working Group

SPACE Working Group

PROSE Working Group

GRACoL Committee

GRACoL Network

Digital Ad Lab/NYC Network

Supply Chain Management

papiNet Working Group

XBITS Working Group

B2B Paper Committee

Paper Inventory Committee

papiNet PPI Group

Postal & Newsstand Distribution

ADIS Working Group

Mail.dat Working Group

Shipment & Logistics Planning Working Group

Addressing/Distribution Committee

Canada Panel Committee

NLAC Council

POISE Committee

NMBFC, Affiliate

MTAC Committee

Contents **See also**

Tim Berners-Lee

Weaving the Web by Tim Berners-Lee with Mark Fischetti, (Harper San Francisco; Hardback: ISBN:0062515861, Abridged audio cassette abridged ISBN:0694521256) and various other languages.

Bio

A graduate of Oxford University, England, Tim now holds the 3Com Founders chair at the Laboratory for Computer Science and Artificial Intelligence Lab (CSAIL)at the Massachusetts Institute of Technology (MIT). He directs the World Wide Web Consortium, an open forum of companies and organizations with the mission to lead the Web to its full potential.

With a background of system design in real-time communications and text processing software development, in 1989 he invented the World Wide Web, an internet-based hypermedia initiative for global information sharing. while working at CERN, the European Particle Physics Laboratory. He wrote the first web client (browser-editor) and server in 1990.

Before coming to CERN, Tim worked with Image Computer Systems, of Ferndown, Dorset, England and before that as a principal engineer with Plessey Telecommunications, in Poole, England.

(Longer bio)

Before you mail me

- If you need someone to find something for you about some arbitrary subject (travel agents, or parakeets or whatever), don't ask me, but try the Virtual Library for example, or your favorite search engine.
- If you want to know how to run a server, or how to edit HTML, check the W3C web or your local bookstore. I'm sorry I can't answer individual requests for help.
- If you can't access something on `www.w3.org` , you find bad links from www.w3.org pages, or errors in the hypertext please see the webmaster's documentation..
- If you are doing homework or a school project on the history of the Web then please check my Kid's questions, or the more general Frequently Asked Questions; and also, by other people: Web FAQ,W3C FAQ, my press FAQ as almost everything I have is there or linked from this page. I am sorry I cannot help with individual projects.
- If you are a member of the press and need clarification or an interview, please mail w3t-pr@w3.org (and Cc me) with details.
- If it is about a possible speaking engagement, see below.

If you have a serious comment on things I have signed, then do email me. I am also always open to discussion with W3C Advisory Committee representatives.

What not to email

Email is safe unless it contains programs. (Data and documents are fine, programs are not). If you send me a program, I will not run it, as it could damage my system and could be a virus.

- Note: Documents for Microsoft word, Excel, and possibly other Office programs tend to execute programs (scripts) in what you would expect to be harmless documents. These can expose my machine to viruses, because these programs do not (it seems) prevent scripts from running within a document when it received by email. Please do not send me Microsoft Office documents.
- If you are sending text, please send it as plain text or HTML. If you use your favorite word process, slide tool, etc, and send it in that program's format, then you are forcing me install proprietary software on whatever machine I read them on. .
- If your email is sent from Microsoft Outlook, and contains an attachment, I will be more likely to discard it as I understand that a famous series of viruses in 2001 resulted from Outlook's tendency to execute scripts in email, and used up a huge amount of my and my colleague's time.

What you can email

- These are all good document standards: Plain text messages, HTML (sometimes called rich text) pages without scripts, Photos (JPEG files, PNG, GIF), SMIL, RDF/ XML N3 and so on. All these can be sent as messages or as attachments to messages. I can read them with a variety of software programs, and they cannot contain viruses. If you don't need anything else, then use plain text.

  These are good rules when emailing anyone.

Address

- Directions

Email
timbl @ w3.org
Address
77 Massachusetts Ave.
MIT Room 32-G524
Cambridge MA 02139
USA
Latitude
N 42.3633690
Longitude
W 71.091796
Phone
+1 (617) 253 5702
Fax:
+1 (617) 258 5999

Talks, articles, etc

If you want to know what we are working on now, look at the W3C site and check out all the activities at W3C.

Past present and future:

- Slides from some talks
- Design Issues: Technical and philosophical notes on web architecture A series of notes about how the web actually works and how to design new technology.
- Disclosures: Mainspring Communications; Curl Corp, Akamai

Essays and articles in text form

- Japan Prize commemorative lecture on the universality of the Web (2002)

- The future of the Web - LCS 35th anniversary talk transcript
- WWW, UU and I - Unitarian Universalism and the Web (1998/4)
- A one-page personal history of the web (1998/5/7)
- Realizing the full potential of the web (1997/12/3)
- The web: Past, Present and Future (1996)
- The Web; Europe and the US; Harmony and Diversity (1996)
- Hypertext and Our Collective Destiny , (1995)
- Presentation to CDA challenge by CDT et al , 28 Feb 1996

Chronological order::

- D.M.Sendall. R.I.P. July 15 1999
- Article about me in NY Times Dec 18 1995
- History of the web: some pointers
- Original proposal for a global hypertext project at CERN (1989)

Speaking Engagements

I do few of these because of pressure of work and the detrimental effects of travel. However, I do enjoy occasionally giving a keynote address. If you have something you seriously think I would be interested in speaking at, please send email to timbl+speaking@w3.org with details of the event, projected audience size and profile, location and date. (Please do not contact mutual friends or family to ask for a favor for your company, as that puts unfair pressure on everyone. Just ask directly.)

AV Requirements

If I use slides (I often do not) I use a laptop -- currently a Mac running OSX. I do not need audio from the laptop.

If you want to test your video on similar stuff, run a web browser on a recent one of my previous talks.

Press: - Interviews and material

If you need a photo for publication, please email w3t-pr@w3.org.

Alternatively, you can ask

- The press office at CERN (+41 22 767 6111)
- The MIT Press Office. +1 617 253 2700

If you need an **interview** for an article, please check the

- Web FAQ ,
- W3 Consortium FAQ
- my press FAQ

first, then please use email rather than phone. Please contact w3t-pr@w3.org the general PR request line at W3C, rather than Amy van der Hiel (my assistant) or Janet Daly (Head of Communications at W3C) to set up interviews with me or with other W3C staff.


Photos: This, from my original home page, at the WWW Wizards Workshop, Cambridge MA 1993 [photo Steve Putz]. Top of page: in Sheldonian, Oxford [LeFevre communications, 2001.]

$Id: Overview.html,v 1.82 2004/04/20 13:37:13 amy Exp $ timbl timbl

disclaimer

A hand conversion to HTML of the original MacWord (or Word for Mac?) document written in March 1989 and later redistributed unchanged apart from the date added in May 1990. Provided for historical interest only. The diagrams are a bit dotty, but available in versioins linked below. The text has not been changed, even to correct errors such as misnumbered figures or unfinished references.

This document was an attempt to persuade CERN management that a global hypertext system was in CERN's interests. Note that the only name I had for it at this time was "Mesh" -- I decided on "World Wide Web" when writing the code in 1990.

Other versions which are available are:

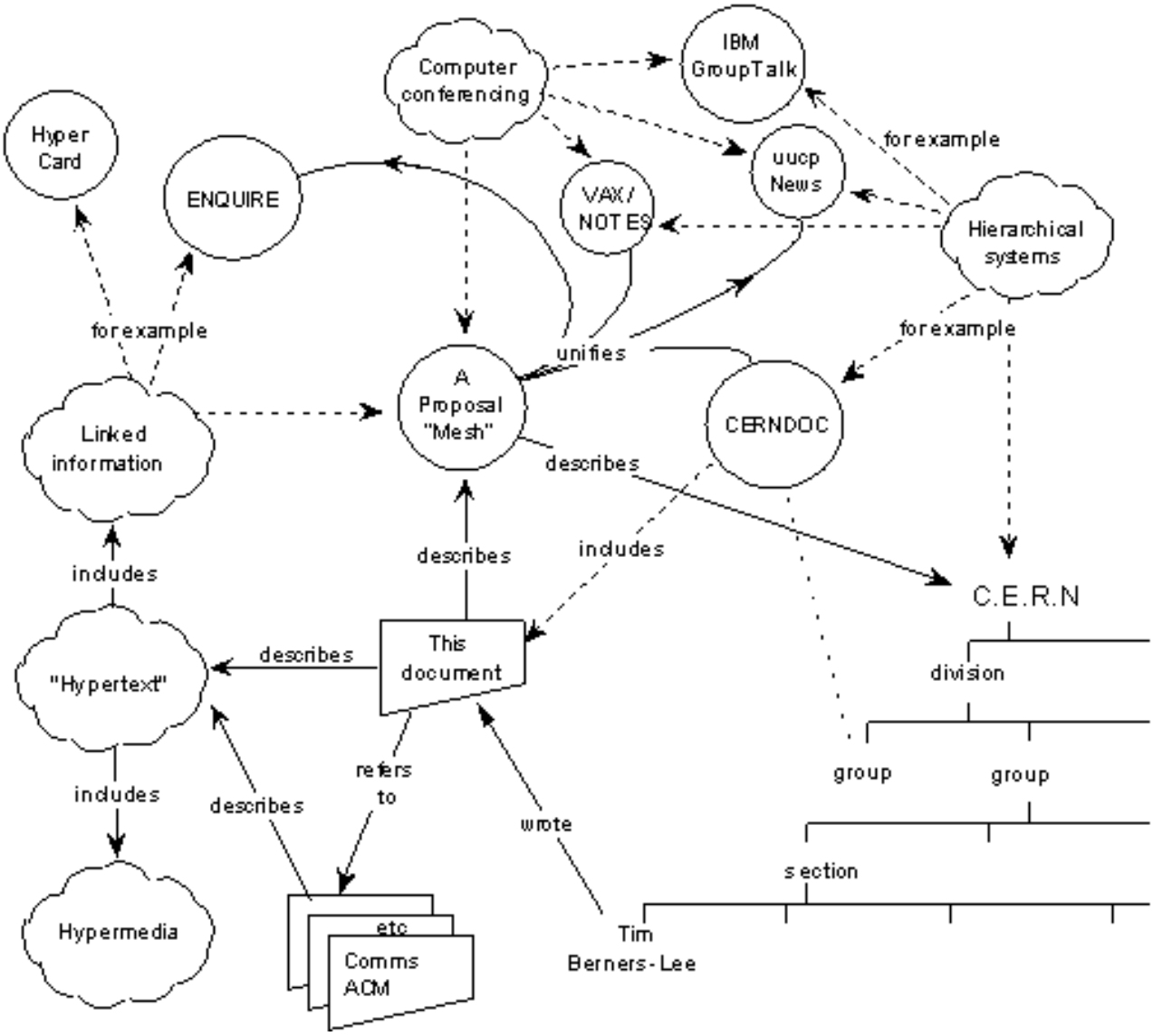
- The original document file (I think - I can't test it)
- The RTF file generated from the above, with scalable drawings
- A Microsoft style HTML file generated from the RTF file by MSword in 1998, with pixely versions of the drawings

---

Information Management: A Proposal

Tim Berners-Lee, CERN
March 1989, May 1990

This proposal concerns the management of general information about accelerators and experiments at CERN. It discusses the problems of loss of information about complex evolving systems and derives a solution based on a distributed hypertext system.



Overview

Many of the discussions of the future at CERN and the LHC era end with the question - ªYes, but how will we ever keep track of such a large project?º This proposal provides an answer to such questions. Firstly, it discusses the problem of information access at CERN. Then, it introduces the idea of linked information systems, and compares them with less flexible ways of finding information.

It then summarises my short experience with non-linear text systems known as ªhypertextº, describes what CERN needs from such a system, and what industry may provide. Finally, it suggests steps we should take to involve ourselves with hypertext now, so that individually and collectively we may understand what we are creating.

Losing Information at CERN

CERN is a wonderful organisation. It involves several thousand people, many of them very creative, all working toward common goals. Although they are nominally organised into a hierarchical management structure,this does not constrain the way people will communicate, and share information, equipment and software across groups.

The actual observed working structure of the organisation is a multiply connected "web" whose

interconnections evolve with time. In this environment, a new person arriving, or someone taking on a new task, is normally given a few hints as to who would be useful people to talk to. Information about what facilities exist and how to find out about them travels in the corridor gossip and occasional newsletters, and the details about what is required to be done spread in a similar way. All things considered, the result is remarkably successful, despite occasional misunderstandings and duplicated effort.

A problem, however, is the high turnover of people. When two years is a typical length of stay, information is constantly being lost. The introduction of the new people demands a fair amount of their time and that of others before they have any idea of what goes on. The technical details of past projects are sometimes lost forever, or only recovered after a detective investigation in an emergency. Often, the information has been recorded, it just cannot be found.

If a CERN experiment were a static once-only development, all the information could be written in a big book. As it is, CERN is constantly changing as new ideas are produced, as new technology becomes available, and in order to get around unforeseen technical problems. When a change is necessary, it normally affects only a small part of the organisation. A local reason arises for changing a part of the experiment or detector. At this point, one has to dig around to find out what other parts and people will be affected. Keeping a book up to date becomes impractical, and the structure of the book needs to be constantly revised.

The sort of information we are discussing answers, for example, questions like

- Where is this module used?
- Who wrote this code? Where does he work?
- What documents exist about that concept?
- Which laboratories are included in that project?
- Which systems depend on this device?
- What documents refer to this one?

The problems of information loss may be particularly acute at CERN, but in this case (as in certain others), CERN is a model in miniature of the rest of world in a few years time. CERN meets now some problems which the rest of the world will have to face soon. In 10 years, there may be many commercial solutions to the problems above, while today we need something to allow us to continue.

Linked information systems

In providing a system for manipulating this sort of information, the hope would be to allow a pool of information to develop which could grow and evolve with the organisation and the projects it describes. For this to be possible, the method of storage must not place its own restraints on the information. This is why a "web" of notes with links (like references) between them is far more useful than a fixed hierarchical system. When describing a complex system, many people resort to diagrams with circles and arrows. Circles and arrows leave one free to describe the interrelationships between things in a way that tables, for example, do not. The system we need is like a diagram of circles and arrows, where circles and arrows can stand for anything.

We can call the circles nodes, and the arrows links. Suppose each node is like a small note, summary article, or comment. I'm not over concerned here with whether it has text or graphics or both. Ideally, it represents or describes one particular person or object. Examples of nodes can be

- People
- Software modules
- Groups of people
- Projects
- Concepts
- Documents
- Types of hardware
- Specific hardware objects

The arrows which links circle A to circle B can mean, for example, that A...

- depends on B
- is part of B
- made B
- refers to B
- uses B
- is an example of B

These circles and arrows, nodes and links, have different significance in various sorts of conventional diagrams:

Diagram	Nodes are	Arrows mean
Family tree	People	"Is parent of"
Dataflow diagram	Software modules"	Passes data to"
Dependency	Module	"Depends on"
PERT chart	Tasks	"Must be done before"
Organisational chart	People	"Reports to"

The system must allow any sort of information to be entered. Another person must be able to find the information, sometimes without knowing what he is looking for.

In practice, it is useful for the system to be aware of the generic types of the links between items (dependences, for example), and the types of nodes (people, things, documents..) without imposing any limitations.

The problem with trees

Many systems are organised hierarchically. The CERNDOC documentation system is an example, as is the Unix file system, and the VMS/HELP system. A tree has the practical advantage of giving every node a unique name. However, it does not allow the system to model the real world. For example, in a hierarchical HELP system such as VMS/HELP, one often gets to a leaf on a tree such as

```
HELP COMPILER SOURCE_FORMAT PRAGMAS DEFAULTS
```

only to find a reference to another leaf: "Please see

```
HELP COMPILER COMMAND OPTIONS DEFAULTS PRAGMAS"
```

and it is necessary to leave the system and re-enter it. What was needed was a link from one node to another, because in this case the information was not naturally organised into a tree.

Another example of a tree-structured system is the uucp News system (try 'rn' under Unix). This is a hierarchical system of discussions ("newsgroups") each containing articles contributed by many people. It is a very useful method of pooling expertise, but suffers from the inflexibility of a tree. Typically, a discussion under one newsgroup will develop into a different topic, at which point it ought to be in a different part of the tree. (See Fig 1).

```
From mcvax!uunet!pyrdc!pyrnj!rutgers!bellcore!geppetto!duncan Thu Mar...
Article 93 of alt.hypertext:
Path: cernvax!mcvax!uunet!pyrdc!pyrnj!rutgers!bellcore!geppetto!duncan
>From: duncan@geppetto.ctt.bellcore.com (Scott Duncan)
Newsgroups: alt.hypertext
Subject: Re: Threat to free information networks
Message-ID: <14646@bellcore.bellcore.com>
Date: 10 Mar 89 21:00:44 GMT
References: <1784.2416BB47@isishq.FIDONET.ORG> <3437@uhccux.uhcc...
Sender: news@bellcore.bellcore.com
Reply-To: duncan@ctt.bellcore.com (Scott Duncan)
Organization: Computer Technology Transfer, Bellcore
Lines: 18

Doug Thompson has written what I felt was a thoughtful article on
censorship -- my acceptance or rejection of its points is not
particularly germane to this posting, however.

In reply Greg Lee has somewhat tersely objected.

My question (and reason for this posting) is to ask where we might
logically take this subject for more discussion.  Somehow alt.hypertext
does not seem to be the proper place.

Would people feel it appropriate to move to alt.individualism or even
one of the soc groups.  I am not so much concerned with the specific
issue of censorship of rec.humor.funny, but the views presented in
Greg's article.

Speaking only for myself, of course, I am...
Scott P. Duncan (duncan@ctt.bellcore.com OR ...!bellcore!ctt!duncan)
                (Bellcore, 444 Hoes Lane  RRC 1H-210, Piscataway, NJ...)
                (201-699-3910 (w)   201-463-3683 (h))
```

Fig 1. An article in the UUCP News scheme.

The Subject field allows notes on the same topic to be linked together within a "newsgroup". The name of the newsgroup (alt.hypertext) is a hierarchical name. This particular note is expresses a problem with the strict tree structure of the scheme: this discussion is related to several areas. Note that the "References", "From" and "Subject" fields can all be used to generate links.

The problem with keywords

Keywords are a common method of accessing data for which one does not have the exact coordinates. The usual problem with keywords, however, is that two people never chose the same keywords. The keywords then become useful only to people who already know the application well.

Practical keyword systems (such as that of VAX/NOTES for example) require keywords to be registered. This is already a step in the right direction. A linked system takes this to the next logical step. Keywords can be nodes which stand for a concept. A keyword node is then no different from any other node. One can link documents, etc., to keywords. One can then find keywords by finding any node to which they are related. In this way, documents on similar topics are indirectly linked, through their key concepts. A keyword search then becomes a search starting from a small number of named nodes, and finding nodes which are close to all of them.

It was for these reasons that I first made a small linked information system, not realising that a term had already been coined for the idea: "hypertext".

A solution: Hypertext

Personal Experience with Hypertext

In 1980, I wrote a program for keeping track of software with which I was involved in the PS control system. Called Enquire, it allowed one to store snippets of information, and to link related pieces together in any way. To find information, one progressed via the links from one sheet to another, rather like in the old computer game "adventure". I used this for my personal record of people and modules. It was similar to the application Hypercard produced more recently by Apple for the Macintosh. A difference was that Enquire, although lacking the fancy graphics, ran on a multiuser system, and allowed many people to access the same data.

```
Documentation of the RPC project                          (concept)


   Most of the documentation is available on VMS, with the two
   principle manuals being stored in the CERNDOC system.

    1) includes: The VAX/NOTES conference VXCERN::RPC
    2) includes: Test and Example suite
    3) includes: RPC BUG LISTS
    4) includes: RPC System: Implementation Guide
       Information for maintenance, porting, etc.
    5) includes: Suggested Development Strategy for RPC Applications
    6) includes: "Notes on RPC", Draft 1, 20 feb 86
```

```
    7) includes: "Notes on Proposed RPC Development" 18 Feb 86
    8) includes: RPC User Manual
       How to build and run a distributed system.
    9) includes: Draft Specifications and Implementation Notes
   10) includes: The RPC HELP facility
   11) describes: THE REMOTE PROCEDURE CALL PROJECT in DD/OC




 Help  Display  Select  Back  Quit Mark  Goto_mark  Link  Add  Edit
```

Fig 2. A screen in an Enquire scheme.

This example is basically a list, so the list of links is more important than the text on the node itself. Note that each link has a type ("includes" for example) and may also have comment associated with it. (The bottom line is a menu bar.)

Soon after my re-arrival at CERN in the DD division, I found that the environment was similar to that in PS, and I missed Enquire. I therefore produced a version for the VMS, and have used it to keep track of projects, people, groups, experiments, software modules and hardware devices with which I have worked. I have found it personally very useful. I have made no effort to make it suitable for general consumption, but have found that a few people have successfully used it to browse through the projects and find out all sorts of things of their own accord.

Hot spots

Meanwhile, several programs have been made exploring these ideas, both commercially and academically. Most of them use "hot spots" in documents, like icons, or highlighted phrases, as sensitive areas. touching a hot spot with a mouse brings up the relevant information, or expands the text on the screen to include it. Imagine, then, the references in this document, all being associated with the network address of the thing to which they referred, so that while reading this document you could skip to them with a click of the mouse.

"Hypertext" is a term coined in the 1950s by Ted Nelson [...], which has become popular for these systems, although it is used to embrace two different ideas. One idea (which is relevant to this problem) is the concept: "Hypertext": Human-readable information linked together in an unconstrained way.

The other idea, which is independent and largely a question of technology and time, is of multimedia documents which include graphics, speech and video. I will not discuss this latter aspect further here, although I will use the word "Hypermedia" to indicate that one is not bound to text.

It has been difficult to assess the effect of a large hypermedia system on an organisation, often because these systems never had seriously large-scale use. For this reason, we require large amounts of existing information should be accessible using any new information management system.

CERN Requirements

To be a practical system in the CERN environment, there are a number of clear practical requirements.

Remote access across networks.

CERN is distributed, and access from remote machines is essential.

Heterogeneity

Access is required to the same data from different types of system (VM/CMS, Macintosh, VAX/VMS, Unix)

Non-Centralisation

Information systems start small and grow. They also start isolated and then merge. A new system must allow existing systems to be linked together without requiring any central control or coordination.

Access to existing data

If we provide access to existing databases as though they were in hypertext form, the system will get off the ground quicker. This is discussed further below.

Private links

One must be able to add one's own private links to and from public information. One must also be able to annotate links, as well as nodes, privately.

Bells and Whistles

Storage of ASCII text, and display on 24x80 screens, is in the short term sufficient, and essential. Addition of graphics would be an optional extra with very much less penetration for the moment.

Data analysis

An intriguing possibility, given a large hypertext database with typed links, is that it allows some degree of automatic analysis. It is possible to search, for example, for anomalies such as undocumented software or divisions which contain no people. It is possible to generate lists of people or devices for other purposes, such as mailing lists of people to be informed of changes. It is also possible to look at the topology of an organisation or a project, and draw conclusions about how it should be managed, and how it could evolve. This is particularly useful when the database becomes very large, and groups of projects, for example, so interwoven as to make it difficult to see the wood for the trees.

In a complex place like CERN, it's not always obvious how to divide people into groups. Imagine making a large three-dimensional model, with people represented by little spheres, and strings between people who have something in common at work.

Now imagine picking up the structure and shaking it, until you make some sense of the tangle: perhaps, you see tightly knit groups in some places, and in some places weak areas of communication spanned by only a

few people. Perhaps a linked information system will allow us to see the real structure of the organisation in which we work.

Live links

The data to which a link (or a hot spot) refers may be very static, or it may be temporary. In many cases at CERN information about the state of systems is changing all the time. Hypertext allows documents to be linked into "live" data so that every time the link is followed, the information is retrieved. If one sacrifices portability, it is possible so make following a link fire up a special application, so that diagnostic programs, for example, could be linked directly into the maintenance guide.

Non requirements

Discussions on Hypertext have sometimes tackled the problem of copyright enforcement and data security. These are of secondary importance at CERN, where information exchange is still more important than secrecy. Authorisation and accounting systems for hypertext could conceivably be designed which are very sophisticated, but they are not proposed here.

In cases where reference must be made to data which is in fact protected, existing file protection systems should be sufficient.

Specific Applications

The following are three examples of specific places in which the proposed system would be immediately useful. There are many others.

Development Project Documentation.

The Remote procedure Call project has a skeleton description using Enquire. Although limited, it is very useful for recording who did what, where they are, what documents exist, etc. Also, one can keep track of users, and can easily append any extra little bits of information which come to hand and have nowhere else to be put. Cross-links to other projects, and to databases which contain information on people and documents would be very useful, and save duplication of information.

Document retrieval.

The CERNDOC system provides the mechanics of storing and printing documents. A linked system would allow one to browse through concepts, documents, systems and authors, also allowing references between documents to be stored. (Once a document had been found, the existing machinery could be invoked to print it or display it).

The "Personal Skills Inventory".

Personal skills and experience are just the sort of thing which need hypertext flexibility. People can be linked to projects they have worked on, which in turn can be linked to particular machines, programming languages, etc.

The State of the Art in Hypermedia

An increasing amount of work is being done into hypermedia research at universities and commercial research labs, and some commercial systems have resulted. There have been two conferences, Hypertext '87 and '88, and in Washington DC, the National Institute of Standards and Technology (NST) hosted a workshop on standardisation in hypertext, a followup of which will occur during 1990.

The Communications of the ACM special issue on Hypertext contains many references to hypertext papers. A bibliography on hypertext is given in [NIST90], and a uucp newsgroup alt.hypertext exists. I do not, therefore, give a list here.

Browsing techniques

Much of the academic research is into the human interface side of browsing through a complex information space. Problems addressed are those of making navigation easy, and avoiding a feeling of being "lost in hyperspace". Whilst the results of the research are interesting, many users at CERN will be accessing the system using primitive terminals, and so advanced window styles are not so important for us now.

Interconnection or publication?

Most systems available today use a single database. This is accessed by many users by using a distributed file system. There are few products which take Ted Nelson's idea of a wide "docuverse" literally by allowing links between nodes in different databases. In order to do this, some standardisation would be necessary. However, at the standardisation workshop, the emphasis was on standardisation of the format for exchangeable media, nor for networking. This is prompted by the strong push toward publishing of hypermedia information, for example on optical disk. There seems to be a general consensus about the abstract data model which a hypertext system should use.

Many systems have been put together with little or no regard for portability, unfortunately. Some others, although published, are proprietary software which is not for external release. However, there are several interesting projects and more are appearing all the time. Digital's "Compound Document Architecture" (CDA) , for example, is a data model which may be extendible into a hypermedia model, and there are rumours that this is a way Digital would like to go.

Incentives and CALS

The US Department of Defence has given a big incentive to hypermedia research by, in effect, specifying hypermedia documentation for future procurement. This means that all manuals for parts for defence equipment must be provided in hypermedia form. The acronym CALS stands for ªComputer-aided Acquisition and Logistic Support).

There is also much support from the publishing industry, and from librarians whose job it is to organise information.

What will the system look like?

Let us see what components a hypertext system at CERN must have. The only way in which sufficient

flexibility can be incorporated is to separate the information storage software from the information display software, with a well defined interface between them. Given the requirement for network access, it is natural to let this clean interface coincide with the physical division between the user and the remote database machine.

This division also is important in order to allow the heterogeneity which is required at CERN (and would be a boon for the world in general).

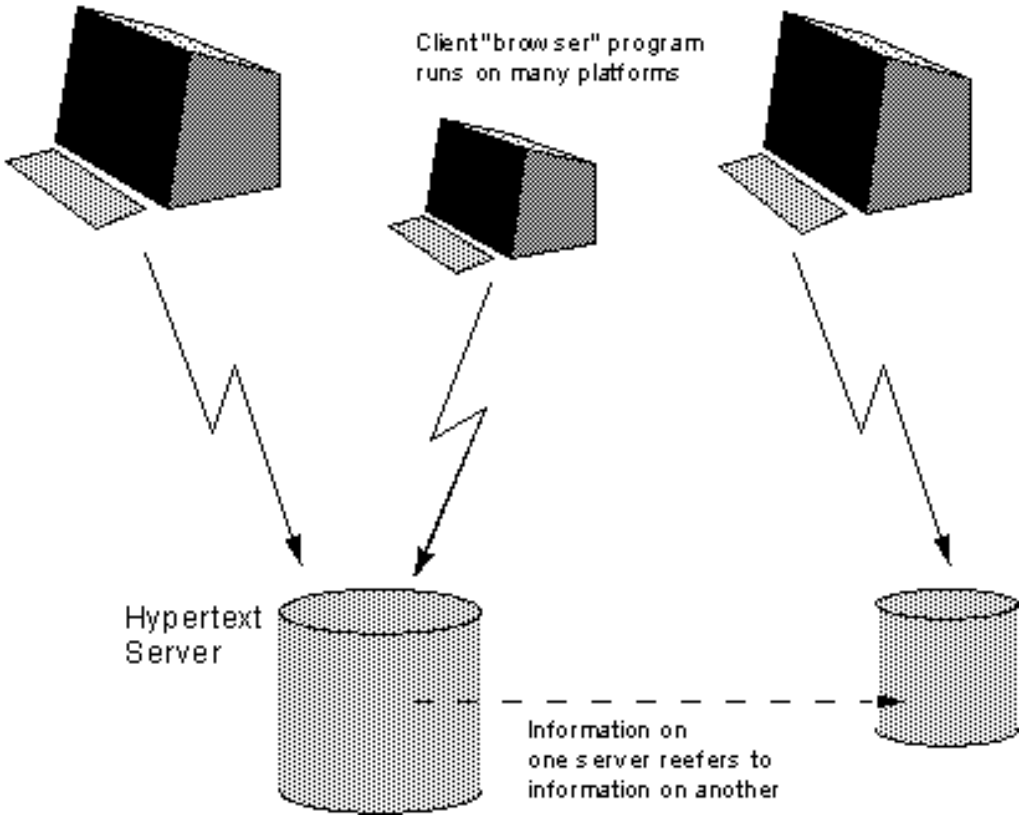


Fig 2. A client/server model for a distributed hypertext system.

Therefore, an important phase in the design of the system is to define this interface. After that, the development of various forms of display program and of database server can proceed in parallel. This will have been done well if many different information sources, past, present and future, can be mapped onto the definition, and if many different human interface programs can be written over the years to take advantage of new technology and standards.

Accessing Existing Data

The system must achieve a critical usefulness early on. Existing hypertext systems have had to justify themselves solely on new data. If, however, there was an existing base of data of personnel, for example, to which new data could be linked, the value of each new piece of data would be greater.

What is required is a gateway program which will map an existing structure onto the hypertext model, and allow limited (perhaps read-only) access to it. This takes the form of a hypertext server written to provide existing information in a form matching the standard interface. One would not imagine the server actually generating a hypertext database from and existing one: rather, it would generate a hypertext view of an existing database.

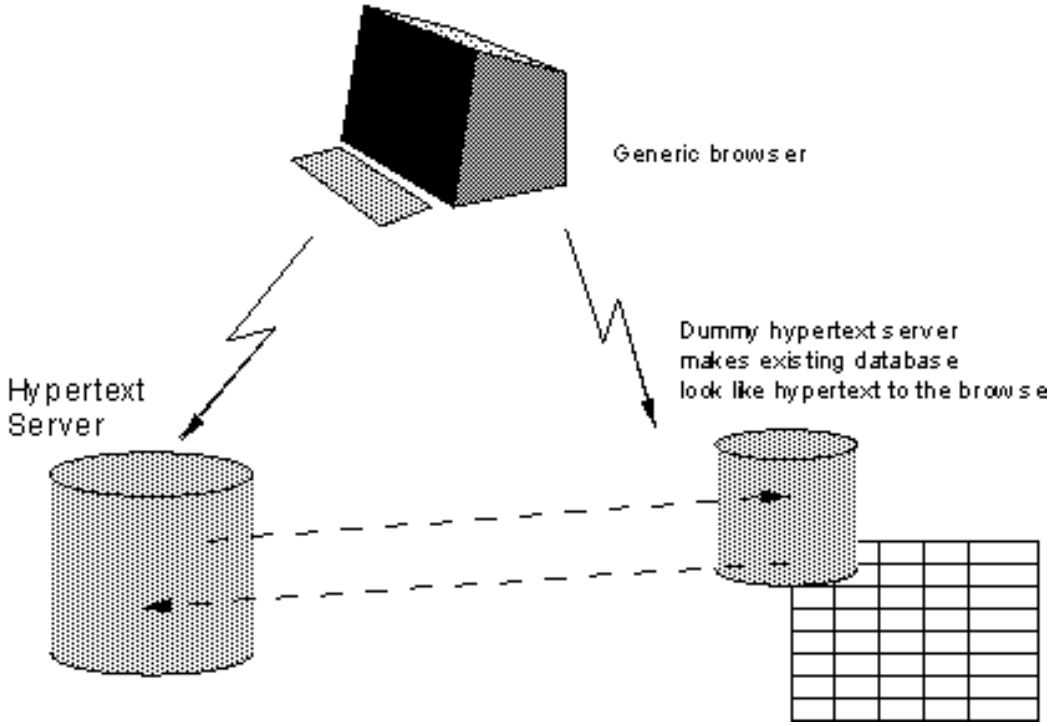


Fig 3. A hypertext gateway allows existing data to be seen in hypertext form by a hypertext browser.

Some examples of systems which could be connected in this way are

uucp News
: This is a Unix electronic conferencing system. A server for uucp news could makes links between notes on the same subject, as well as showing the structure of the conferences.

VAX/Notes
: This is Digital's electronic conferencing system. It has a fairly wide following in FermiLab, but much less in CERN. The topology of a conference is quite restricting.

CERNDOC
: This is a document registration and distribution system running on CERN's VM machine. As well as documents, categories and projects, keywords and authors lend themselves to representation as hypertext nodes.

File systems
: This would allow any file to be linked to from other hypertext documents.

The Telephone Book
: Even this could even be viewed as hypertext, with links between people and sections, sections and groups, people and floors of buildings, etc.

The unix manual
: This is a large body of computer-readable text, currently organised in a flat way, but which also contains link information in a standard format ("See also..").

Databases
: A generic tool could perhaps be made to allow any database which uses a commercial DBMS to be displayed as a hypertext view.

In some cases, writing these servers would mean unscrambling or obtaining details of the existing protocols and/or file formats. It may not be practical to provide the full functionality of the original system through hypertext. In general, it will be more important to allow read access to the general public: it may be that there is a limited number of people who are providing the information, and that they are content to use the existing facilities.

It is sometimes possible to enhance an existing storage system by coding hypertext information in, if one knows that a server will be generating a hypertext representation. In 'news' articles, for example, one could use (in the text) a standard format for a reference to another article. This would be picked out by the hypertext gateway and used to generate a link to that note. This sort of enhancement will allow greater integration between old and new systems.

There will always be a large number of information management systems - we get a lot of added usefulness from being able to cross-link them. However, we will lose out if we try to constrain them, as we will exclude systems and hamper the evolution of hypertext in general.

Conclusion

We should work toward a universal linked information system, in which generality and portability are more important than fancy graphics techniques and complex extra facilities.

The aim would be to allow a place to be found for any information or reference which one felt was important, and a way of finding it afterwards. The result should be sufficiently attractive to use that it the information contained would grow past a critical threshold, so that the usefulness the scheme would in turn encourage its increased use.

The passing of this threshold accelerated by allowing large existing databases to be linked together and with new ones.

A Practical Project

Here I suggest the practical steps to go to in order to find a real solution at CERN. After a preliminary discussion of the requirements listed above, a survey of what is available from industry is obviously required. At this stage, we will be looking for a systems which are future-proof:

- portable, or supported on many platforms,
- Extendible to new data formats.

We may find that with a little adaptation, pars of the system we need can be combined from various sources: for example, a browser from one source with a database from another.

I imagine that two people for 6 to 12 months would be sufficient for this phase of the project.

A second phase would almost certainly involve some programming in order to set up a real system at CERN on many machines. An important part of this, discussed below, is the integration of a hypertext system with existing data, so as to provide a universal system, and to achieve critical usefulness at an early stage.

(... and yes, this would provide an excellent project with which to try our new object oriented programming techniques!) TBL March 1989, May 1990

References

[NEL67]

Nelson, T.H. "Getting it out of our system" in Information Retrieval: A Critical Review", G. Schechter, ed. Thomson Books, Washington D.C., 1967, 191-210

[SMISH88]

Smish, J.B and Weiss, S.F,"An Overview of Hypertext",in Communications of the ACM, July 1988 Vol 31, No. 7,and other articles in the same special "Hypertext" issue.

[CAMP88]

Campbell, B and Goodman, J,"HAM: a general purpose Hypertext Abstract Machine",in Communications of the ACM July 1988 Vol 31, No. 7

[ASKCYN88]

Akscyn, R.M, McCracken, D and Yoder E.A,"KMS: A distributed hypermedia system for managing knowledge in originations", in Communications of the ACM , July 1988 Vol 31, No. 7

[HYP88]

Hypertext on Hypertext, a hypertext version of the special Comms of the ACM edition, is avialble from the ACM for the Macintosh or PC.

[RN]

Under unix, type `man rn` to find out about the `rn` command which is used for reading uucp news.

[NOTES]

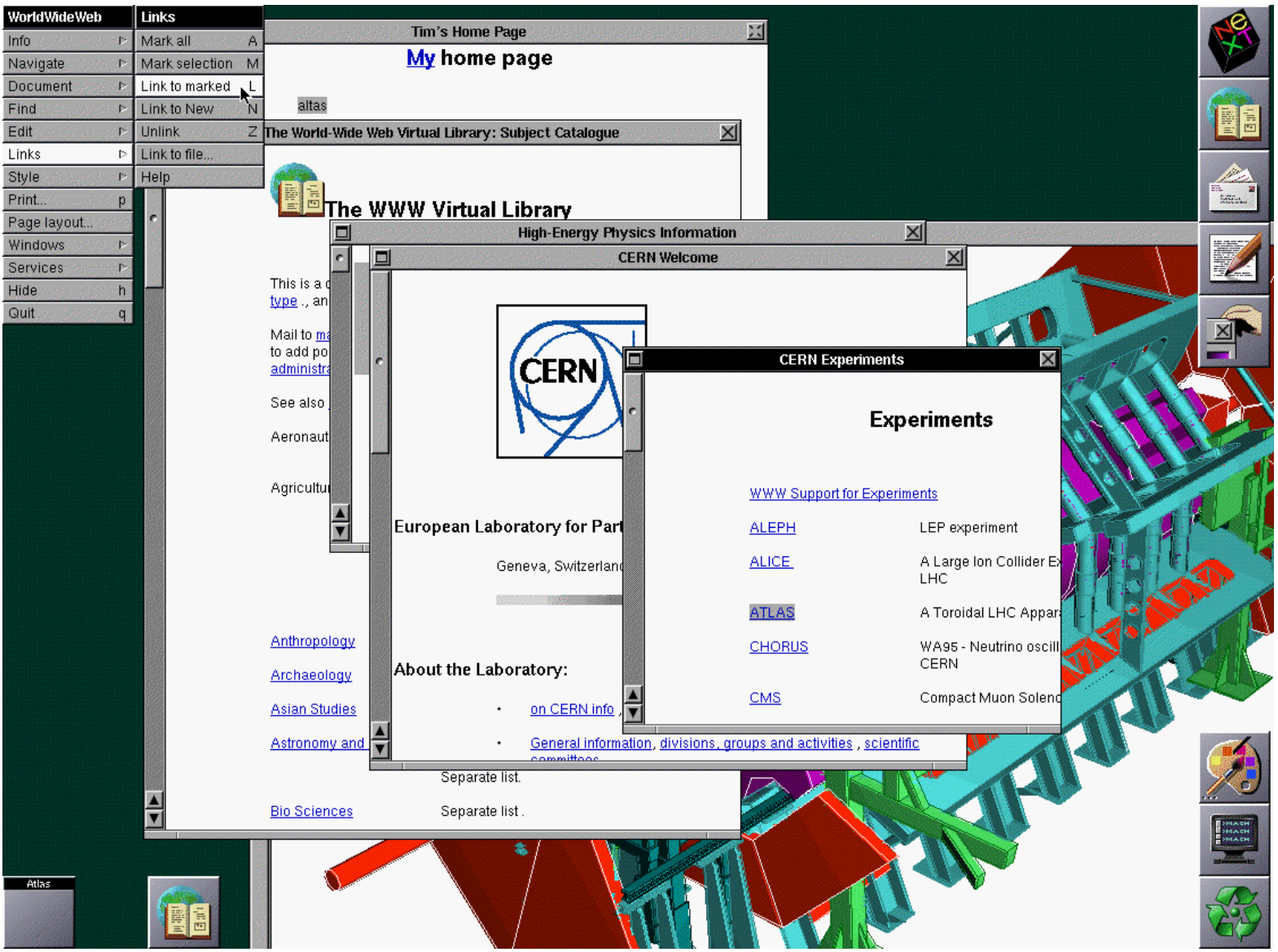
Under VMS, type HELP NOTES to find out about the VAX/NOTES system

[CERNDOC]

On CERNVM, type FIND DOCFIND for infrmation about how to access the CERNDOC programs.

[NIST90]

J. Moline et. al. (ed.) Proceedings of the Hypertext Standardisation Workshop January 16-18, 1990, National Institute of Standards and Technology, pub. U.S. Dept. of Commerce


WorldWideWeb
Info
Navigate
Document
Find
Edit
Links
Style
Print... p
Page layout...
Windows
Services
Hide h
Quit q
Links
Mark all A
Mark selection M
Link to marked L
Link to New N
Unlink Z
Link to file...
Help
Tim's Home Page
My home page
altas
The World-Wide Web Virtual Library: Subject Catalogue
The WWW Virtual Library
High-Energy Physics Information
CERN Welcome
CERN
European Laboratory for Part
Geneva, Switzerland
About the Laboratory:
on CERN info
General information, divisions, groups and activities, scientific
Anthropology
Archaeology
Asian Studies
Astronomy and
Bio Sciences
Separate list.
Separate list.
CERN Experiments
Experiments
WWW Support for Experiments
ALEPH
LEP experiment
ALICE
A Large Ion Collider Ex
LHC
ATLAS
A Toroidal LHC Appar
CHORUS
WA95 - Neutrino oscill
CERN
CMS
Compact Muon Solenc
Atlas

Status of OLD WWW Browser-Editor on the NeXT

This is a complete hypertext generation and browsing application, designed to test the concepts and design decisions of the WWW project.

Old version

The old version is at version 0.17. It is superceded by the new version.

This old version can edit local files, make links easily, format stuff, but has no HTML+ features or even nested lists. It is ONLY available for the M68k NeXT, and that is how it is. The documentation reflects this version.

Author
: TBL

Status:
: Frozen. Beta-test version or prototype available. Work has restarted! wtach this space (Nov 93).

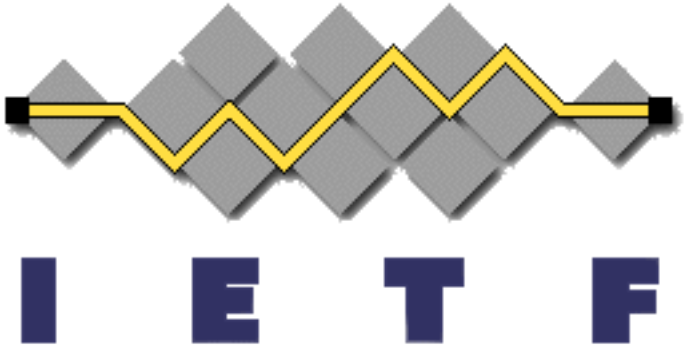
Prerequisites
: NeXTStep 3.0

Latest version
: 0.17 released 1993. . 2.0 pre-alpha contains images, etc. See change history .

Next target:
: Update to HTML+, annotation and forms.

More information:
: User Documentation , Change history , Prioritized list of things still to be done .

IETF Keyword Search **IETF Mirror Sites**



The Internet Engineering Task Force

- Overview of the IETF
- IESG Activities/Actions
- IETF Working Groups
- Internet-Drafts
- RFC Pages
- Additional Information
- The Internet Standards Process
- The Nomcom
- IETF Liaison Activities
- Documents in Last Call
- Meetings
  - 60th IETF - San Diego, CA, USA

  (August 1-6, 2004)
- Proceedings
- Mailing lists
- Intellectual Property Right Notices
- Joining the IETF
- WG Chairs Web Page

Related Web Pages: IAB RFC Editor IANA IRTF

The IETF is an organized activity of the Internet Society

The IETF Secretariat is hosted by the Corporation for National Research Initiatives.

Leading the Web to Its Full Potential...

The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding. On this page, you'll find W3C news, links to W3C technologies and ways to get involved. New visitors can find help in *Finding Your Way at W3C*. We encourage you to read the *Prospectus* and learn more about W3C.

W3C A to Z

- Accessibility
- Amaya
- Annotea
- Binary XML
- CC/PP
- CSS
- CSS Validator
- Device Independence
- DOM
- HTML
- HTML Tidy
- HTML Validator
- HTTP
- InkML
- Internationalization
- Jigsaw
- Libwww
- MathML
- Multimodal Interaction
- OWL
- Patent Policy
- PICS
- PNG
- Privacy and P3P
- Quality Assurance (QA)
- RDF
- Semantic Web
- SMIL

- SOAP/XMLP
- Style
- SVG
- TAG
- Timed Text
- URI/URL
- Validators
- Voice
- WAI
- WebCGM
- Web Services
- Web Ontology
- XForms
- XHTML
- XLink
- XML
- XML Base
- XML Encryption
- XML Key Management
- XML Query
- XML Schema
- XML Signature
- XPath
- XPointer
- XSL and XSLT

More topics...

News

W3C Track Featured at WWW2004

2004-05-14: The W3C Track chaired by Marie-Claire Forgue runs from 19-21 May at the Thirteenth International World Wide Web Conference (WWW2004) in New York, NY USA. W3C Members and Team present three days of content on W3C technologies and achievements. The W3C Track conference room is New York Ballroom B. Conference attendees are also invited to Developers Day presentations on 22 May. (News archive)

Call for Participation: Workshop on Multimodal Interaction

2004-05-14: Position papers are due 11 June for the W3C Workshop on Multimodal

Interaction to be held in Sophia Antipolis, France on 19-20 July. Attendees from user and research communities will discuss current plans, and provide feedback and suggestions for future multimodal work. Read about Workshops and visit the Multimodal Interaction home page at W3C. (News archive)

Working Draft: Web Services Internationalization Usage Scenarios

2004-05-12: The Web Services Task Force of the Internationalization Working Group has released an updated Working Draft of Web Services Internationalization Usage Scenarios with additional guidance for implementers of Web service technologies. The document examines how language, culture and related issues interact with Web services architecture and technology. Comments are welcome on this draft. Visit the Internationalization home page. (News archive)

CSS3 Basic User Interface Is a W3C Candidate Recommendation

2004-05-11: W3C is pleased to announce the advancement of the CSS3 Basic User Interface Module to Candidate Recommendation. The Cascading Style Sheets (CSS) language is used to render structured documents like HTML and XML on screen, on paper, and in speech. This module addresses user interface states and features, element fragments, forms, stylistic attributes in HTML, focus navigation, and styling elements as icons for accessibility. Comments are invited through 11 November. Visit the CSS home page. (News archive)

Working Drafts: Authoring Techniques for XHTML and HTML Internationalization

2004-05-10: The GEO (Guidelines, Education and Outreach) Task Force of the Internationalization Working Group has published three First Public Working Drafts. The drafts cover Specifying the Language of Content, Characters and Encodings and Handling Bidirectional Text. Designed for content authors, the documents are aids to ensuring that HTML and XHTML are written for an international audience. Visit the Internationalization home page. (News archive)

Working Draft: The QA Handbook

2004-05-10: The Quality Assurance (QA) Working Group has released the First Public Working Draft of The QA Handbook. Written for W3C Working Group Chairs and Team Contacts, the document replaces and incorporates the best features of the former QA Framework's Introduction and Operational Guidelines. It provides techniques, tools, and templates for test suites and specifications, and is designed to facilitate and accelerate the work of W3C Working Groups. Visit the QA home page. (News archive)

W3C Markup Validator Upgraded

2004-05-06: W3C is pleased to announce an upgrade to the W3C Markup Validation Service. The new release is easier to use and install. It features new documentation and navigation, and offers helpful explanations and recovery mechanisms instead of fatal errors. Managed by a team of volunteers and the W3C Quality Assurance Activity, and supported by a large community, this validator is the single most popular resource on the W3C Web site. Read the announcement. (News archive)

Past News

Search

Search W3C

Go

Search W3C Mailing Lists

Members

- Member Home Page
- Member Submissions
- Current Members
- W3C Fellows

Get Involved

- Join W3C
- Participate
- Mailing Lists
- Translations
- World Offices
- Workshops
- W3C Donors
- Open Source Software
- Employment
- Subscribe to W3C Weekly News

Mission

- W3C in Seven Points
- Prospectus
- Frequently Asked Questions (FAQ)
- Process Document

W3C Team

- People
- Team Submissions

Presentations

- Upcoming
- Past Talks

News Room

- W3C News Archive
- Press Releases
- W3C in the Press

Systems

- Get Member Password
- System Status

W3C would like to thank the organizations who have contributed hardware, software, and services to W3C.

Read the FAQ and send comments about this page. Syndicate this page with RSS 1.0, an RDF vocabulary used for site summaries.

Webmaster · Last modified: $Date: 2004/05/17 13:25:49 $ |

|


|


|

WD-xml-961114

Extensible Markup Language (XML)

W3C Working Draft 14-Nov-96

This version:
http://www.w3.org/pub/WWW/TR/WD-xml-961114.html
Previous versions:
Latest version:
http://www.textuality.com/sgml-erb/WD-xml.html

Editors:
Tim Bray (Textuality) <tbray@textuality.com>
C. M. Sperberg-McQueen (University of Illinois at Chicago) <cmsmcq@uic.edu>

Status of this memo

This is a W3C Working Draft for review by W3C members and other interested parties. It is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". A list of current W3C working drafts can be found at: http://www.w3.org/pub/WWW/TR

Note: since working drafts are subject to frequent change, you are advised to reference the above URL, rather than the URLs for working drafts themselves.

This work is part of the W3C SGML Activity.

Abstract

Extensible Markup Language (XML) is an extremely simple dialect of SGML which is completely described in this document. The goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. For this reason, XML has been designed for ease of implementation, and for interoperability with both SGML and HTML.
Note on status of this document: This is even more of a moving target than the typical W3C working draft. Several important decisions on the details of XML are still outstanding - members of the W3C SGML Working Group will recognize these areas of particular volatility in the spec, but those who

are not intimately familiar with the deliberative process should be careful to avoid actions based on the content of this document, until the notice you are now reading has been removed.

Table of Contents

---

1. Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects stored on computers and partially describes the behavior of programs which process these objects. Such objects are called XML documents. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879].

XML documents are made up of storage units called entities, which contain either text or binary data. Text is made up of characters, some of which form the character data in the document, and some of which form markup. Markup encodes a description of the document's storage layout, structure, and arbitrary attribute-value pairs associated with that structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an *XML processor* is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, referred to as the *application*. This specification describes some of the required behavior of an XML processor in terms of the manner it must read XML data, and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by a Generic SGML Editorial Review Board formed under the auspices of the W3 Consortium in 1996 and chaired by Jon Bosak of Sun Microsystems, with the very active participation of a Generic SGML Working Group also organized by the W3C. The membership of these groups is given in an appendix.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.

6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness is of minimal importance.

This specification, together with the associated standards, provides all the information necessary to understand XML version 1.0 and construct computer programs to process it.

This version of the XML specification (0.01) is for internal discussion within the SGML ERB only. It should not be distributed outside the ERB.

Known problems in version 0.01:

1. Several items in the bibliography have no references to them; several references in the text do not point to anything in the bibliograpy.
2. The EBNF grammar has not been checked for completeness, and has at least two start productions.
3. The description of conformance in the final section is incomplete.
4. Language exists in the spec which describes the effect of several decisions which have not been taken. Specifically, XML may have INCLUDE/IGNORE marked sections as does SGML, the comment syntax may change, XML may have CONREF attributes, the 8879 syntax for EMPTY elements may be outlawed, XML may choose to rule out what 8879 calls "ambiguous" content models, XML may choose to prohibit overlap between enumerated attribute values for different attributes, the handling for attribute values in the absence of a DTD may be specified, there may be a way to signal whether the DTD is complete, the DTD summary may be dropped, and XML may support parameter entities, and XML may predefine a large number of character entities, for example those from HTML 3.2.

1.2 Relationship to Other Standards

Other standards relevant to users and implementors of XML include:

1. SGML (ISO 8879-1986). Valid XML Documents are SGML documents in the sense described in ISO standard 8879.
2. Unicode, ISO 10646. This specification depends on ISO standard 10646 and the technically identical Unicode Standard, Version 2.0, which define the encodings and meanings of the characters which make up XML text data.
3. IETF RFC 1738. This specification defines the syntax and semantics of Uniform Resource Locators, or URLs.
4. World-Wide Web Consortium Working Draft WD-html32-960909: HTML 3.2 Reference Specification. This includes the repertoire of characters to be predefined by an XML processor.

1.3 Notation

The formal grammar of XML is given using a simple Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one non-terminal or terminal symbol of the grammar, in the form

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are defined by a regular expression, with an initial lowercase letter if they have a more complex definition (i.e. if they require a stack for proper recognition). Literal strings are quoted; unless otherwise noted they are not case-sensitive. The distinction between symbols which can and cannot be recognized using simple regular-expressions is made for clarity only. It may be reflected in the boundary between an implementation's lexical scanner and its parser, but there is are no assumptions about the placement of such a boundary, nor even that the implementation has separate modules for parser and lexical scanner.

Within the expression on the right-hand side of a rule, the meaning of symbols is as shown below:

`#NN`
: where *NN* is a decimal integer, the expression matches the character in ISO 10646 whose UCS-4 bit-string, when interpreted as an unsigned binary number, has the value indicated

`#xNN`
: where *NN* is a hexadecimal integer, the expression matches the character in ISO 10646 whose UCS-4 bit-string, when interpreted as an unsigned binary number, has the value indicated

`[#xNN-#xNN], [a-zA-Z]`
: matches any character with a value in the range(s) indicated (inclusive)

`[^#xNN-#xNN], [^a-z]`
: matches any character with a value *outside* the range indicated

`[^abc]`
: matches any character with a value not among the characters given

`"`*`string`*`"`
: matches the literal string given inside the double quotes

`'`*`string`*`'`
: matches the literal string given inside the single quotes

`a b`
: *a* followed by *b*

`a | b`
: *a* or *b* but not both

`a?`
: *a* or nothing; optional *a*

`a+`
: one or more occurrences of *a*

`a*`
: zero or more occurrences of *a*

`(`*`expression`*`)`
: *expression* is treated as a unit; allows subgroups to carry the operators `?`, `*`, or `+`

`/* ... */`

comment

`[ WFC: ... ]`

Well-formedness check; this identifies by name a check for well-formedness that is associated with a production.

`[ VC: ... ]`

Validity check; this identifies by name a check for validity that is associated with a production.

1.4 Terminology

Some terms used with special meaning in this specification are:

`may`

Conforming data and software may but need not behave as described.

`must`

Conforming data and software must behave as described; otherwise they are in error.

`error`

A violation of the rules of this specification; results are undefined. Conforming software may detect and report an error and may recover from it.

`reportable error`

An error which conforming software must report to the user, unless the user has explicitly disabled error reporting.

`validity constraint`

A rule which applies to all valid XML documents. Violations of validity constraints are errors; they must be reported by validating XML processors.

`well-formedness constraint`

A rule which applies to all well-formed XML documents. Violations of well-formedness constraints are reportable errors.

`at user option`

Conforming software may or must (depending on the verb in the sentence) provide users a means to select the behavior described; it must also allow the user *not* to select it.

`match`

Case-insensitive match: two strings or names being compared must be identical except for differences between upper- and lower-case letters in scripts which have such a distinction. Characters with multiple possible representations in ISO 10646 (e.g. both precomposed and base+diacritic forms) match only if they have the same representation, except for case differences, in both strings. Case folding must be performed as specified in The Unicode Standard, Version 2.0, section 4.1; in particular, it is recommended that case-insensitive matching be performed by folding uppercase letters to lowercase, not vice versa.

`exact(ly) match`

Case-sensitive match: two strings or names being compared must be identical. Characters with multiple possible representations in ISO 10646 (e.g. both precomposed and base+diacritic forms) match only if they have the same representation in both strings.

`for compatibility`

A feature of XML included solely to ensure that XML remains compatible with SGML; the

expectation is that in many cases, those aspects of SGML that are not required to satisfy XML's requirements but mandated only to achieve conformance may be removed or replaced in the near future by the organizations that maintain that standard.

1.5 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.

S (white space) consists of one or more blank characters, carriage returns, line feeds, or tabs.

< 1 White space >

```
S :: (#x0020 | #x000a | #x000d | #x0009)
   =  +
```

For some purposes, characters are classified as letters, digits, or other characters:

< 2 Name >

```
Character :: [#x20-#xFFFFFFFF]   / any ISO 10646 32-bit
          =                      * code */
   Letter :: [#x41-#x5A] |       / Latin 1 upper and
          =  [#x61-#x7A]         * lowercase */
             | #xAA | #xB5 |
             #xBA
             | [#xC0-#xD6] |     / Latin 1 supplementary
             [#xD8-#xF6]         * letters */
             | [#xF8-#xFF]       / Latin 1 supplementary
                                 * letters */
             | [#x0100-#x017F]   / Extended Latin-A */
                                 *
             | [#x0180-#x0217]   / Extended Latin-B */
                                 *
```

```
              | [#x0250-#x1FFF]   / IPA extensions,
                                  * spacing modifiers,
                                    diacritics, Greek,
                                    Coptic, Cyrillic,
                                    Armenian, Hebrew, ...
                                    */
              | [#x3040-#x9FFF]   / CJK */
                                  *
              | [#xF900-#xFDFF]   / CJK compatibility
                                  * ideographs ... */
              | [#xFE70-#xFEFE]   / Arabic presentation
                                  * forms B */
              | [#xFF21-#xFF3A]   / Fullwidth Latin A-Z */
                                  *
              | [#xFF41-#xFF5A]   / Fullwidth Latin a-z */
                                  *
              | [#xFF66-#xFFDC]   / Halfwidth katakana,
                                  * hangul */
    Digit :: [#x0030-#x0039]      / Correct this table
          =                       * using section 4.5 of
                                    Unicode 2.0
                                    ISO 646 digits */
              | [#x0660-#x0669]   / Arabic-Indic digits */
                                  *
              | [#x06F0-#x06F9]   / Eastern Arabic-Indic
                                  * digits */
              | [#x0966-#x096F]   / Devanagari digits */
                                  *
              | [#x09E6-#x09EF]   / Bengali digits */
                                  *
              | [#x0A66-#x0A6F]   / Gurmukhi digits */
                                  *
              | [#x0AE6-#x0AEF]   / Gujarati digits */
                                  *
              | [#x0B66-#x0B6F]   / Oriya digits */
                                  *
```

```
                  | [#x0BE7-#x0BEF]   / Tamil digits (no zero)
                                      * */
                  | [#x0C66-#x0C6F]   / Telugu digits */
                                      *
                  | [#x0CE6-#x0CEF]   / Kannada digits */
                                      *
                  | [#x0D66-#x0D6F]   / Malayalam digits */
                                      *
                  | [#x0E50-#x0E59]   / Thai digits */
                                      *
                  | [#x0ED0-#x0ED9]   / Lao digits */
                                      *
                  | [#xFF10-#xFF19]   / Fullwidth digits
                                      * Ranges taken from Java
                                        documentation. Check
                                        against Unicode 2.0,
                                        section 4.6.
                                        N.B. not clear whether
                                        the relevant Greek and
                                        Hebrew letters should
                                        also be digits. Will
                                        matter for NUMBER
                                        attributes. */
```

A *Name* is a token beginning with a letter or hyphen and continuing with letters, digits, hyphens, or full stops (together known as name characters). The use of any name beginning with a string which matches "`-XML-`" in a fashion other than those described in this specification is a reportable error.

A *Number* is a sequence of digits. An *Nmtoken* (name token) is any mixture of name characters.

< 3 Names, Numbers, and Tokens >

```
    Name :: (Letter | '-') (Letter | Digit | '-' | '.')
         =  *
  Number :: Digit+
         =
 Nmtoken :: (Letter | Digit | '-' | '.')+
         =
Nmtokens :: Nmtoken (S Nmtoken)*
         =
```

Literal data is any quoted string containing neither the quotation mark used as a delimiter nor angle brackets. It may contain entity and character references.

< 4 Literals >

```
    Literal :: '"' [^"<>]* '"'
            =
               | "'" [^'<>]* "'"
QuotedCData :: '"' [^"<>]* '"'
            =
               | "'" [^'<>]* "'"
QuotedNames :: '"' Names '"' | "'" Names
            =  "'"
```

2. Documents

A textual object is an *XML Document* if it is either valid, or failing that, well-formed, as defined in this specification.

2.1 Logical and Physical Structure

Each XML document has both a logical and a physical structure.

Physically, the document is composed of units called entities; it begins in a "root" or document entity, which may refer to other entities, and so on ad infinitum. Entities referred to are embedded in the document at the point of reference.

The document contains declarations, elements, comments, entity references, character references, and processing instructions, all of which are indicated in the document by explicit markup. These

concepts and their markup are all explained elsewhere in this specification.

The two structures must be synchronous: tags and elements must each begin and end in the same entity, but may refer to other entities internally; comments, processing instructions, character references, and entity references must each be contained entirely within a single entity. Entities must each contain an integral number of elements, comments, processing instructions, and references, possibly together with character data not contained within any element in the entity, or else they must contain non-textual data, which by definition contains no elements.

2.2 Characters

The data stored in an XML entity is either text or binary. Binary data has an associated notation, identified by name; beyond a requirement to make available the notation name and the size in bytes of the binary data in a storage object, XML provides no information about and places no constraints on binary data. In fact, so-called binary data may in fact be textual, perhaps even well-formed XML text; but its identification as binary means that an XML processor need not parse it in the fashion described by the specification. XML text data is a sequence of characters. A character is A character is an atomic unit of text represented by a bit string; valid bit strings and their meanings are specified by ISO 10646.

Users may extend the ISO 10646 character repertoire, in the rare cases where this is necessary, by making use of the private use areas.

The mechanism for encoding character values into bit patterns may vary from entity to entity. All XML processors must accept the UTF-8 and UCS-2 encodings of 10646; the mechanisms for signalling which of the two are in use, and for bringing other encodings into play, are discussed later, in the discussion of character encodings.

Regardless of the specific encoding used, any character in the ISO 10646 character set may be referred to by the decimal or hexadecimal equivalent of its bit string:

< 5 Character references >

```
    Hex :: [0-9a-fA-F]
           =
   Hex4 :: Hex Hex Hex Hex
           =
CharRef :: '&#' Number ';'
           =
              | '&u-' Hex4
              ';'
```

2.3 Syntax of Text and Markup

XML text consists of intermingled character data and markup. Markup takes the form of start-tags, end-tags, empty elements, entity references, character references, comments, marked sections, document type declarations, and processing instructions. The simplest form of XML processor thus could parse a well-formed XML document using the following rules:

< 6 Trivial text grammar >

```
Trivial :: (PCDATA | Markup)*
        =
     Eq :: S? '=' S?
        =
 Markup :: '<' Name (S Name Eq      / start-tags */
        = QuotedCData)* S? '>'      *
           | '</' Name S? '>'       / end-tags */
                                    *
           | '<' Name (S Name Eq    / empty elements */
           QuotedCData)* S? '/>'    *
           | '&' Name ';'           / entity references */
                                    *
           | '&#' Number ';'        / character
                                    * references */
           | '&u-' Hex4 ';'         / character
                                    * references */
           | '<!--' [^-]*           / comments */
           ('-' [^-]+)* '-->'       *
           | '<![CDATA[' MsData     / marked sections */
           ']]>'                    *
           | '<!DOCTYPE' (Name |    / doc type
           S)+ ('[' [^]]* ']')?     * declaration */
           '>'
           | '<?' [^?]* ('?' [^>]   / processing
           +)* '?>'                 * instructions */
```

Most processors will require the more complex grammar given in the rest of this specification.

All text that is not markup constitutes the character data of the document.

The ampersand character (&) and the left and right angle bracket (< and >) may appear in their literal form *only* when used as markup delimiters, or within comments, processing instructions, or marked sections. If they are needed in the text, they must be represented using the strings "`&`", "`<`", and "`>`". Parsed character data is thus any string of characters which does not contain the start-delimiter of any markup. Character data is any string of characters not including the marked-section-close delimiter, "`]]>`". For convenience, the single-quote character (') may be represented as "`&sq;`", and the double-quote character (") as "`&dq;`".

< 7 Character Data >

```
PCDATA :: [^<&]*
       =
MsData :: [^]]* ((']' ([^]])) | (']]' [^>])) [^]]*)
       =  *
```

2.4 Comments

Comments may appear anywhere that character data may, except in a marked section (more properly, comments appearing in a marked section will not be recognized as such). They are not part of the document's character data; an XML processor may, but need not, make it possible for an application to retrieve the text of comments. An XML processor must inform the application of the length of comments if they are not passed through, to enable the application to keep track of the correct location of objects in the XML document. For compatibility, the string -- (double-hyphen) may not occur within comments.

< 8 Comments >

```
Comment :: '<!--' [^-]* ('-' [^-]+)* '--
        =  >'
```

2.5 Processing Instructions

Processing instructions, usually referred to as PIs, allow the XML processor to pass instructions directly to selected applications.

< 9 Processing Instructions >

```
PI :: '<?' Name S [^?]* ('?' [^>]+)* '?
    = >'
```

PIs are not part of the document's character data, but must be passed through to the application. The Name which follows the '?' at the beginning of the PI is called the *PI target*. It is normally the name of a declared notation, identifying the application to which it belongs. The use of the PI target "XML" in any other way other than those described in this specification is a reportable error.

2.6 Marked Sections

Marked sections can occur anywhere character data may occur; they are used to escape blocks of text which may contain characters which would otherwise be recognized as markup. Marked sections begin with the string `<![CDATA[` and end with the string `]]>`:

< 10 Marked Sections >

```
      MS :: MsStart MsData
         = MsEnd
 MsStart :: '<![CDATA['
         =
   MsEnd :: ']]>'
         =
```

Within a marked section, only the *MsEnd* string is recognized, so that angle brackets and ampersands may occur in their literal form and need not be escaped using `<`, etc. Marked sections cannot nest.

2.7 White Space Handling

While authoring XML documents, it is often convenient to use "white space" (spaces, tabs, blank lines, denoted by the nonterminal S in this specification) to set apart the markup for greater readability. Such white space is typically not intended for inclusion in the delivered version of the document. On the other hand, "significant" white space, to be retained in the delivered version, is common; for example in poetry or source code.

An XML processor must provide two distinct white space handling modes, *COLLAPSE* and *KEEP*, and have the ability to apply these modes on a per-element basis. They operate as follows:

`COLLAPSE`

The XML processor must suppress (i.e. not pass to the application) all white space in an

element which immediately follows the start-tag and all that which immediately precedes the end-tag. In the element's character data, it must convert all sequences of white space characters to a single space (#x0020) character, before passing the data to the application.

KEEP
: The XML processor must suppress initial line break characters which immediately follow the start-tag of the element, and which immediately precede its end-tag. All other characters in the character data of the element must be passed to the application without change.

The white space handling mode is signaled through the use of a reserved attribute, whose declaration is as follows:

```
<!ATTLIST * -XML-SPACE (KEEP|COLLAPSE) #IMPLIED>
```

where the "*" signifies that this attribute may apply to any element.

The value of the attribute sets the white space handling mode for the element and for any contained elements. Unless otherwise specified, an XML processor is to set the white space handling mode for the root element of a document to *COLLAPSE*.

2.8 Prolog and Document Type Declaration

The function of the markup in an XML document is to describe its storage and logical structures, and associate attribute-value pairs with the logical structure. XML provides a mechanism, the document type declaration, to define constraints on that logical structure, and to support the use of predefined storage units. An XML document is said to be *valid* if there is an associated document type declaration and if the document complies with the constraints expressed in it.

The document type declaration must appear before the first element in the document.

< 11 XML document >

```
 document :: Prolog element Misc*
          =
   Prolog :: EncodingDecl? Misc* RMDecl? Misc*
          = (doctypdecl | DtdSummary)? Misc*
     Misc :: Comment | PI | S
          =
```

For example, the following is a complete XML document, well-formed but not valid:

```
<greeting>Hello, world!</greeting>
```

The XML *document type declaration* may include a pointer to an external entity containing a subset of the necessary markup declarations, and may also directly include another, internal, subset of the necessary markup declarations.

< **12 Document type declaration** >

```
   doctypedecl :: '<!DOCTYPE' S Name Extid? S? ('['
               = internalsubset* ']' S?)? '>'
internalsubset :: elementdecl | AttlistDecl | EntityDecl
               =
                  | NotationDecl | DtdSummary | S |
                  Comment
```

These two subsets, taken together, are properly referred to as the *document type definition*, abbreviated *DTD*. However, it is a common practice for the bulk of the markup declarations to appear in the external subset, and for this subset, usually contained in a file, to be referred to as "the DTD" for a class of documents. For example:

```
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

The system identifier `hello.dtd` indicates the location of a full DTD for the document.

The declarations can also be given locally, as in this slightly larger example:

```
<?XML encoding="UTF-8">
<!DOCTYPE greeting [
<!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

The character-set label `<?XML encoding="UTF-8">` indicates that the document entity is encoded using the UTF-8 transformation of ISO 10646. The legal values of the character set code are given in the discussion of character encodings.

Individual markup declaration types are described elsewhere in this specification.

2.9 Required Markup Declaration

In many cases, an XML processor can read an XML document and accomplish useful tasks without having first processed the entire DTD. However, certain declarations can substantially affect the actions of an XML processor. A document author can communicate whether or not DTD processing is necessary using a *required markup declaration* (abbreviated RMD) processing instruction:

< 13 Required markup declaration >

```
RMDecl :: '<?XML' S 'RMD' Eq ('NONE' | 'INTERNAL' |
        = 'ALL') S? '?>'
```

In an RMD, the value *NONE* indicates that an XML processor can parse the containing document correctly without first reading any part of the DTD. The value *INTERNAL* indicates that the XML processor must read and process the internal subset of the DTD to parse the containing document correctly. The value *ALL* indicates that the XML processor must read and process the declarations in both the subsets of the DTD to parse the containing document directly.

The RMD must indicate that the entire DTD is required if the external subset contains any declarations of

1. undistinguished empty elements, and these elements occur in the document, or
2. attributes with default values, and elements to which these attributes apply appear in the document, or
3. entities, and references to those entities appear in the document.

If such declarations occur in the internal but not the external subset, the RMD should take the value *INTERNAL*. It is an error to specify *INTERNAL* if the external subset is required, or to specify *NONE* if the internal or external subset is required.

If no RMD is provided, the effect is identical to an RMD with the value *ALL*.

3. Logical Structures

Each XML document contains one or more *elements*, the boundaries of which are either delineated by start-tags and end-tags, or, for empty elements, are limited to the start-tag. Each element has a type, identified by name (sometimes called its *generic identifier* or *GI*), and may have a set of attributes. Each attribute has a name and a value.

This specification does not constrain the semantics, use, or (beyond syntax) names of the elements and attributes.

3.1 Start- and End-Tags

The beginning of every XML element is marked by a *start-tag*.

< 14 Start-tag Recognition >

```
      STag :: '<' Name (S Attribute)
           =  * S? '>'
 Attribute :: Name Eq QuotedCData      [ VC: Attribute
           =                             Value Type ]
```

The *Name* in the start- and end-tag rules gives the element's *type*. The *Name-QuotedCData* pairs are referred to as the *attributes* of the element, with the *Name* referred to as the *attribute name* and the content of the *QuotedCData* (the characters between the "'" or "" delimiters) as the *attribute value*.

Validity Constraint - Attribute Value Type:
Attribute values must be of the type declared for the attribute. (For attribute types, see the discussion of attribute declarations.)

The end of every element which is not empty is marked by an *end-tag*:

< 15 End-tag Recognition >

```
ETag :: '</' Name S?
     =  '>'
```

The Name, once again, gives the element's type.

If an element is *empty*, the start-tag constitutes the whole element. An empty element takes a special form:

< 16 Tags for empty elements >

```
EmptyElement :: '<' Name (S Attribute)* S? '/
             =  >';
```

For compatibility, an empty element may have the same syntax as a start-tag; in this case, it cannot be recognized based on syntax, but must be declared as being empty. Such elements are called *undistinguished empty elements*.

The text between the start-tag and end-tag is called the element's *content*:

< 17 Content of elements >

```
content ::= (element | PCDATA | MS |
        PI | Comment)*
element ::= EmptyElement                    / empty elements
                                            * */
          | STag content ETag               [ WFC: GI Match ]
```

Well-Formedness Constraint - GI Match:
The *Name* in an element's end-tag must match that in the start-tag.

3.2 Well-Formed XML Documents

A textual object is said to be a *well-formed* XML document if, first, it matches the production above labeled *XML Document*, and if:

1. There are no undistinguished empty elements which have not been specified as such in an element declaration.
2. For each entity reference which appears in the document, the entity name has been declared in the document type declaration.

Matching the "XML Document" production implies that:

1. It contains one or more elements.
2. There is one element, called the *root*, for which neither the start-tag nor the end-tag are in the content of any other element. For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delineated by start- and end-tags, nest within each other properly.

As a consequence of this, for each non-root element *C*, there is one other element *P* such that *C* is in the content of *P*, but is not in the content of any other element that is in the content of *P*. Then *P* is referred to as the *parent* of *C*, and *C* as the *child* of *P*.

3.3 Element Declaration

The element structure of an XML document may be declared fully or partially. Such declarations serve two purposes:

1. To establish a set of structural constraints, i.e. a grammar, for a class of documents, and to verify that documents are valid, i.e. comply with that grammar.
2. To make XML documents well-formed by declaring undistinguished empty elements.

An element declaration constrains the element's type and its content. The content constraints will be described first; four forms are available: empty, any, mixed content, and element content.

Declarations often contain references to element types, for example when constraining which element types can appear as children of others, and which attributes may be attached to which element types. At user option, an XML processor may issue a warning when no element is declared whose type matches that given, but this is not an error.

An *element declaration* takes the form:

< 18 Element declarations >

```
elementdecl :: '<!ELEMENT' S        [ VC: Unique Element
            =  Name S ('EMPTY' |      Declaration ]
               'ANY' | Mixed |
               elements ) S? '>'
```

where *Name* identifies the type of the element.

Validity Constraint - Unique Element Declaration:
No element type may be declared more than once.

For compatibility, an element type may be declared using the keyword EMPTY; this is an undistinguished empty element.

If an element type is declared using the keyword ANY, then there are no constraints on its content aside from its being well-formed: it may contain subelements of any type and number, interspersed with character data.

3.3.1 Mixed Content

An element type may be declared to contain mixed content, that is text comprising character data optionally interspersed with child elements. In this case, the types of the child elements are constrained, but not their order nor their number of occurrences:

< 19 Mixed-content declaration >

```
Mixed :: '(' S? '#PCDATA' ( S? '|' S? Name )* S? ')'
      =  *'
         | '(' S? '#PCDATA' S? ')'
```

where the *Name*s give the types of elements that may appear as children.

3.3.2 Element Content

An element type may be declared to have element content, consisting only of other elements. In this case, the constraint includes a content model, a context-free grammar governing the allowed types of the child elements and the order in which they appear. The grammar is built on content particles (CPs), which consist of names, choice lists of content particles, or sequence lists of content particles:

< 20 Element-content models >

```
elements :: cp
         =
      cp :: (Name | choice | seq) ('?' | '*' |
         =  '+')?
     cps :: S? cp S?
         =
     seq :: '(' cps (',' cps)* ')'
         =
  choice :: '(' cps ('|' cps)+ ')'
         =
```

where each *Name* gives the type of an element which may appear as a Child. Any content particle in a choice list may appear in the element content at the appropriate location; content particles occurring in a sequence list must each appear in the element content in the order given. The optional character following a name or list governs whether the element or the content particles in the list may occur one or more, zero or more, or zero or one times respectively. The syntax and meaning is identical to that used in the productions in this specification.

3.4 Attribute Declaration

Attributes are used to associate name-value pairs with elements. Attributes may appear only within start-tags; thus, the productions used to recognize them appear in that discussion. Attribute declarations may be used:

1. To define the set of attributes pertaining to a given element type.
2. To establish a set of type constraints on these attributes.
3. To provide default values for atributes.

Attribute-list declarations specify the name, data type, and default value (if any) of each attribute associated with a given element type:

< 21 Attribute list declaration >

```
AttlistDecl :: '<!ATTLIST' S          [ VC: Unique Attribute-
            =  Name AttDef+ S?          List Declaration ]
               '>'
     AttDef :: S Name S               [ VC: Unique Attribute
            =  AttType S                Name ]
               Default
```

The *Name* in the *AttlistDecl* rule is the type of an element; it is a reportable error to declare attributes for an element type not itself declared. The *Name* in the *AttDef* rule is the name of the attribute.

Validity Constraint - Unique Attribute-List Declaration:
At most one attribute-list declaration may be provided for a given element type.

Validity Constraint - Unique Attribute Name:
An attribute name may appear at most once in an attribute-list declaration.

3.4.1 Attribute Types

XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The string type may take any literal string as a value; the tokenized types have varying lexical and semantic constraints, as noted:

< 22 Attribute types >

```
      AttType :: StringType |
                 TokenizedType |
                 EnumeratedType
    StringType :: 'CDATA'
                 =
 TokenizedType :: 'ID'                    [ VC: ID ]
                 =
                    | 'IDREF'             [ VC: Idref ]
                    | 'IDREFS'            [ VC: Idref ]
                    | 'ENTITY'            [ VC: EntityName ]
                    | 'ENTITIES'          [ VC: EntityName ]
                    | 'NMTOKEN'           [ VC: Name Token ]
                    | 'NMTOKENS'          [ VC: Name
                                            Tokens ]
```

Validity Constraint - ID:
Values of this type must be valid Names. A name must not appear more than once in an XML document as a value of this type: i.e. ID values must uniquely identify the elements which bear them.

Validity Constraint - Idref:
Values of this type must be one (for IDREFS one or more) Names, which must each match the value of an ID attribute on some element in the XML document; i.e. IDREF values must match some ID.

Validity Constraint - Entity Name:
Values of this type must be one (for ENTITIES one or more) Names, which must each exactly match the name of an external binary entity declared in the XML DTD.

Validity Constraint - Name token:
Values of this type must consist of a string matching the `Nmtoken` nonterminal of the grammar defined in this specification. The XML processor must normalize the case of the attribute value before passing it to the application.

Validity Constraint - Name tokens:
Values of this type must consist of a string matching the `Nmtokens` nonterminal of the grammar defined in this specification. The XML processor must reduce white space to single blanks, and normalize the case of the attribute value, before passing it to the application.

Enumerated attributes can take one of a list of values provided in the declaration; there are two types:

< 23 Enumerated attribute declarations >

```
EnumeratedType :: NotationType |
                  = Enumeration
  NotationType :: 'NOTATION' S '('    [ VC: Notation
                  = S? Name (S? '|'     Attributes ]
                    S? Name)* S? ')'
   Enumeration :: '(' S? Nmtoken      [ VC: Enumeration ]
                  = (S? '|' S?
                    Nmtoken)* S? ')'
```

Validity Constraint - Notation Attributes:
The names in the declaration of NOTATION attributes must be names of declared notations (see the discussion of notations). Values of this type must match one of the notation names included in the declaration.

Validity Constraint - Enumeration:
Values of this type must match one of the *Nmtoken* tokens in the declaration.

3.4.2 Attribute Defaults

An attribute declaration may provide information on whether the attribute's presence is required, and if not, how an XML processor should react if a declared attribute is absent in a document:

< 24 Attribute defaults >

```
Default :: '#REQUIRED'
        =
           | '#IMPLIED'
           | ( '#FIXED'?
           Literal)
```

#REQUIRED means that it is a reportable error should the processor encounter a start-tag where this attribute is omitted, i.e. could occur but does not. #IMPLIED means that if an attribute is omitted, the XML processor must inform the application that no value was specified; no constraint is placed on the behavior of the application.

If the attribute is neither #REQUIRED nor #IMPLIED, then the *Literal* value contains the declared *default* value. If the #FIXED is present, it is a reportable error if the attribute is present with a

different value from the default. If a default value is declared, when an XML processor encounters an omitted attribute, it is to assume that the attribute is present and has the declared default value.

3.5 Partial DTD Information

The prolog of an XML document may contain an abbreviated summary of the DTD for the convenience of non-validating processors.

< 25 DTD summary >

```
  DtdSummary :: (EmptyInfo | TextInfo | NoText | IdInfo |
             = DefaultInfo)*
   EmptyInfo :: '<?XML' S 'empty' S 'names' Eq
             = QuotedNames S? '?>'
    TextInfo :: '<?XML' S 'text' S 'names' Eq QuotedNames
             = S? '?>'
      NoText :: '<?XML' S 'notext' S 'names' Eq
             = QuotedNames S? '?>'
      IdInfo :: '<?XML' S 'idinfo' S
             =
                'ids' Eq QuotedPairs S
                'refs' Eq QuotedPairs S? '?>'
 QuotedPairs :: '"' Pairs '"' | "'" Pairs "'"
             =
       Pairs :: ('*' | Name) S Name (S ('*' | Name) S
             = Name)*
 DefaultInfo :: '<?XML' S 'default' Name (S Name Eq
             = Literal)* S? '?>'
```

The *empty*, *text*, and *notext* declarations give, respectively, lists of element types declared as empty, mixed-content, or element-content elements. The *idinfo* declaration lists attributes declared as id or idref(s) (in the *ids* and *refs* values, respectively). Each attribute name is given as an element-attribute pair; an asterisk matches all element types which have attributes of the given name. The *default* declaration specifies default values for the attributes of a given element type.

For example:

```
<?XML empty names="ptr xptr pb" ?>
<?XML notext names="div0 div1 div2 div3 list" ?>
<?XML text names="p emph q title hi" ?>
<?XML idinfo ids='* id' refs='* target * targets' ?>
<?XML default div type='section' ?>
```

Elements not listed as `notext` are implicitly declared as `text` elements; the *text* declaration is thus not strictly necessary.

4. Physical Structures

An XML document may consist of one or many storage units. The unit of storage is the *entity*; entities are identified by name and have *content*. Each XML document has one entity called the document entity, which serves as the starting point for the XML processor (and may contain the whole document).

Entitities may be either binary or text. A text entity contains text data which is to be considered as an integral part of the document. A binary entity contains binary data identified by notation. Text and binary entities cannot be distinguished by syntax; their types are established in their declarations, described below.

4.1 Character and Entity References

A *character reference* refers to a specific character in the ISO 10646 character set, often one not directly accessible from available input devices:

< 26 Character Reference >

```
CharRef :: '&#' Number ';'
        =
           | '&u-' Hex4
           ';'
```

An *entity reference* refers to the content of a named entity.

< 27 Entity Reference >

```
Reference :: EntityRef |
          = CharRef
EntityRef :: '&' Name ';'          [ WFC: Entity
          =                          Declared ]
                                   [ WFC: Text Entity ]
                                   [ WFC: No Recursion ]
```

Well-Formedness Constraint - Entity Declared:
The *Name* given in the entity reference must match exactly the name given in the declaration of the entity.

Well-Formedness Constraint - Text Entity:
An entity reference may not contain the name of a binary entity. Binary entities may be referred to only in attribute values declared to be of type ENTITY or ENTITIES.

Well-Formedness Constraint - No Recursion:
A text entity may not contain a reference to itself.

4.2 Declaring Entities

Entities are declared thus:

< 28 Entity declaration >

```
EntityDecl :: '<!ENTITY' S Name EntityDef S?
           =  '>'
 EntityDef :: Literal | ExternalDef;
           =
```

The *Name* is that by which the entity is invoked by exact match in an entity reference.

4.2.1 Internal Entities

If the entity definition is just a *Literal*, this is called an *internal* entity - there is no separate storage unit, and the content of the entity is given in the declaration. An internal entity is a text entity.

4.2.2 External Entities

If the entity is not internal, it is an *external entity*, declared as follows:

< 29 External entity declaration >

```
ExternalDef ::= NDataDecl? 'SYSTEM'
                Literal
  NDataDecl ::= 'NDATA' S Name S      [ VC: Notation
                                        Declared ]
```

If the *NDataDecl* is present, this is a binary data entity, otherwise a text entity.

Validity Constraint - Notation Declared:
The *Name* must match one of those in a notation declaration.

The *Literal* that follows the keyword SYSTEM called the entity's *system identifier*. It is a URL, which may be used to retrieve the content of the entity.

4.2.3 Character Encoding in Entities

Each text entity in an XML document may use a different encoding for its characters. All XML processors must be able to read entities in either the UTF-8 or UCS-2 encodings. It is recognized that for for some advanced work, particularly with Asian languages, the use of the UTF-16 encoding is required, and correct handling of this encoding is a desirable characteristic in XML processor implementations.

Entities encoded in UCS-2 must begin with the Byte Order Mark described by ISO 10646 Annex E and Unicode Appendix B (the ZERO WIDTH NO-BREAK SPACE character, U+FEFF) -- this is an encoding signature, not part of either the markup or character data of the XML document. XML processors must be able to use this character to differentiate between UTF-8 and UCS-2 encoded documents.

Although an XML processor is only required to read entities in the UTF-8 and UCS-2, it is recognized that many other encodings are in daily use around the world, and it may be advantageous for XML processors to read entities that use these non-standard encodings. For this purpose, XML provides an encoding declaration processing instruction, which must appear at the beginning of an entity, before any other character data or markup:

< 30 Encoding declaration >

```
Encodingdecl :: '<?XML' S 'encoding' Eq
             =  QEncoding S? '?>'
   QEncoding :: '"' Encoding '"' | "'"
             =  Encoding "'"
    Encoding :: 'UTF8' | 'UTF16' | 'UCS2' |  / Unicode */
             =  'UCS4'                       *
                | 'ISO8859' ('-' [1-9])?     / 8-bit */
                                             *
                | 'Shift-JIS' | 'EUC-JIS' |  / Japanese
                'New-JIS'                    * */
```

It is a reportable error for an entity including an encoding declaration to be stored in an encoding other than that named in the declaration.

An entity which begins with neither a Byte Order Mark nor an encoding declaration must be in the UTF-8 encoding.

While XML provides mechanisms for distinguishing encodings, it is recognized that in a heterogeneous networked environment, there is often difficulty in reliably signaling the encoding of an entity. Errors in this area fall into two categories:

1. failing to read an entity because of inability to recognize its actual encoding, and
2. reading an entity incorrectly because of an incorrect guess of its proper encoding.

The first class of error is extremely damaging, and the second class is extremely unlikely. For these reasons, XML processors should make an effort to use all available information, internal and external, to aid in detecting an entity's correct encoding. Such information may include, but is not limited to:

1. Using information from an HTTP header
2. Using a MIME header obtained other than through HTTP
3. Metadata provided by the native OS filesystem or through other mechanisms
4. Analyzing the bit patterns at the front of an entity to determine if the application of any known encoding yields a valid encoding declaration.

If an XML processor encounters an entity with an encoding that it is unable to process, it must inform the application of this fact and allow the application to request either that the entity should be treated as an binary entity, or that processing should cease.

4.2.4 The Document Entity

The *document entity* serves as the root of the entity tree and a starting-point for an XML processor. This specification does not specify how the document entity is to be located by an XML processor; unlike other entities, the document entity might well appear on an input stream of the processor without any identification at all.

4.3 XML Processor Treatment of Entities

When an XML processor encounters a character reference or an entity name (in a reference or attribute value):

1. In all cases, the XML processor must inform the application of the reference's occurrence and its identifier (for an entity reference, the name; for a character reference, the "`&...`" string.)
2. For both character and entity references, the processor must remove the reference itself from the text data before passing the data to the application.
3. For character references, the processor must insert the indicated ISO 10646 bit pattern into the text data at the location of the reference before passing it to the application.
4. For an external entity reference, the processor must inform the application of the entity's system identifier.
5. If the external entity is binary, the processor must inform the application of the associated notation name, and the notation's associated system identifier.
6. For an internal (text) entity, the processor must *include* the entity; that is, retrieve its content, and treat the content as a part of the text data of the XML document, processing this replacement text data (which may contain both text and markup) to determine what data to pass to the application. Since the entity's content text may contain other entity references, an XML processor may have to repeat the inclusion process recursively.
7. If the entity is an external text entity, then in order to validate the XML document, the processor must include the content of the entity.
8. If the entity is an external text entity, and the processor is not attempting to validate the XML document, the processor may, but need not, include the entity's content. This rule is based on the recognition that the inclusion semantic provided by the SGML and XML text entity mechanism, primarily designed to support modularity in authoring, may not be appropriate for other applications, in particular document browsing. Browsers, for example, when encountering an external text entity reference, might choose to provide a visual indication of the entity's presence and retrieve it for display only on demand. While this behavior would not allow the application to validate the document, it is compliant with this specification.

4.4 Notation Declaration

Notations identify by name the format of external binary entities.

Notation declarations provide a name for the notation, for use in entity and attribute declarations and in attribute-value specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.

< 31 Notation declarations >

```
NotationDecl :: '<!NOTATION' S Name S Extid S?
                =  '>'
```

XML processors must provide applications with the name and external identifier of any notation declared and referred to in an attribute value, attribute definition, or entity declaration. They may additionally resolve the external identifier into the system identifier, file name, API address, or other information needed to allow the application to call a processor for data in the notation described. (It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.)

5. Conformance

Conforming XML processors fall into two classes: validating and non-validating.

Validating and non-validating systems alike must report violations of the well-formedness constraints given in this specification.

Validating systems must report locations in which the document does not comply with the constraints expressed by the declarations in the DTD. They must also report all failures to fulfill the validity constraints given in this specification.

A. XML and SGML

XML is designed to be a subset of SGML, in that every valid XML document should also be a valid SGML document, using the same DTD, and that the parse trees produced by an SGML parser and an XML processor should be the same. To achieve this, XML was defined by removing features and options from the specification of SGML.

Despite this, there are a small number of cases where XML fails to be a pure subset of SGML, including:

1. XML's white space handling rules are much less elaborate than those of SGML. One effect is that for elements using the KEEP mode for white-space handling, an XML processor will pass through a few record-separator markers that an SGML processor will eat.
2. XML defines, for documents, the property of being well-formed; this does not really correspond to any SGML concept.

The following list describes features which are available in SGML but not in XML. It may not be complete.

1. Tag omission
2. The CONCUR, LINK, DATATAG, and SHORTREF features
3. The "&" connector in content models
4. Inclusions and exclusions in content models
5. CURRENT, CONREF, NAME, NAMES, NUMBER, NUMBERS, NUTOKEN, and NUTOKENS declarations for attributes
6. The NET construct
7. Abstract syntax
8. Capacities and quantities
9. Comments appearing within other markup declarations
10. Public Identifiers
11. Omission of quotes on attribute values

B. References

ISO/IEC 8879

ISO (International Organization for Standardization). *ISO/IEC 8879-1986 (E). Information processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML).* First edition -- 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO/IEC 10646

ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 1993.

ISO/IEC 10744

ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology -- Hypermedia/Time-based Structuring Language (HyTime).* [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe.* [Geneva]: International Organization for Standardization, 1996. (?)

Unicode

The Unicode Consortium. *The Unicode Standard: Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press, 1996.

C. Working Group and Editorial Review Board Membership

C.1 Working Group

This specification was prepared with the assistance and participation of the W3C SGML Working Group. Publication of this specification does not necessarily imply that all members of the working group agree with its content. At the time this specification was prepared, the W3C SGML Working Group comprised the following members.

1. Sharon Adler, EBT (sca@ebt.com)
2. Paula Angerstein, Texcel (paula@texcel.no)
3. Todd Bauman, U. of Maryland (bbauma1@cs.umbc.edu)
4. Anders Berglund, EBT (alb@ebt.com)
5. Karl Best, Novell (kbest@novell.com)
6. Michel Biezunski, High Text (michel@hightext.com)
7. Harvey Bingham, SGML consultant (hbingham@acm.org)
8. Steve Brown, InfoObjects (sbrown@cortland.com)
9. Martin Bryan, SGML Centre (mtbryan@sgml.u-net.com)
10. Mark Buckley, Microsoft (mbuckley@microsoft.com)
11. Len Bullard, Lockheed-Martin (cbullard@HiWAAY.net)
12. Lou Burnard, Oxford University (lou@vax.ox.ac.uk)
13. Steve Byrne, JavaSoft (steve.byrne@sun.com)
14. Kurt Conrad, Sagebrush Group (conrad@cbvcp.com)
15. Paul Cope, auto-graphics (prc@auto-graphics.com)
16. Keith Corbett, Harlequin (kmc@harlequin.com)
17. Robin Cover (robin@acadcomp.sil.org)
18. Derek Denny-Brown, TechnoTeacher (derdb@techno.com)
19. Dave Durand, Boston University (dgd@cs.bu.edu)
20. Carol Ellerbeck, INSO (Carol_Ellerbeck@inso.com)
21. Joe English, CR Laboratories (jenglish@crl.com)
22. Ralph Ferris, Fujitsu OSSI (ralph@ossi.com)
23. Lee Fogal, Digital Equipment (fogal@zk3.dec.com)
24. Todd Freter, Novell (tfreter@novell.com)
25. Matthew Fuchs, Disney Imagineering (matt@wdi.disney.com)
26. Charles Goldfarb (charles@sgmlsource.com)
27. Paul Grosso, ArborText (paul@arbortext.com)
28. Eduardo Gutentag, SunSoft (eduardo@eng.sun.com)
29. Hasse Haitto, Synex (haitto@synex.se)
30. Catherine Hamon, Hightext (hamon@hightext.com)
31. Ken Holman, Microstar (gkholman@microstar.com)
32. Rick Jelliffe, Allette Systems (ricko@allette.com.au)
33. Alan Karben, Wall Street Journal (karben@interactive.wsj.com)
34. Pär Karlsson, Ericsson (etxpkar@stdoca.ericsson.se)
35. Bill Lindsey, BDM (blindsey@BDMTech.com)
36. Per Åke Ling, Ericsson (etxperl@stdoca.ericsson.se)
37. Debbie Lapeyre, Atlis (dlapeyre@netcom.com)
38. Megan MacMillan, BDM Technologies (megan@bdmtech.com)
39. Christopher Maden, EBT (crm@ebt.com)
40. James Mason, ORNL (masonjd@ornl.gov)
41. Alex Milowski, Copernican Solutions (alex@copsol.com)
42. Steve Newcomb, TechnoTeacher (srn@techno.com)
43. Gavin Nicol, EBT (gtn@ebt.com)
44. Nancy Paisner, Hitachi (nancy@hi.com)
45. Gary Palmer, ActiveSystems (gpalmer@sonetis.com)
46. Dave Peterson, SGML Works (davep@acm.org)

47. Paul Prescod, U. of Waterloo (papresco@calum.csclub.uwaterloo.ca)
48. Lynne Price, SGML consultant (lprice@ix.netcom.com)
49. Arjun Ray, Q2-II (aray@nmds.com)
50. Bill Smith, SunSoft (bill.smith@sun.co)
51. Bob Stayton, SCO (bobs@sco.com)
52. Robert Streich, Schlumberger (streich@slb.com)
53. Jeff Suttor, SunSoft (jeff.suttor@sun.com)
54. James Tauber, U. of Western Australia (jtauber@library.uwa.edu.au)
55. Wayne Taylor, Novell (wtaylor@novell.com)
56. Henry Thompson, University of Edinburgh (ht@cogsci.ed.ac.uk)
57. Sam Wilmott, OmniMark (s.wilmott@omnimark.com)
58. Chris Wilson, Microsoft (cwilso@microsoft.com)
59. Wayne Wohler, IBM (wohler@vnet.ibm.com)
60. Lauren Wood, SoftQuad (lauren@sqwest.bc.ca)

C.2 Editorial Review Board

This specification was prepared and approved for publication by the W3C SGML Editorial Review Board (ERB). ERB approval of this specification does not necessarily imply that all ERB members voted for its approval. At the time it approved this specification, the SGML ERB had the following members:

1. Jon Bosak, Sun (jon.bosak@sun.com), chair
2. Tim Bray, Textuality (tbray@textuality.com), editor
3. James Clark (jjc@jclark.com), technical lead
4. Dan Connolly (connolly@w3.org), W3C contact
5. Steve DeRose, EBT (sjd@ebt.com)
6. Dave Hollander, HP (dmh@hpsgml.fc.hp.com)
7. Eliot Kimber, Passage Systems (kimber@passage.com)
8. Tom Magliery, NCSA (mag@ncsa.uiuc.edu)
9. Eve Maler, ArborText (elm@arbortext.com)
10. Jean Paoli, Microsoft (jeanpa@microsoft.com)
11. Peter Sharpe, SoftQuad (peter@sqwest.bc.ca)
12. C. M. Sperberg-McQueen, U. of Ill. at Chicago (cmsmcq@uic.edu), editor

REC-html32

HTML 3.2 Reference Specification

W3C Recommendation *14-Jan-1997*

Author: *Dave Raggett* <dsr@w3.org>

Status of this document

This document has been reviewed by W3C members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/pub/WWW/TR/.

Abstract

The HyperText Markup Language (HTML) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of applications. This specification defines HTML version 3.2. HTML 3.2 aims to capture recommended practice as of early '96 and as such to be used as a replacement for HTML 2.0 (RFC 1866).

Contents

Introduction to HTML 3.2

HTML 3.2 is W3C's specification for HTML, developed in early `96 together with vendors including IBM, Microsoft, Netscape Communications Corporation, Novell, SoftQuad, Spyglass, and Sun Microsystems. HTML 3.2 adds widely deployed features such as tables, applets and text flow around images, while providing full backwards compatibility with the existing standard HTML 2.0.

W3C is continuing to work with vendors on extensions for accessibility features, multimedia objects, scripting, style sheets, layout, forms, math and internationalization. W3C plans on incorporating this work in further versions of HTML.

HTML as an SGML Application

HTML 3.2 is an SGML application conforming to International Standard ISO 8879 -- Standard Generalized Markup Language. As an SGML application, the syntax of conforming HTML 3.2 documents is defined by the combination of the SGML declaration and the document type definition (DTD). This specification defines the intended interpretation of HTML 3.2 elements, and places further constraints on the permitted syntax which are otherwise inexpressible in the DTD.

The SGML rules for record boundaries are tricky. In particular, a record end immediately following a start tag should be discarded. For example:

```
<P>
Text
```

is equivalent to:

```
<P>Text
```

Similarly, a record end immediately preceding an end tag should be discarded. For example:

```
Text
</P>
```

is equivalent to:

```
Text</P>
```

Except within literal text (e.g. the PRE element), HTML treats contiguous sequences of white space characters as being equivalent to a single space character (ASCII decimal 32). These rules allow authors considerable flexibility when editing the marked-up text directly. Note that future revisions to HTML may allow for the interpretation of the horizontal tab character (ASCII decimal 9) with respect to a tab rule defined by an associated style sheet.

SGML entities in PCDATA content or in CDATA attributes are expanded by the parser, e.g. `&#233;` is expanded to the ISO Latin-1 character decimal 233 (a lower case letter e with an acute accent). This could also have been written as a named character entity, e.g. `&eacute;`. The & character can be included in its own right using the named character entity `&`.

HTML allows CDATA attributes to be unquoted provided the attribute value contains only letters (a to z and A to Z), digits (0 to 9), hyphens (ASCII decimal 45) or, periods (ASCII decimal 46). Attribute values can be quoted using double or single quote marks (ASCII decimal 34 and 39 respectively). Single quote marks can be included within the attribute value when the value is delimited by double quote marks, and vice versa.

Note that some user agents require attribute minimisation for the following attributes: `COMPACT`, `ISMAP`, `CHECKED`, `NOWRAP`, `NOSHADE` and `NOHREF`. These user agents don't accept syntax such as `COMPACT=COMPACT` or `ISMAP=ISMAP` although this is legitimate according to the HTML 3.2 DTD.

The SGML declaration and the DTD for use with HTML 3.2 are given in appendices. Further guidelines for parsing HTML are given in WD-html-lex.

The Structure of HTML documents

HTML 3.2 Documents start with a <!DOCTYPE> declaration followed by an HTML element containing a HEAD and then a BODY element:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
<TITLE>A study of population dynamics</TITLE>
... other head elements
</HEAD>
<BODY>
... document body
```

```
</BODY>
</HTML>
```

In practice, the HTML, HEAD and BODY start and end tags can be omitted from the markup as these can be inferred in all cases by parsers conforming to the HTML 3.2 DTD.

Every conforming HTML 3.2 document **must** start with the <!DOCTYPE> declaration that is needed to distinguish HTML 3.2 documents from other versions of HTML. The HTML specification is not concerned with storage entities. As a result, it is not required that the document type declaration reside in the same storage entity (i.e. file). A Web site may choose to dynamically prepend HTML files with the document type declaration if it is known that all such HTML files conform to the HTML 3.2 specification.

Every HTML 3.2 document must also include the descriptive title element. A minimal HTML 3.2 document thus looks like:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<TITLE>A study of population dynamics</TITLE>
```

Note: the word "Final" replaces "Draft" now that the HTML 3.2 specification has been ratified by the W3C member organizations.

The HEAD element

This contains the document head, but you can always omit both the start and end tags for HEAD. The contents of the document head is an unordered collection of the following elements:

- The TITLE element
- The STYLE element
- The SCRIPT element
- The ISINDEX element
- The BASE element
- The META element
- The LINK element

```
<!ENTITY % head.content "TITLE & ISINDEX? & BASE?">
<!ENTITY % head.misc "SCRIPT|STYLE|META|LINK">

<!ELEMENT HEAD O O  (%head.content) +(%head.misc)>
```

The %head.misc entity is used to allow the associated elements to occur multiple times at arbitrary

positions within the HEAD. The following elements can be part of the document head:

TITLE defines the document title, and is always needed.
ISINDEX for simple keyword searches, see PROMPT attribute.
BASE defines base URL for resolving relative URLs.
SCRIPT reserved for future use with scripting languages.
STYLE reserved for future use with style sheets.
META used to supply meta info as name/value pairs.
LINK used to define relationships with other documents.

TITLE, SCRIPT and STYLE are containers and require both start and end tags. The other elements are not containers so that end tags are forbidden. Note that conforming browsers won't render the contents of SCRIPT and STYLE elements.

TITLE

```
<!ELEMENT TITLE - -  (#PCDATA)* -(%head.misc)>
```

Every HTML 3.2 document **must** have exactly one TITLE element in the document's HEAD. It provides an advisory title which can be displayed in a user agent's window caption etc. The content model is PCDATA. As a result, character entities can be used for accented characters and to escape special characters such as & and <. Markup is not permitted in the content of a TITLE element.

Example TITLE element:

```
    <TITLE>A study of population dynamics</TITLE>
```

STYLE and SCRIPT

```
<!ELEMENT STYLE  - - CDATA -- placeholder for style info -->
<!ELEMENT SCRIPT - - CDATA -- placeholder for script statements -->
```

These are place holders for the introduction of style sheets and client-side scripts in future versions of HTML. User agents should hide the contents of these elements.

These elements are defined with CDATA as the content type. As a result they may contain only SGML characters. All markup characters or delimiters are ignored and passed as data to the application, except for ETAGO ("</") delimiters followed immediately by a name character [a-zA-Z]. This means that the element's end-tag (or that of an element in which it is nested) is recognized, while an error occurs if the ETAGO is invalid.

ISINDEX

```
<!ELEMENT ISINDEX - O EMPTY>
<!ATTLIST ISINDEX
    prompt CDATA #IMPLIED -- prompt message -->
```

The ISINDEX element indicates that the user agent should provide a single line text input field for entering a query string. There are no restrictions on the number of characters that can be entered. The PROMPT attribute can be used to specify a prompt string for the input field, e.g.

```
<ISINDEX PROMPT="Search Phrase">
```

The semantics for ISINDEX are currently well defined only when the base URL for the enclosing document is an HTTP URL. Typically, when the user presses the enter (return) key, the query string is sent to the server identified by the base URL for this document. For example, if the query string entered is "ten green apples" and the base URL is:

```
http://www.acme.com/
```

then the query generated is:

```
http://www.acme.com/?ten+green+apples"
```

Note that space characters are mapped to "+" characters and that normal URL character escaping mechanisms apply. For further details see the HTTP specification.

***Note** in practice, the query string is resticted to Latin-1 as there is no current mechanism for the URL to specify a character set for the query.*

BASE

```
<!ELEMENT BASE - O EMPTY>
<!ATTLIST BASE
    href %URL  #REQUIRED
    >
```

The BASE element gives the base URL for dereferencing relative URLs, using the rules given by the URL specification, e.g.

```
<BASE href="http://www.acme.com/intro.html">
 ...
<IMG SRC="icons/logo.gif">
```

The image is deferenced to

```
http://www.acme.com/icons/logo.gif
```

In the absence of a BASE element the document URL should be used. Note that this is not necessarily the same as the URL used to request the document, as the base URL may be overridden by an HTTP header accompanying the document.

META

```
<!ELEMENT META - O EMPTY    -- Generic Metainformation -->
<!ATTLIST META
        http-equiv  NAME    #IMPLIED  -- HTTP response header name
--
        name        NAME    #IMPLIED  -- metainformation name
--
        content     CDATA   #REQUIRED -- associated information
--
        >
```

The META element can be used to include name/value pairs describing properties of the document, such as author, expiry date, a list of key words etc. The NAME attribute specifies the property name while the CONTENT attribute specifies the property value, e.g.

```
    <META NAME="Author" CONTENT="Dave Raggett">
```

The HTTP-EQUIV attribute can be used in place of the NAME attribute and has a special significance when documents are retrieved via the Hypertext Transfer Protocol (HTTP). HTTP servers may use the property name specified by the HTTP-EQUIV attribute to create an RFC 822 style header in the HTTP response. This can't be used to set certain HTTP headers though, see the HTTP specification for details.

```
    <META HTTP-EQUIV="Expires" CONTENT="Tue, 20 Aug 1996 14:25:27
GMT">
```

will result in the HTTP header:

```
    Expires: Tue, 20 Aug 1996 14:25:27 GMT
```

This can be used by caches to determine when to fetch a fresh copy of the associated document.

LINK

LINK provides a media independent method for defining relationships with other documents and resources. LINK has been part of HTML since the very early days, although few browsers as yet take advantage of it (most still ignore LINK elements).

LINK elements can be used *in principle*:

a. for document specific navigation toolbars or menus
b. to control how collections of HTML files are rendered into printed documents
c. for linking associated resources such as style sheets and scripts
d. to provide alternative forms of the current document

```
<!ELEMENT LINK - O EMPTY>
<!ATTLIST LINK
    href    %URL    #IMPLIED    -- URL for linked resource --
    rel     CDATA   #IMPLIED    -- forward link types --
    rev     CDATA   #IMPLIED    -- reverse link types --
    title   CDATA   #IMPLIED    -- advisory title string --
    >
```

href
Specifies a URL designating the linked resource.

rel
The forward relationship also known as the "link type". It specifies a named relationship from the enclosing document to the resource specified by the HREF attribute. HTML link relationships are as yet unstandardized, although some conventions have been established.

rev
This defines a reverse relationship. A link from document A to document B with REV=*relation* expresses the same relationship as a link from B to A with REL=*relation*. REV=made is sometimes used to identify the document author, either the author's email address with a mailto URL, or a link to the author's home page.

title
An advisory title for the linked resource.

Here are some proposed relationship values:

rel=top
The link references the top of a hierarchy, e.g. the first or cover page in a collection.

rel=contents
The link references a document serving as a table of contents.

rel=index
The link references a document providing an index for the current document.

rel=glossary
The link references a document providing a glossary of terms that are relevant to the current document.

rel=copyright
The link references a copyright statement for the current document.

rel=next
The link references the next document to visit in a guided tour. It can be used, for example, to preload the next page.

`rel=previous`
: The link references the previous document in a guided tour.

`rel=help`
: The link references a document offering help, e.g. describing the wider context and offering further links to relevant documents. This is aimed at reorienting users who have lost their way.

`rel=search`
: The link references a page for searching material related to a collection of pages

Example `LINK` elements:

```
<LINK REL=Contents HREF=toc.html>
<LINK REL=Previous HREF=doc31.html>
<LINK REL=Next HREF=doc33.html>

<LINK REL=Chapter REV=Contents HREF=chapter2.html>
```

The BODY element

This contains the document body. Both start and end tags for `BODY` may be omitted. The body can contain a wide range of elements:

- Headings (H1 - H6)
- The ADDRESS element
- Block level Elements
- Text level elements

The key attributes are: `BACKGROUND`, `BGCOLOR`, `TEXT`, `LINK`, `VLINK` and `ALINK`. These can be used to set a repeating background image, plus background and foreground colors for normal text and hypertext links.

```
<!ENTITY % body.content "(%heading | %text | %block | ADDRESS)*">

<!ENTITY % color "CDATA" -- a color specification: #HHHHHH @@
details? -->

<!ENTITY % body-color-attrs "
        bgcolor %color #IMPLIED
        text %color #IMPLIED
        link %color #IMPLIED
        vlink %color #IMPLIED
        alink %color #IMPLIED
        ">
```

```
<!ELEMENT BODY O O  %body.content>
<!ATTLIST BODY
        background %URL #IMPLIED  -- texture tile for document
background --
        %body-color-attrs;  -- bgcolor, text, link, vlink, alink --
        >
```

Example:

```
 <body bgcolor=white text=black link=red vlink=maroon alink=fuchsia>
```

bgcolor
: Specifies the background color for the document body. See below for the syntax of color values.

text
: Specifies the color used to stroke the document's text. This is generally used when you have changed the background color with the `BGCOLOR` or `BACKGROUND` attributes.

link
: Specifies the color used to stroke the text for unvisited hypertext links.

vlink
: Specifies the color used to stroke the text for visited hypertext links.

alink
: Specifies the highlight color used to stroke the text for hypertext links at the moment the user clicks on the link.

background
: Specifies a URL for an image that will be used to tile the document background.

Colors are given in the sRGB color space as hexadecimal numbers (e.g. `COLOR="#C0FFC0"`), or as one of 16 widely understood color names. These colors were originally picked as being the standard 16 colors supported with the Windows VGA palette.

Color names and sRGB values

Black = "#000000"	Green = "#008000"
Silver = "#C0C0C0"	Lime = "#00FF00"
Gray = "#808080"	Olive = "#808000"
White = "#FFFFFF"	Yellow = "#FFFF00"
Maroon = "#800000"	Navy = "#000080"
Red = "#FF0000"	Blue = "#0000FF"


Purple = "#800080"


Teal = "#008080"


Fuchsia = "#FF00FF"


Aqua = "#00FFFF"

Block and Text level elements

Most elements that can appear in the document body fall into one of two groups: block level elements which cause paragraph breaks, and text level elements which don't. Common block level elements include H1 to H6 (headers), P (paragraphs) LI (list items), and HR (horizontal rules). Common text level elements include EM, I, B and FONT (character emphasis), A (hypertext links), IMG and APPLET (embedded objects) and BR (line breaks). Note that block elements generally act as containers for text level and other block level elements (excluding headings and address elements), while text level elements can only contain other text level elements. The exact model depends on the element.

Headings

```
<!--
  There are six levels of headers from H1 (the most important)
  to H6 (the least important).
-->

<!ELEMENT ( %heading )  - -  (%text;)*>
<!ATTLIST ( %heading )
        align  (left|center|right) #IMPLIED
        >
```

H1, H2, H3, H4, H5 and H6 are used for document headings. You always need the start and end tags. H1 elements are more important than H2 elements and so on, so that H6 elements define the least important level of headings. More important headings are generally rendered in a larger font than less important ones. Use the optional ALIGN attribute to set the text alignment within a heading, e.g.

```
  <H1 ALIGN=CENTER> ... centered heading ... </H1>
```

The default is left alignment, but this can be overridden by an enclosing DIV or CENTER element.

ADDRESS

```
    <!ENTITY % address.content "((%text;) | P)*">

    <!ELEMENT ADDRESS - - %address.content>
```

The ADDRESS element requires start and end tags, and specifies information such as authorship and

contact details for the current document. User agents should render the content with paragraph-breaks before and after. Note that the content is restricted to paragraphs, plain text and text-like elements as defined by the %text entity.

Example:

```
<ADDRESS>
Newsletter editor<BR>
J.R. Brown<BR>
8723 Buena Vista, Smallville, CT 01234<BR>
Tel: +1 (123) 456 7890
</ADDRESS>
```

Block elements

P *paragraphs*

The paragraph element requires a start tag, but the end tag can always be omitted. Use the `ALIGN` attribute to set the text alignment within a paragraph, e.g. `<P ALIGN=RIGHT>`

UL *unordered lists*

These require start and end tags, and contain one or more `LI` elements representing individual list items.

OL *ordered (i.e. numbered) lists*

These require start and end tags, and contain one or more `LI` elements representing individual list items.

DL *definition lists*

These require start and end tags and contain `DT` elements that give the terms, and `DD` elements that give corresponding definitions.

PRE *preformatted text*

Requires start and end tags. These elements are rendered with a monospaced font and preserve layout defined by whitespace and line break characters.

DIV *document divisions*

Requires start and end tags. It is used with the `ALIGN` attribute to set the text alignment of the block elements it contains. `ALIGN` can be one of `LEFT`, `CENTER` or `RIGHT`.

CENTER *text alignment*

Requires start and end tags. It is used to center text lines enclosed by the `CENTER` element. See `DIV` for a more general solution.

BLOCKQUOTE *quoted passage*

Requires start and end tags. It is used to enclose extended quotations and is typically rendered with indented margins.

FORM *fill-out forms*

Requires start and end tags. This element is used to define a fill-out form for processing by HTTP servers. The attributes are `ACTION`, `METHOD` and `ENCTYPE`. Form elements can't be nested.

ISINDEX *primitive HTML forms*

Not a container, so the end tag is forbidden. This predates FORM and is used for simple kinds of forms which have a single text input field, implied by this element. A single ISINDEX can appear in the document head or body.

HR *horizontal rules*

Not a container, so the end tag is forbidden. attributes are ALIGN, NOSHADE, SIZE and WIDTH.

TABLE *can be nested*

Requires start and end tags. Each table starts with an optional CAPTION followed by one or more TR elements defining table rows. Each row has one or more cells defined by TH or TD elements. attributes for TABLE elements are WIDTH, BORDER, CELLSPACING and CELLPADDING.

Paragraphs

```
<!ELEMENT P     - O (%text)*>
<!ATTLIST P
    align  (left|center|right) #IMPLIED
    >
```

The P element is used to markup paragraphs. It is a container and requires a start tag. The end tag is optional as it can always be inferred by the parser. User agents should place paragraph breaks before and after P elements. The rendering is user agent dependent, but text is generally wrapped to fit the space available.

Example:

```
<P>This is the first paragraph.
<P>This is the second paragraph.
```

Paragraphs are usually rendered flush left with a ragged right margin. The ALIGN attribute can be used to explicitly specify the horizontal alignment:

`align=left`
: The paragraph is rendered flush left.

`align=center`
: The paragraph is centered.

`align=right`
: The paragraph is rendered flush right.

For example:

```
<p align=center>This is a centered paragraph.
```

```
<p align=right>and this is a flush right paragraph.
```

The default is left alignment, but this can be overridden by an enclosing DIV or CENTER element.

Lists

List items can contain block and text level items, including nested lists, although headings and address elements are excluded. This limitation is defined via the %flow entity.

Unordered Lists

```
    <!ELEMENT UL - -  (LI)+>
    <!ENTITY % ULStyle "disc|square|circle">

    <!ATTLIST UL -- unordered lists --
        type    (%ULStyle)   #IMPLIED   -- bullet style --
        compact (compact)    #IMPLIED   -- reduced interitem
spacing --
        >

    <!ELEMENT LI - O %flow -- list item -->
    <!ATTLIST LI
        type   (%LIStyle)    #IMPLIED   -- list item style --
        >
```

Unordered lists take the form:

```
    <UL>
      <LI> ... first list item
      <LI> ... second list item
      ...
    </UL>
```

The UL element is used for unordered lists. Both start and end tags are always needed. The LI element is used for individual list items. The end tag for LI elements can always be omitted. Note that LI elements can contain nested lists. The COMPACT attribute can be used as a hint to the user agent to render lists in a more compact style.

The TYPE attribute can be used to set the bullet style on UL and LI elements. The permitted values are "disc", "square" or "circle". The default generally depends on the level of nesting for lists.

- with `<li type=disc>`
  - with `<li type=square>`
    - with `<li type=circle>`

This list was chosen to cater for the original bullet shapes used by Mosaic in 1993.

Ordered (i.e. numbered) Lists

```
    <!ELEMENT OL - -  (LI)+>
    <!ATTLIST OL -- ordered lists --
        type      CDATA      #IMPLIED   -- numbering style --
        start     NUMBER     #IMPLIED   -- starting sequence number
--
        compact  (compact)   #IMPLIED   -- reduced interitem
spacing --
        >

    <!ELEMENT LI - O %flow -- list item -->
    <!ATTLIST LI
        type      CDATA      #IMPLIED   -- list item style --
        value     NUMBER     #IMPLIED   -- set sequence number --
        >
```

Ordered (i.e. numbered) lists take the form:

```
    <OL>
      <LI> ... first list item
      <LI> ... second list item
      ...
    </OL>
```

The `OL START` attribute can be used to initialize the sequence number (by default it is initialized to 1). You can set it later on with the `VALUE` attribute on `LI` elements. Both of these attributes expect integer values. You can't indicate that numbering should be continued from a previous list, or to skip missing values without giving an explicit number.

The `COMPACT` attribute can be used as a hint to the user agent to render lists in a more compact style. The `OL TYPE` attribute allows you to set the numbering style for list items:

Type	Numbering style	
1	Arabic numbers	1, 2, 3, ...
a	lower alpha	a, b, c, ...
A	upper alpha	A, B, C, ...
i	lower roman	i, ii, iii, ...
I	upper roman	I, II, III, ...

Definition Lists

```
<!-- definition lists - DT for term, DD for its definition -->

<!ELEMENT DL    - -  (DT|DD)+>
<!ATTLIST DL
        compact (compact) #IMPLIED -- more compact style --
        >

<!ELEMENT DT - O  (%text)*>
<!ELEMENT DD - O  %flow;>
```

Definition lists take the form:

```
  <DL>
    <DT> term name
    <DD> term definition
    ...
  </DL>
```

DT elements can only act as containers for text level elements, while DD elements can hold block level elements as well, excluding headings and address elements.

For example:

```
<DL>
<DT>Term 1<dd>This is the definition of the first term.
<DT>Term 2<dd>This is the definition of the second term.
</DL>
```

which could be rendered as:

Term 1
: This is the definition of the first term.

Term 2
: This is the definition of the second term.

The COMPACT attribute can be used with the DL element as a hint to the user agent to render lists in a more compact style.

DIR and MENU

```
<!ELEMENT (DIR|MENU) - -  (LI)+ -(%block)>
<!ATTLIST (DIR|MENU)
        compact (compact) #IMPLIED
```

```
        >
```

These elements have been part of HTML from the early days. They are intended for unordered lists similar to UL elements. User agents are recommended to render DIR elements as multicolumn directory lists, and MENU elements as single column menu lists. In practice, Mosaic and most other user agents have ignored this advice and instead render DIR and MENU in an identical way to UL elements.

Preformatted Text

```
<!ELEMENT PRE - - (%text)* -(%pre.exclusion)>
<!ATTLIST PRE
        width NUMBER #implied
        >
```

The PRE element can be used to include preformatted text. User agents render this in a fixed pitch font, preserving spacing associated with white space characters such as space and newline characters. Automatic word-wrap should be disabled within PRE elements.

Note that the SGML standard requires that the parser remove a newline immediately following the start tag or immediately preceding the end tag.

PRE has the same content model as paragraphs, excluding images and elements that produce changes in font size, e.g. IMG, BIG, SMALL, SUB, SUP and FONT.

A few user agents support the WIDTH attribute. It provides a hint to the user agent of the required width in characters. The user agent can use this to select an appropriate font size or to indent the content appropriately.

Here is an example of a PRE element; a verse from Shelley (To a Skylark):

```
<PRE>
       Higher still and higher
         From the earth thou springest
       Like a cloud of fire;
         The blue deep thou wingest,
And singing still dost soar, and soaring ever singest.
</PRE>
```

which is rendered as:

```
       Higher still and higher
         From the earth thou springest
       Like a cloud of fire;
```

```
         The blue deep thou wingest,
And singing still dost soar, and soaring ever singest.
```

The horizontal tab character (encoded in Unicode, US ASCII and ISO 8859-1 as decimal 9) should be interpreted as the smallest non-zero number of spaces which will leave the number of characters so far on the line as a multiple of 8. Its use is strongly discouraged since it is common practice when editing to set the tab-spacing to other values, leading to misaligned documents.

XMP, LISTING and PLAINTEXT

```
<![ %HTML.Deprecated [

<!ENTITY % literal "CDATA"
        -- historical, non-conforming parsing mode where
           the only markup signal is the end tag
           in full
        -->

<!ELEMENT (XMP|LISTING) - -  %literal>
<!ELEMENT PLAINTEXT - O %literal>

]]>
```

These are obsolete tags for preformatted text that predate the introduction of PRE. User agents may support these for backwards compatibility. Authors should avoid using them in new documents!

DIV and CENTER

```
<!ELEMENT DIV - - %body.content>
<!ATTLIST DIV
        align   (left|center|right) #IMPLIED -- alignment of
following text --
        >

<!-- CENTER is a shorthand for DIV with ALIGN=CENTER -->
<!ELEMENT center - - %body.content>
```

DIV elements can be used to structure HTML documents as a hierarchy of divisions. The ALIGN attribute can be used to set the default horizontal alignment for elements within the content of the DIV element. Its value is restricted to LEFT, CENTER or RIGHT, and is defined in the same way as for the paragraph element <P>.

Note that because DIV is a block-like element it will terminate an open P element. Other than this, user agents are **not** expected to render paragraph breaks before and after DIV elements. CENTER is directly equivalent to DIV with ALIGN=CENTER. Both DIV and CENTER require start and end tags.

`CENTER` was introduced by Netscape before they added support for the HTML 3.0 `DIV` element. It is retained in HTML 3.2 on account of its widespread deployment.

BLOCKQUOTE

```
<!ELEMENT BLOCKQUOTE - - %body.content>
```

This is used to enclose block quotations from other works. Both the start and end tags are required. It is often rendered indented, e.g.

> They went in single file, running like hounds on a strong scent, and an eager light was in their eyes. Nearly due west the broad swath of the marching Orcs tramped its ugly slot; the sweet grass of Rohan had been bruised and blackened as they passed.
>
> *from "The Two Towers" by J.R.R. Tolkien.*

FORM

```
<!ENTITY % HTTP-Method "GET | POST"
        -- as per HTTP specification
        -->

<!ELEMENT FORM - - %body.content -(FORM)>
<!ATTLIST FORM
        action %URL #IMPLIED  -- server-side form handler --
        method (%HTTP-Method) GET -- see HTTP specification --
        enctype %Content-Type; "application/x-www-form-urlencoded"
        >
```

This is used to define an HTML form, and you can have more than one form in the same document. Both the start and end tags are required. For very simple forms, you can also use the `ISINDEX` element. Forms can contain a wide range of HTML markup including several kinds of form fields such as single and multi-line text fields, radio button groups, checkboxes, and menus.

action
: This specifies a URL which is either used to post forms via email, e.g. `action="mailto:foo@bar.com"`, or used to invoke a server-side forms handler via HTTP, e.g. `action="http://www.acme.com/cgi-bin/register.pl"`

method
: When the action attribute specifies an HTTP server, the method attribute determines which HTTP method will be used to send the form's contents to the server. It can be either `GET` or `POST`, and defaults to `GET`.

enctype

This determines the mechanism used to encode the form's contents. It defaults to *application/x-www-form-urlencoded*.

Further details on handling forms are given in RFC 1867.

HR - *horizontal rules*

Horizontal rules may be used to indicate a change in topic. In a speech based user agent, the rule could be rendered as a pause.

```
<!ELEMENT HR    - O EMPTY>
<!ATTLIST HR
        align (left|right|center) #IMPLIED
        noshade (noshade) #IMPLIED
        size  %Pixels #IMPLIED
        width %Length #IMPLIED
        >
```

`HR` elements are not containers so the end tag is forbidden. The attributes are: `ALIGN`, `NOSHADE`, `SIZE` and `WIDTH`.

align
This determines whether the rule is placed at the left, center or right of the space between the current left and right margins for `align=left`, `align=center` or `align=right` respectively. By default, the rule is centered.

noshade
This attribute requests the user agent to render the rule in a solid color rather than as the traditional two colour "groove".

size
This can be used to set the height of the rule in pixels.

width
This can be used to set the width of the rule in pixels (e.g. `width=100`) or as the percentage between the current left and right margins (e.g. `width="50%"`). The default is 100%.

Tables

HTML 3.2 includes a widely deployed subset of the specification given in RFC 1942 and can be used to markup tabular material or for layout purposes. Note that the latter role typically causes problems when rending to speech or to text only user agents.

```
<!-- horizontal placement of table relative to window -->
<!ENTITY % Where "(left|center|right)">

<!-- horizontal alignment attributes for cell contents -->
```

```
<!ENTITY % cell.halign
        "align  (left|center|right) #IMPLIED"
        >

<!-- vertical alignment attributes for cell contents -->
<!ENTITY % cell.valign
        "valign  (top|middle|bottom)  #IMPLIED"
        >

<!ELEMENT table - - (caption?, tr+)>
<!ELEMENT tr - O (th|td)*>
<!ELEMENT (th|td) - O %body.content>

<!ATTLIST table                         -- table element --
        align     %Where;   #IMPLIED  -- table position relative to
window --
        width     %Length   #IMPLIED  -- table width relative to
window --
        border    %Pixels   #IMPLIED  -- controls frame width
around table --
        cellspacing %Pixels #IMPLIED  -- spacing between cells --
        cellpadding %Pixels #IMPLIED  -- spacing within cells --
        >

<!ELEMENT CAPTION - - (%text;)* -- table or figure caption -->
<!ATTLIST CAPTION
        align (top|bottom) #IMPLIED
        >

<!ATTLIST tr                          -- table row --
        %cell.halign;               -- horizontal alignment in cells
--
        %cell.valign;               -- vertical alignment in cells --
        >

<!ATTLIST (th|td)                     -- header or data cell --
        nowrap (nowrap)  #IMPLIED  -- suppress word wrap --
        rowspan NUMBER   1         -- number of rows spanned by
cell --
        colspan NUMBER   1         -- number of cols spanned by
cell --
        %cell.halign;               -- horizontal alignment in cells
--
        %cell.valign;               -- vertical alignment in cells --
        width   %Pixels  #IMPLIED  -- suggested width for cell --
        height  %Pixels  #IMPLIED  -- suggested height for cell --
```

```
        >
```

Tables take the general form:

```
<TABLE BORDER=3 CELLSPACING=2 CELLPADDING=2 WIDTH="80%">
<CAPTION> ... table caption ... </CAPTION>
<TR><TD> first cell <TD> second cell
<TR> ...
...
</TABLE>
```

The attributes on TABLE are all optional. By default, the table is rendered without a surrounding border. The table is generally sized automatically to fit the contents, but you can also set the table width using the WIDTH attribute. BORDER, CELLSPACING and CELLPADDING provide further control over the table's appearence. Captions are rendered at the top or bottom of the table depending on the ALIGN attribute.

Each table row is contained in a TR element, although the end tag can always be omitted. Table cells are defined by TD elements for data and TH elements for headers. Like TR, these are containers and can be given without trailing end tags. TH and TD support several attributes: ALIGN and VALIGN for aligning cell content, ROWSPAN and COLSPAN for cells which span more than one row or column. A cell can contain a wide variety of other block and text level elements including form fields and other tables.

The TABLE element always requires both start and end tags. It supports the following attributes:

align
: This takes one of the case insensitive values: LEFT, CENTER or RIGHT. It specifies the horizontal placement of the table relative to the current left and right margins. It defaults to left alignment, but this can be overridden by an enclosing DIV or CENTER element.

width
: In the absence of this attribute the table width is automatically determined from the table contents. You can use the WIDTH attribute to set the table width to a fixed value in pixels (e.g. WIDTH=212) or as a percentage of the space between the current left and right margins (e.g. WIDTH="80%").

border
: This attribute can be used to specify the width of the outer border around the table to a given number of pixels (e.g. BORDER=4). The value can be set to zero to suppress the border altogether. In the absence of this attribute the border should be suppressed. Note that some browsers also accept <TABLE BORDER> with the same semantics as BORDER=1.

cellspacing
: In traditional desktop publishing software, adjacent table cells share a common border. This is not the case in HTML. Each cell is given its own border which is separated from the borders around neighboring cells. This separation can be set in pixels using the CELLSPACING attribute, (e.g. CELLSPACING=10). The same value also determines the separation between

the table border and the borders of the outermost cells.

cellpadding
This sets the padding in pixels between the border around each cell and the cell's contents.

The CAPTION element has one attribute ALIGN which can be either ALIGN=TOP or ALIGN=BOTTOM. This can be used to force the caption to be placed above the top or below the bottom of the table respectively. Most user agents default to placing the caption above the table. CAPTION always requires both start and end tags. Captions are limited to plain text and text-level elements as defined by the %text entity. Block level elements are not permitted.

The TR or table row element requires a start tag, but the end tag can always be left out. TR acts as a container for table cells. It has two attributes:

align
Sets the default horizontal alignment of cell contents. It takes one of the case insensitive values: LEFT, CENTER or RIGHT and plays the same role as the ALIGN attribute on paragraph elements.

valign
This can be used to set the default vertical alignment of cell contents within each cell. It takes one of the case insensitive values: TOP, MIDDLE or BOTTOM to position the cell contents at the top, middle or bottom of the cell respectively.

There are two elements for defining table cells. TH is used for header cells and TD for data cells. This distinction allows user agents to render header and data cells in different fonts, and enables speech based browsers to do a better job. The start tags for TH and TD are always needed but the end tags can be left out. Table cells can have the following attributes:

nowrap
The presence of this attribute disables automatic word wrap within the contents of this cell (e. g. <TD NOWRAP>). This is equivalent to using the   entity for non-breaking spaces within the content of the cell.

rowspan
This takes a positive integer value specifying the number of rows spanned by this cell. It defaults to one.

colspan
This takes a positive integer value specifying the number of columns spanned by this cell. It defaults to one.

align
Specifies the default horizontal alignment of cell contents, and overrides the ALIGN attribute on the table row. It takes the same values: LEFT, CENTER and RIGHT. If you don't specify an ALIGN attribute value on the cell, the default is left alignment for <td> and center alignment for <th> although you can override this with an ALIGN attribute on the TR element.

valign
Specifies the default vertical alignment of cell contents, overriding the VALIGN attribute on the table row. It takes the same values: TOP, MIDDLE and BOTTOM. If you don't specify a

VALIGN attribute value on the cell, the default is middle although you can override this with a `VALIGN` attribute on the `TR` element.

width

Specifies the suggested width for a cell content in pixels excluding the cell padding. This value will normally be used except when it conflicts with the width requirements for other cells in the same column.

height

Specifies the suggested height for a cell content in pixels excluding the cell padding. This value will normally be used except when it conflicts with the height requirements for other cells in the same row.

Tables are commonly rendered in bas-relief, raised up with the outer border as a bevel, and individual cells inset into this raised surface. Borders around individual cells are only drawn if the cell has explicit content. White space doesn't count for this purpose with the exception of .

The algorithms used to automatically size tables should take into account the minimum and maximum width requirements for each cell. This is used to determine the minimum and maximum width requirements for each column and hence for the table itself.

Cells spanning more than one column contribute to the widths of each of the columns spanned. One approach is to evenly apportion the cell's minimum and maximum width between these columns, another is to weight the apportioning according to the contributions from cells that don't span multiple columns.

For some user agents it may be necessary or desirable to break text lines within words. In such cases a visual indication that this has occurred is advised.

The minimum and maximum width of nested tables contribute to the minimum and maximum width of the cell in which they occur. Once the width requirements are known for the top level table, the column widths for that table can be assigned. This allows the widths of nested tables to be assigned and hence in turn the column widths of such tables. If practical, all columns should be assigned at least their minimum widths. It is suggested that any surplus space is then shared out proportional to the difference between the minimum and maximum width requirements of each column.

Note that pixel values for width and height refer to screen pixels, and should be multiplied by an appropriate factor when rendering to very high resolution devices such as laser printers. For instance if a user agent has a display with 75 pixels per inch and is rendering to a laser printer with 600 dots per inch, then the pixel values given in HTML attributes should be multiplied by a factor of 8.

Text level elements

These don't cause paragraph breaks. Text level elements that define character styles can generally be nested. They can contain other text level elements but not block level elements.

Font style elements

These all require start and end tags, e.g.

```
This has some <B>bold text</B>.
```

Text level elements must be properly nested - the following is in error:

```
This has some <B>bold and <I></B>italic text</I>.
```

User agents should do their best to respect nested emphasis, e.g.

```
This has some <B>bold and <I>italic text</I></B>.
```

Where the available fonts are restricted or for speech output, alternative means should be used for rendering differences in emphasis.

TT teletype or monospaced text
I italic text style
B bold text style
U underlined text style
STRIKE strike-through text style
BIG places text in a large font
SMALL places text in a small font
SUB places text in subscript style
SUP places text in superscript style

Note: future revisions to HTML may be phase out STRIKE in favor of the more concise "S" tag from HTML 3.0.

Phrase Elements

These all require start and end tags, e.g.

```
  This has some <EM>emphasized text</EM>.
```

EM basic emphasis typically rendered in an italic font
STRONG strong emphasis typically rendered in a bold font
DFN defining instance of the enclosed term
CODE used for extracts from program code
SAMP used for sample output from programs, and scripts etc.
KBD used for text to be typed by the user
VAR used for variables or arguments to commands
CITE used for citations or references to other sources

Form fields

INPUT, SELECT and TEXTAREA are only allowed within FORM elements. INPUT can be used for a variety of form fields including single line text fields, password fields, checkboxes, radio buttons, submit and reset buttons, hidden fields, file upload, and image buttons. SELECT elements are used for single or multiple choice menus. TEXTAREA elements are used to define multi-line text fields. The content of the element is used to initialize the field.

INPUT *text fields, radio buttons, check boxes, ...*

INPUT elements are not containers and so the end tag is forbidden.

```
<!ENTITY % IAlign "(top|middle|bottom|left|right)">

<!ENTITY % InputType
        "(TEXT | PASSWORD | CHECKBOX | RADIO | SUBMIT
            | RESET | FILE | HIDDEN | IMAGE)">

<!ELEMENT INPUT - O EMPTY>
<!ATTLIST INPUT
        type %InputType TEXT     -- what kind of widget is needed --
        name  CDATA #IMPLIED     -- required for all but submit and
reset --
        value CDATA #IMPLIED     -- required for radio and
checkboxes --
        checked (checked) #IMPLIED -- for radio buttons and check
boxes --
        size CDATA  #IMPLIED     -- specific to each type of field
--
        maxlength NUMBER #IMPLIED
        src   %URL  #IMPLIED     -- for fields with background
```

```
images --
        align %IAlign #IMPLIED   -- vertical or horizontal
alignment --
        >
```

type

Used to set the type of input field:

`type=text` *(the default)*

A single line text field whose visible size can be set using the `size` attribute, e.g. `size=40` for a 40 character wide field. Users should be able to type more than this limit though with the text scrolling through the field to keep the input cursor in view. You can enforce an upper limit on the number of characters that can be entered with the `maxlength` attribute. The `name` attribute is used to name the field, while the `value` attribute can be used to initialize the text string shown in the field when the document is first loaded.

```
<input type=text size=40 name=user value="your name">
```

`type=password`

This is like type=text, but echoes characters using a character like * to hide the text from prying eyes when entering passwords. You can use `size` and `maxlength` attributes to control the visible and maximum length exactly as per regular text fields.

```
<input type=password size=12 name=pw>
```

`type=checkbox`

Used for simple Boolean attributes, or for attributes that can take multiple values at the same time. The latter is represented by several checkbox fields with the same `name` and a different `value` attribute. Each checked checkbox generates a separate name/value pair in the submitted data, even if this results in duplicate names. Use the `checked` attribute to initialize the checkbox to its checked state.

```
<input type=checkbox checked name=uscitizen value=yes>
```

`type=radio`

Used for attributes which can take a single value from a set of alternatives. Each radio button field in the group should be given the same `name`. Radio buttons require an explicit `value` attribute. Only the checked radio button in the group generates a name/value pair in the submitted data. One radio button in each group should be initially checked using the `checked` attribute.

```
<input type=radio name=age value="0-12">
<input type=radio name=age value="13-17">
```

```
<input type=radio name=age value="18-25">
<input type=radio name=age value="26-35" checked>
<input type=radio name=age value="36-">
```

`type=submit`

This defines a button that users can click to submit the form's contents to the server. The button's label is set from the `value` attribute. If the `name` attribute is given then the submit button's name/value pair will be included in the submitted data. You can include several submit buttons in the form. See `type=image` for graphical submit buttons.

```
<input type=submit value="Party on ...">
```

`type=image`

This is used for graphical submit buttons rendered by an image rather than a text string. The URL for the image is specified with the `src` attribute. The image alignment can be specified with the `align` attribute. In this respect, graphical submit buttons are treated identically to `IMG` elements, so you can set align to left, right, top, middle or bottom. The x and y values of the location clicked are passed to the server: In the submitted data, image fields are included as two name/value pairs. The names are derived by taking the name of the field and appending ".x" for the x value, and ".y" for the y value.

```
<p>Now choose a point on the map:

<input type=image name=point src="map.gif">
```

***Note**: image fields typically cause problems for text-only and speech-based user agents!*

`type=reset`

This defines a button that users can click to reset form fields to their initial state when the document was first loaded. You can set the label by providing a `value` attribute. Reset buttons are never sent as part of the form's contents.

```
<input type=reset value="Start over ...">
```

`type=file`

This provides a means for users to attach a file to the form's contents. It is generally rendered by text field and an associated button which when clicked invokes a file browser to select a file name. The file name can also be entered directly in the text field. Just like type=text you can use the `size` attribute to set the visible width of this field in average character widths. You can set an upper limit to the length of file names using the `maxlength` attribute. Some user agents support the ability to restrict the kinds of files to those matching a comma separated list of MIME content types given

with the ACCEPT attribute e.g. `accept="image/*"` restricts files to images. Further information can be found in RFC 1867.

```
<input type=file name=photo size=20 accept="image/*">
```

`type=hidden`

These fields should not be rendered and provide a means for servers to store state information with a form. This will be passed back to the server when the form is submitted, using the name/value pair defined by the corresponding attributes. This is a work around for the statelessness of HTTP. Another approach is to use HTTP "Cookies".

```
<input type=hidden name=customerid value="c2415-345-8563">
```

name

Used to define the property name that will be used to identify this field's content when it is submitted to the server.

value

Used to initialize the field, or to provide a textual label for submit and reset buttons.

checked

The presence of this attribute is used to initialize checkboxes and radio buttons to their checked state.

size

Used to set the visible size of text fields to a given number of average character widths, e.g. `size=20`

maxlength

Sets the maximum number of characters permitted in a text field.

src

Specifies a URL for the image to use with a graphical submit button.

align

Used to specify image alignment for graphical submit buttons. It is defined just like the IMG align attribute and takes one of the values: `top`, `middle`, `bottom`, `left` or `right`, defaulting to `bottom`.

SELECT *menus*

```
<!ELEMENT SELECT - - (OPTION+)>
<!ATTLIST SELECT
        name CDATA #REQUIRED
        size NUMBER #IMPLIED
        multiple (multiple) #IMPLIED
        >

<!ELEMENT OPTION - O (#PCDATA)*>
```

```
<!ATTLIST OPTION
        selected (selected) #IMPLIED
        value  CDATA  #IMPLIED -- defaults to element content --
        >
```

SELECT is used to define select one from many or many from many menus. SELECT elements require start and end tags and contain one or more OPTION elements that define menu items. One from many menus are generally rendered as drop-down menus while many from many menus are generally shown as list boxes.

Example:

```
<SELECT NAME="flavor">
<OPTION VALUE=a>Vanilla
<OPTION VALUE=b>Strawberry
<OPTION VALUE=c>Rum and Raisin
<OPTION VALUE=d>Peach and Orange
</SELECT>
```

SELECT attributes:

name
: This specifies a property name that is used to identify the menu choice when the form is submitted to the server. Each selected option results in a property name/value pair being included as part of the form's contents.

size
: This sets the number of visible choices for many from many menus.

multiple
: The presence of this attribute signifies that the users can make multiple selections. By default only one selection is allowed.

OPTION attributes:

selected
: When this attribute is present, the option is selected when the document is initially loaded. It is an error for more than one option to be so selected for one from many menus.

value
: Specifies the property value to be used when submitting the form's content. This is combined with the property name as given by the name attribute of the parent SELECT element.

TEXTAREA *multi-line text fields*

```
<!-- Multi-line text input field. -->

<!ELEMENT TEXTAREA - - (#PCDATA)*>
```

```
<!ATTLIST TEXTAREA
        name CDATA #REQUIRED
        rows NUMBER #REQUIRED
        cols NUMBER #REQUIRED
        >
```

TEXTAREA elements require start and end tags. The content of the element is restricted to text and character entities. It is used to initialize the text that is shown when the document is first loaded.

Example:

```
<TEXTAREA NAME=address ROWS=4 COLS=40>
Your address here ...
</TEXTAREA>
```

It is recommended that user agents canonicalize line endings to CR, LF (ASCII decimal 13, 10) when submitting the field's contents. The character set for submitted data should be ISO Latin-1, unless the server has previously indicated that it can support alternative character sets.

name
: This specifies a property name that is used to identify the textarea field when the form is submitted to the server.

rows
: Specifies the number of visible text lines. Users should be able to enter more lines that this, so user agents should provide some means to scroll through the contents of the textarea field when the contents extend beyond the visible area.

cols
: Specifies the visible width in average character widths. Users should be able to enter longer lines that this, so user agents should provide some means to scroll through the contents of the textarea field when the contents extend beyond the visible area. User agents may wrap visible text lines to keep long lines visible without the need for scrolling.

Special Text level Elements

A (Anchor), IMG, APPLET, FONT, BASEFONT, BR and MAP.

The A (anchor) element

```
<!ELEMENT A - - (%text)* -(A)>
<!ATTLIST A
        name    CDATA   #IMPLIED    -- named link end --
        href    %URL    #IMPLIED    -- URL for linked resource --
        rel     CDATA   #IMPLIED    -- forward link types --
        rev     CDATA   #IMPLIED    -- reverse link types --
        title   CDATA   #IMPLIED    -- advisory title string --
```

```
        >
```

Anchors can't be nested and always require start and end tags. They are used to define hypertext links and also to define named locations for use as targets for hypertext links, e.g.

```
   The way to <a href="hands-on.html">happiness</a>.
```

and also to define named locations for use as targets for hypertext links, e.g.

```
   <h2><a name=mit>545 Tech Square - Hacker's Paradise</a></h2>
```

name
: This should be a string defining unique name for the scope of the current HTML document. `NAME` is used to associate a name with this part of a document for use with URLs that target a named section of a document.

href
: Specifies a URL acting as a network address for the linked resource. This could be another HTML document, a PDF file or an image etc.

rel
: The forward relationship also known as the "link type". It can be used to determine to how to deal with the linked resource when printing out a collection of linked resources.

rev
: This defines a reverse relationship. A link from document A to document B with `REV=relation` expresses the same relationship as a link from B to A with `REL=relation`. `REV=made` is sometimes used to identify the document author, either the author's email address with a mailto URL, or a link to the author's home page.

title
: An advisory title for the linked resource.

IMG - *inline images*

```
<!ENTITY % IAlign "(top|middle|bottom|left|right)">

<!ELEMENT IMG    - O EMPTY --  Embedded image -->
<!ATTLIST IMG
        src     %URL     #REQUIRED  -- URL of image to embed --
        alt     CDATA    #IMPLIED   -- for display in place of
image --
        align   %IAlign  #IMPLIED   -- vertical or horizontal
alignment --
        height  %Pixels  #IMPLIED   -- suggested height in pixels --
        width   %Pixels  #IMPLIED   -- suggested width in pixels --
        border  %Pixels  #IMPLIED   -- suggested link border width
--
        hspace  %Pixels  #IMPLIED   -- suggested horizontal gutter
```

```
--
        vspace  %Pixels  #IMPLIED   -- suggested vertical gutter --
        usemap  %URL     #IMPLIED   -- use client-side image map --
        ismap   (ismap)  #IMPLIED   -- use server image map --
        >
```

Used to insert images. `IMG` is an empty element and so the end tag is forbidden. Images can be positioned vertically relative to the current textline or floated to the left or right. See `BR` with the `CLEAR` attribute for control over textflow.

e.g. `<IMG SRC="canyon.gif" ALT="Grand Canyon">`

`IMG` elements support the following attributes:

src
: This attribute is required for every `IMG` element. It specifies a URL for the image resource, for instance a GIF, JPEG or PNG image file.

alt
: This is used to provide a text description of the image and is vital for interoperability with speech-based and text only user agents.

align
: This specifies how the image is positioned relative to the current textline in which it occurs:

`align=top`
: positions the top of the image with the top of the current text line. User agents vary in how they interpret this. Some only take into account what has occurred on the text line prior to the IMG element and ignore what happens after it.

`align=middle`
: aligns the middle of the image with the baseline for the current textline.

`align=bottom`
: is the default and aligns the bottom of the image with the baseline.

`align=left`
: floats the image to the current left margin, temporarily changing this margin, so that subsequent text is flowed along the image's righthand side. The rendering depends on whether there is any left aligned text or images that appear earlier than the current image in the markup. Such text (but not images) generally forces left aligned images to wrap to a new line, with the subsequent text continuing on the former line.

`align=right`
: floats the image to the current right margin, temporarily changing this margin, so that subsequent text is flowed along the image's lefthand side. The rendering depends on whether there is any right aligned text or images that appear earlier than the current image in the markup. Such text (but not images) generally forces right aligned images to wrap to a new line, with the subsequent text continuing on the former line.

Note that some browsers introduce spurious spacing with multiple left or right aligned images.

As a result authors can't depend on this being the same for browsers from different vendors. See BR for ways to control text flow.

width

Specifies the intended width of the image in pixels. When given together with the height, this allows user agents to reserve screen space for the image before the image data has arrived over the network.

height

Specifies the intended height of the image in pixels. When given together with the width, this allows user agents to reserve screen space for the image before the image data has arrived over the network.

border

When the IMG element appears as part of a hypertext link, the user agent will generally indicate this by drawing a colored border (typically blue) around the image. This attribute can be used to set the width of this border in pixels. Use `border=0` to suppress the border altogether. User agents are recommended to provide additional cues that the image is clickable, e.g. by changing the mouse pointer.

hspace

This can be used to provide white space to the immediate left and right of the image. The HSPACE attribute sets the width of this white space in pixels. By default HSPACE is a small non-zero number.

vspace

This can be used to provide white space above and below the image The VSPACE attribute sets the height of this white space in pixels. By default VSPACE is a small non-zero number.

usemap

This can be used to give a URL fragment identifier for a client-side image map defined with the MAP element.

ismap

When the IMG element is part of a hypertext link, and the user clicks on the image, the ISMAP attribute causes the location to be passed to the server. This mechanism causes problems for text-only and speech-based user agents. Whenever its possible to do so use the MAP element instead.

Here is an example of how you use ISMAP:

```
<a href="/cgibin/navbar.map"><img src=navbar.gif ismap border=0></a>
```

The location clicked is passed to the server as follows. The user agent derives a new URL from the URL specified by the HREF attribute by appending `?' the x coordinate `,' and the y coordinate of the location in pixels. The link is then followed using the new URL. For instance, if the user clicked at at the location x=10, y=27 then the derived URL will be: *"/cgibin/navbar.map?10,27"*. It is generally a good idea to suppress the border and use graphical idioms to indicate that the image is clickable.

Note that pixel values refer to screen pixels, and should be multiplied by an appropriate factor when rendering to very high resolution devices such as laser printers. For instance if a user agent has a

display with 75 pixels per inch and is rendering to a laser printer with 600 dots per inch, then the pixel values given in HTML attributes should be multiplied by a factor of 8.

APPLET *(Java Applets)*

```
<!ELEMENT APPLET - - (PARAM | %text)*>
<!ATTLIST APPLET
        codebase %URL     #IMPLIED   -- code base --
        code     CDATA    #REQUIRED  -- class file --
        alt      CDATA    #IMPLIED   -- for display in place of
applet --
        name     CDATA    #IMPLIED   -- applet name --
        width    %Pixels  #REQUIRED  -- suggested width in pixels --
        height   %Pixels  #REQUIRED  -- suggested height in pixels
--
        align    %IAlign  #IMPLIED   -- vertical or horizontal
alignment --
        hspace   %Pixels  #IMPLIED   -- suggested horizontal gutter
--
        vspace   %Pixels  #IMPLIED   -- suggested vertical gutter --
        >

<!ELEMENT PARAM - O EMPTY>
<!ATTLIST PARAM
    name    NMTOKEN   #REQUIRED  -- The name of the parameter --
    value   CDATA     #IMPLIED   -- The value of the parameter --
    >
```

Requires start and end tags. This element is supported by all Java enabled browsers. It allows you to embed a Java applet into HTML documents. APPLET uses associated PARAM elements to pass parameters to the applet. Following the PARAM elements, the content of APPLET elements should be used to provide an alternative to the applet for user agents that don't support Java. It is restricted to text-level markup as defined by the *%text* entity in the DTD. Java-compatible browsers ignore this extra HTML code. You can use it to show a snapshot of the applet running, with text explaining what the applet does. Other possibilities for this area are a link to a page that is more useful for the Java-ignorant browser, or text that taunts the user for not having a Java-compatible browser.

Here is a simple example of a Java applet:

```
<applet code="Bubbles.class" width=500 height=500>
Java applet that draws animated bubbles.
</applet>
```

Here is another one using a PARAM element:

```
<applet code="AudioItem" width=15 height=15>
<param name=snd value="Hello.au|Welcome.au">
Java applet that plays a welcoming sound.
</applet>
```

codebase = *codebaseURL*

This optional attribute specifies the base URL of the applet -- the directory or folder that contains the applet's code. If this attribute is not specified, then the document's URL is used.

code = *appletFile*

This required attribute gives the name of the file that contains the applet's compiled Applet subclass. This file is relative to the base URL of the applet. It cannot be absolute.

alt = *alternateText*

This optional attribute specifies any text that should be displayed if the browser understands the APPLET tag but can't run Java applets.

name = *appletInstanceName*

This optional attribute specifies a name for the applet instance, which makes it possible for applets on the same page to find (and communicate with) each other.

width = *pixels*
height = *pixels*

These required attributes give the initial width and height (in pixels) of the applet display area, not counting any windows or dialogs that the applet brings up.

align = *alignment*

This attribute specifies the alignment of the applet. This attribute is defined in exactly the same way as the IMG element. The permitted values are: `top`, `middle`, `bottom`, `left` and `right`. The default is `bottom`.

vspace = *pixels*
hspace = *pixels*

These optional attributes specify the number of pixels above and below the applet (VSPACE) and on each side of the applet (HSPACE). They're treated the same way as the `IMG` element's VSPACE and HSPACE attributes.

The PARAM element is used to pass named parameters to applet:

<PARAM NAME = *appletParameter* **VALUE =** *value***>**

PARAM elements are the only way to specify applet-specific parameters. Applets read user-specified values for parameters with the `getParameter()` method.

name = *applet parameter name*

value = *parameter value*

SGML character entities such as *é* and *¹* are expanded before the parameter value is passed to the applet. To include an & character use &.

Note: PARAM elements should be placed at the start of the content for the APPLET element. This is not specified as part of the DTD due to technicalities with SGML mixed content models.

FONT

```
<!ELEMENT FONT - - (%text)*     -- local change to font -->
<!ATTLIST FONT
    size    CDATA   #IMPLIED    -- [+]nn e.g. size="+1", size=4 --
    color   CDATA   #IMPLIED    -- #RRGGBB in hex, e.g. red:
color="#FF0000" --
    >
```

Requires start and end tags. This allows you to change the font size and/or color for the enclosed text. The attributes are: SIZE and COLOR. Font sizes are given in terms of a scalar range defined by the user agent with no direct mapping to point sizes etc. The FONT element may be phased out in future revisions to HTML.

size

This sets the font size for the contents of the font element. You can set size to an integer ranging from 1 to 7 for an absolute font size, or specify a relative font size with a signed integer value, e.g. size="+1" or size="-2". This is mapped to an absolute font size by adding the current base font size as set by the BASEFONT element (see below).

color

Used to set the color to stroke the text. Colors are given as RGB in hexadecimal notation or as one of 16 widely understood color names defined as per the BGCOLOR attribute on the BODY element.

Some user agents also support a FACE attribute which accepts a comma separated list of font names in order of preference. This is used to search for an installed font with the corresponding name. FACE is not part of HTML 3.2.

The following shows the effects of setting font to absolute sizes:

size=1 size=2 size=3 size=4 size=5 size=6 size=7

The following shows the effect of relative font sizes using a base font size of 3:

size=-4 size=-3 size=-2 size=-1 size=+1 size=+2 size=+3 size=+4

The same thing with a base font size of 6:

size=-4 size=-3 size=-2 size=-1 size=+1 size=+2 size=+3 size=+4

BASEFONT

```
<!ELEMENT BASEFONT - O EMPTY    -- base font size (1 to 7) -->
<!ATTLIST BASEFONT
    size    CDATA   #IMPLIED    -- e.g. size=4, defaults to 3 --
    >
```

Used to set the base font size. BASEFONT is an empty element so the end tag is forbidden. The SIZE attribute is an integer value ranging from 1 to 7. The base font size applies to the normal and preformatted text but not to headings, except where these are modified using the FONT element with a relative font size.

BR

Used to force a line break. This is an empty element so the end tag is forbidden. The CLEAR attribute can be used to move down past floating images on either margin. <BR CLEAR=LEFT> moves down past floating images on the left margin, <BR CLEAR=RIGHT> does the same for floating images on the right margin, while <BR CLEAR=ALL> does the same for such images on both left and right margins.

MAP

The MAP element provides a mechanism for client-side image maps. These can be placed in the same document or grouped in a separate document although this isn't yet widely supported. The MAP element requires start and end tags. It contains one or more AREA elements that specify hotzones on the associated image and bind these hotzones to URLs.

```
<!ENTITY % SHAPE "(rect|circle|poly)">
<!ENTITY % COORDS "CDATA" -- comma separated list of numbers -->

<!ELEMENT MAP - - (AREA)+>
<!ATTLIST MAP
    name    CDATA   #REQUIRED
    >

<!ELEMENT AREA - O EMPTY>
```

```
<!ATTLIST AREA
    shape   %SHAPE  rect
    coords  %COORDS #IMPLIED  -- defines coordinates for shape --
    href    %URL    #IMPLIED  -- this region acts as hypertext link
--
    nohref (nohref) #IMPLIED  -- this region has no action --
    alt     CDATA   #REQUIRED -- needed for non-graphical user
agents --
    >
```

Here is a simple example for a graphical navigational toolbar:

```
<img src="navbar.gif" border=0 usemap="#map1">

<map name="map1">
 <area href=guide.html alt="Access Guide" shape=rect
coords="0,0,118,28">
 <area href=search.html alt="Search" shape=rect
coords="184,0,276,28">
 <area href=shortcut.html alt="Go" shape=rect coords="118,0,184,28">
 <area href=top10.html alt="Top Ten" shape=rect
coords="276,0,373,28">
</map>
```

The MAP element has one attribute NAME which is used to associate a name with a map. This is then used by the USEMAP attribute on the IMG element to reference the map via a URL fragment identifier. Note that the value of the NAME attribute is case sensitive.

The AREA element is an empty element and so the end tag is forbidden. It takes the following attributes: SHAPE, COORDS, HREF, NOHREF and ALT. The SHAPE and COORDS attributes define a region on the image. If the SHAPE attribute is omitted, SHAPE="RECT" is assumed.

shape=rect coords="*left-x*, *top-y*, *right-x*, *bottom-y*"

shape=circle coords="*center-x*, *center-y*, *radius*"

shape=poly coords="x_1,y_1, x_2,y_2, x_3,y_3, ..."

Where **x** and **y** are measured in pixels from the left/top of the associated image. If **x** and **y** values are given with a percent sign as a suffix, the values should be interpreted as percentages of the image's width and height, respectively. For example:

```
   SHAPE=RECT COORDS="0, 0, 50%, 100%"
```

The HREF attribute gives a URL for the target of the hypertext link. The NOHREF attribute is used when you want to define a region that doesn't act as a hotzone. This is useful when you want to cut a hole in an underlying region acting as a hotzone.

If two or more regions overlap, the region defined first in the map definition takes precedence over subsequent regions. This means that AREA elements with NOHREF should generally be placed before ones with the HREF attribute.

The ALT attribute is used to provide text labels which can be displayed in the status line as the mouse or other pointing device is moved over hotzones, or for constructing a textual menu for non-graphical user agents. Authors are **strongly recommended** to provide meaningful ALT attributes to support interoperability with speech-based or text-only user agents.

Sample SGML Open Catalog for HTML 3.2

This can be used with an SGML parser like nsgmls to verify that files conform to the HTML 3.2 DTD. It assumes that the DTD has been saved as the file "HTML32.dtd" and that the Latin-1 entities are in the file "ISOlat1.ent".

```
-- html32.soc: catalog for parsing HTML 3.2 documents --
SGMLDECL "HTML32.dcl"
PUBLIC "-//W3C//DTD HTML 3.2 Final//EN" HTML32.dtd
PUBLIC "-//W3C//DTD HTML 3.2 Draft//EN" HTML32.dtd
PUBLIC "-//W3C//DTD HTML 3.2//EN" HTML32.dtd
PUBLIC "ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML" ISOlat1.ent
```

SGML Declaration for HTML 3.2

This uses the 8 bit ISO Latin-1 character set. The size limits on properties like literals and tag names have been considerably increased from their HTML 2.0 values, but it is recommended that user agents avoid imposing arbitrary length limits.

```
<!SGML  "ISO 8879:1986"
   --
        SGML Declaration for HyperText Markup Language version 3.2

        With support for ISO Latin-1 and increased limits
        for tag and literal lengths etc.
   --
```

```
  CHARSET
        BASESET  "ISO 646:1983//CHARSET
                  International Reference Version
                  (IRV)//ESC 2/5 4/0"
        DESCSET  0   9   UNUSED
                 9   2   9
                 11  2   UNUSED
                 13  1   13
                 14  18  UNUSED
                 32  95  32
                 127 1   UNUSED
        BASESET  "ISO Registration Number 100//CHARSET
                  ECMA-94 Right Part of
                  Latin Alphabet Nr. 1//ESC 2/13 4/1"
        DESCSET  128  32   UNUSED
                 160  96    32

  CAPACITY   SGMLREF
             TOTALCAP        200000
             GRPCAP          150000
             ENTCAP          150000

  SCOPE    DOCUMENT
  SYNTAX
     SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
              17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 127
     BASESET  "ISO 646:1983//CHARSET
               International Reference Version
               (IRV)//ESC 2/5 4/0"
     DESCSET  0 128 0

     FUNCTION
             RE              13
             RS              10
             SPACE           32
             TAB SEPCHAR      9

     NAMING   LCNMSTRT ""
              UCNMSTRT ""
              LCNMCHAR ".-"
              UCNMCHAR ".-"
              NAMECASE GENERAL YES
                       ENTITY  NO
     DELIM    GENERAL  SGMLREF
              SHORTREF SGMLREF
     NAMES    SGMLREF
```

```
      QUANTITY SGMLREF
               ATTSPLEN 65536
               LITLEN   65536
               NAMELEN  65536
               PILEN    65536
               TAGLVL   100
               TAGLEN   65536
               GRPGTCNT 150
               GRPCNT   64

   FEATURES
     MINIMIZE
       DATATAG  NO
       OMITTAG  YES
       RANK     NO
       SHORTTAG YES
     LINK
       SIMPLE   NO
       IMPLICIT NO
       EXPLICIT NO
     OTHER
       CONCUR   NO
       SUBDOC   NO
       FORMAL   YES
   APPINFO      NONE
>
```

HTML 3.2 Document Type Definition

```
<!--
        W3C Document Type Definition for the HyperText Markup
Language
        version 3.2 as ratified by a vote of W3C member companies.
        For more information on W3C look at  URL http://www.w3.org/

        Date: Tuesday January 14th 1997

        Author: Dave Raggett <dsr@w3.org>

        HTML 3.2 aims to capture recommended practice as of early
'96
        and as such to be used as a replacement for HTML 2.0 (RFC
1866).
```

```
        Widely deployed rendering attributes are included where they
        have been shown to be interoperable. SCRIPT and STYLE are
        included to smooth the introduction of client-side scripts
        and style sheets. Browsers must avoid showing the contents
        of these element Otherwise support for them is not required.
        ID, CLASS and STYLE attributes are not included in this
version
        of HTML.
-->

<!ENTITY % HTML.Version
        "-//W3C//DTD HTML 3.2 Final//EN"

        -- Typical usage:

            <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
            <html>
            ...
            </html>
        --
        >

<!--================== Deprecated Features Switch
==========================-->

<!ENTITY % HTML.Deprecated "INCLUDE">

<!--================== Imported Names
====================================-->

<!ENTITY % Content-Type "CDATA"
        -- meaning a MIME content type, as per RFC1521
        -->

<!ENTITY % HTTP-Method "GET | POST"
        -- as per HTTP specification
        -->

<!ENTITY % URL "CDATA"
        -- The term URL means a CDATA attribute
           whose value is a Uniform Resource Locator,
           See RFC1808 (June 95) and RFC1738 (Dec 94).
        -->

<!-- Parameter Entities -->
```

```
<!ENTITY % head.misc "SCRIPT|STYLE|META|LINK" -- repeatable head
elements -->

<!ENTITY % heading "H1|H2|H3|H4|H5|H6">

<!ENTITY % list "UL | OL |  DIR | MENU">

<![ %HTML.Deprecated [
    <!ENTITY % preformatted "PRE | XMP | LISTING">
]]>

<!ENTITY % preformatted "PRE">

<!--================ Character mnemonic entities
=========================-->

<!ENTITY % ISOlat1 PUBLIC
       "ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML">
%ISOlat1;

<!--================ Entities for special symbols
=========================-->
<!-- &trade and &cbsp are not widely deployed and so not included
here -->

<!ENTITY amp    CDATA "&#38;"   -- ampersand          -->
<!ENTITY gt     CDATA "&#62;"   -- greater than       -->
<!ENTITY lt     CDATA "&#60;"   -- less than          -->

<!--=================== Text Markup
=====================================-->

<!ENTITY % font "TT | I | B  | U | STRIKE | BIG | SMALL | SUB |
SUP">

<!ENTITY % phrase "EM | STRONG | DFN | CODE | SAMP | KBD | VAR |
CITE">

<!ENTITY % special "A | IMG | APPLET | FONT | BASEFONT | BR |
SCRIPT | MAP">

<!ENTITY % form "INPUT | SELECT | TEXTAREA">

<!ENTITY % text "#PCDATA | %font | %phrase | %special | %form">

<!ELEMENT (%font|%phrase) - - (%text)*>
```

```
<!-- there are also 16 widely known color names although
  the resulting colors are implementation dependent:

   aqua, black, blue, fuchsia, gray, green, lime, maroon,
   navy, olive, purple, red, silver, teal, white, and yellow

 These colors were originally picked as being the standard
 16 colors supported with the Windows VGA palette.
 -->

<!ELEMENT FONT - - (%text)*     -- local change to font -->
<!ATTLIST FONT
    size    CDATA   #IMPLIED    -- [+]nn e.g. size="+1", size=4 --
    color   CDATA   #IMPLIED    -- #RRGGBB in hex, e.g. red:
color="#FF0000" --
    >

<!ELEMENT BASEFONT - O EMPTY    -- base font size (1 to 7)-->
<!ATTLIST BASEFONT
    size    CDATA   #IMPLIED    -- e.g. size=3 --
    >

<!ELEMENT BR    - O EMPTY    -- forced line break -->
<!ATTLIST BR
        clear (left|all|right|none) none -- control of text flow --
        >

<!--================== HTML content models
================================-->
<!--
    HTML has three basic content models:

        %text       character level elements and text strings
        %flow       block-like elements e.g. paragraphs and lists
        %bodytext   as %flow plus headers H1-H6 and ADDRESS
-->

<!ENTITY % block
     "P | %list | %preformatted | DL | DIV | CENTER |
      BLOCKQUOTE | FORM | ISINDEX | HR | TABLE">

<!-- %flow is used for DD and LI -->

<!ENTITY % flow "(%text | %block)*">
```

```
<!--================== Document Body
======================================-->

<!ENTITY % body.content "(%heading | %text | %block | ADDRESS)*">

<!ENTITY % color "CDATA" -- a color specification: #HHHHHH @@
details? -->

<!ENTITY % body-color-attrs "
        bgcolor %color #IMPLIED
        text %color #IMPLIED
        link %color #IMPLIED
        vlink %color #IMPLIED
        alink %color #IMPLIED
        ">

<!ELEMENT BODY O O  %body.content>
<!ATTLIST BODY
        background %URL #IMPLIED  -- texture tile for document
background --
        %body-color-attrs;  -- bgcolor, text, link, vlink, alink --
        >

<!ENTITY % address.content "((%text;) | P)*">

<!ELEMENT ADDRESS - - %address.content>

<!ELEMENT DIV - - %body.content>
<!ATTLIST DIV
        align   (left|center|right) #IMPLIED -- alignment of
following text --
        >

<!-- CENTER is a shorthand for DIV with ALIGN=CENTER -->
<!ELEMENT center - - %body.content>

<!--================== The Anchor Element
================================-->

<!ELEMENT A - - (%text)* -(A)>
<!ATTLIST A
        name    CDATA   #IMPLIED    -- named link end --
        href    %URL    #IMPLIED    -- URL for linked resource --
        rel     CDATA   #IMPLIED    -- forward link types --
        rev     CDATA   #IMPLIED    -- reverse link types --
        title   CDATA   #IMPLIED    -- advisory title string --
```

```
        >

<!--================== Client-side image maps
============================-->

<!-- These can be placed in the same document or grouped in a
     separate document although this isn't yet widely supported -->

<!ENTITY % SHAPE "(rect|circle|poly)">
<!ENTITY % COORDS "CDATA" -- comma separated list of numbers -->

<!ELEMENT MAP - - (AREA)*>
<!ATTLIST MAP
    name    CDATA   #IMPLIED
    >

<!ELEMENT AREA - O EMPTY>
<!ATTLIST AREA
    shape   %SHAPE  rect
    coords  %COORDS #IMPLIED  -- defines coordinates for shape --
    href    %URL    #IMPLIED  -- this region acts as hypertext link
--
    nohref (nohref) #IMPLIED  -- this region has no action --
    alt     CDATA   #REQUIRED -- needed for non-graphical user
agents --
    >

<!--================== The LINK Element
==================================-->

<!ENTITY % Types "CDATA"
        -- See Internet Draft: draft-ietf-html-relrev-00.txt
           LINK has been part of HTML since the early days
           although few browsers as yet take advantage of it.

           Relationship values can be used in principle:

                a) for document specific toolbars/menus when used
                   with the LINK element in the document head:
                b) to link to a separate style sheet
                c) to make a link to a script
                d) by stylesheets to control how collections of
                   html nodes are rendered into printed documents
                e) to make a link to a printable version of this
document
                   e.g. a postscript or pdf version
```

```
-->

<!ELEMENT LINK - O EMPTY>
<!ATTLIST LINK
        href    %URL    #IMPLIED    -- URL for linked resource --
        rel     %Types  #IMPLIED    -- forward link types --
        rev     %Types  #IMPLIED    -- reverse link types --
        title   CDATA   #IMPLIED    -- advisory title string --
        >

<!--================== Images
============================================-->

<!ENTITY % Length "CDATA"   -- nn for pixels or nn% for percentage
length -->
<!ENTITY % Pixels "NUMBER"  -- integer representing length in
pixels -->

<!-- Suggested widths are used for negotiating image size
     with the module responsible for painting the image.
     align=left or right cause image to float to margin
     and for subsequent text to wrap around image -->

<!ENTITY % IAlign "(top|middle|bottom|left|right)">

<!ELEMENT IMG    - O EMPTY --  Embedded image -->
<!ATTLIST IMG
        src     %URL     #REQUIRED  -- URL of image to embed --
        alt     CDATA    #IMPLIED   -- for display in place of
image --
        align   %IAlign  #IMPLIED   -- vertical or horizontal
alignment --
        height  %Pixels  #IMPLIED   -- suggested height in pixels --
        width   %Pixels  #IMPLIED   -- suggested width in pixels --
        border  %Pixels  #IMPLIED   -- suggested link border width
--
        hspace  %Pixels  #IMPLIED   -- suggested horizontal gutter
--
        vspace  %Pixels  #IMPLIED   -- suggested vertical gutter --
        usemap  %URL     #IMPLIED   -- use client-side image map --
        ismap   (ismap)  #IMPLIED   -- use server image map --
        >

<!-- USEMAP points to a MAP element which may be in this document
  or an external document, although the latter is not widely
supported -->
```

```
<!--================== Java APPLET tag
===================================-->
<!--
  This tag is supported by all Java enabled browsers. Applet
resources
  (including their classes) are normally loaded relative to the
document
  URL (or <BASE> element if it is defined). The CODEBASE attribute
is used
  to change this default behavior. If the CODEBASE attribute is
defined then
  it specifies a different location to find applet resources. The
value
  can be an absolute URL or a relative URL. The absolute URL is
used as is
  without modification and is not effected by the documents <BASE>
element.
  When the codebase attribute is relative, then it is relative to
the
  document URL (or <BASE> tag if defined).
-->
<!ELEMENT APPLET - - (PARAM | %text)*>
<!ATTLIST APPLET
        codebase %URL     #IMPLIED   -- code base --
        code     CDATA    #REQUIRED  -- class file --
        alt      CDATA    #IMPLIED   -- for display in place of
applet --
        name     CDATA    #IMPLIED   -- applet name --
        width    %Pixels  #REQUIRED  -- suggested width in pixels --
        height   %Pixels  #REQUIRED  -- suggested height in pixels
--
        align    %IAlign  #IMPLIED   -- vertical or horizontal
alignment --
        hspace   %Pixels  #IMPLIED   -- suggested horizontal gutter
--
        vspace   %Pixels  #IMPLIED   -- suggested vertical gutter --
        >

<!ELEMENT PARAM - O EMPTY>
<!ATTLIST PARAM
        name    NMTOKEN   #REQUIRED  -- The name of the parameter --
        value   CDATA     #IMPLIED   -- The value of the parameter
--
        >
```

```
<!--
Here is an example:

    <applet codebase="applets/NervousText"
        code=NervousText.class
        width=300
        height=50>
    <param name=text value="Java is Cool!">
    <img src=sorry.gif alt="This looks better with Java support">
    </applet>
-->

<!--=================== Horizontal Rule
====================================-->

<!ELEMENT HR    - O EMPTY>
<!ATTLIST HR
        align (left|right|center) #IMPLIED
        noshade (noshade) #IMPLIED
        size  %Pixels #IMPLIED
        width %Length #IMPLIED
        >
<!--====================
Paragraphs=============================================-->

<!ELEMENT P     - O (%text)*>
<!ATTLIST P
        align  (left|center|right) #IMPLIED
        >

<!--=================== Headings
============================================-->

<!--
  There are six levels of headers from H1 (the most important)
  to H6 (the least important).
-->

<!ELEMENT ( %heading )  - -  (%text;)*>
<!ATTLIST ( %heading )
        align  (left|center|right) #IMPLIED
        >

<!--=================== Preformatted Text
==================================-->
```

```
<!-- excludes images and changes in font size -->

<!ENTITY % pre.exclusion "IMG|BIG|SMALL|SUB|SUP|FONT">

<!ELEMENT PRE - - (%text)* -(%pre.exclusion)>
<!ATTLIST PRE
        width NUMBER #implied -- is this widely supported? --
        >

<![ %HTML.Deprecated [

<!ENTITY % literal "CDATA"
        -- historical, non-conforming parsing mode where
           the only markup signal is the end tag
           in full
        -->

<!ELEMENT (XMP|LISTING) - -  %literal>
<!ELEMENT PLAINTEXT - O %literal>

]]>

<!--================== Block-like Quotes
================================-->

<!ELEMENT BLOCKQUOTE - - %body.content>

<!--================== Lists
============================================-->

<!--
    HTML 3.2 allows you to control the sequence number for ordered
lists.
    You can set the sequence number with the START and VALUE
attributes.
    The TYPE attribute may be used to specify the rendering of
ordered
    and unordered lists.
-->

<!-- definition lists - DT for term, DD for its definition -->

<!ELEMENT DL    - -  (DT|DD)+>
<!ATTLIST DL
        compact (compact) #IMPLIED -- more compact style --
        >
```

```
<!ELEMENT DT - O  (%text)*>
<!ELEMENT DD - O  %flow;>

<!-- Ordered lists OL, and unordered lists UL -->
<!ELEMENT (OL|UL) - -  (LI)+>

<!--
       Numbering style
    1   Arabic numbers      1, 2, 3, ...
    a   lower alpha         a, b, c, ...
    A   upper alpha         A, B, C, ...
    i   lower Roman         i, ii, iii, ...
    I   upper Roman         I, II, III, ...

    The style is applied to the sequence number which by default
    is reset to 1 for the first list item in an ordered list.

    This can't be expressed directly in SGML due to case folding.
-->

<!ENTITY % OLStyle "CDATA" -- constrained to: [1|a|A|i|I] -->

<!ATTLIST OL -- ordered lists --
        type      %OLStyle   #IMPLIED   -- numbering style --
        start     NUMBER     #IMPLIED   -- starting sequence number
--
        compact  (compact)   #IMPLIED   -- reduced interitem
spacing --
        >

<!-- bullet styles -->

<!ENTITY % ULStyle "disc|square|circle">

<!ATTLIST UL -- unordered lists --
        type    (%ULStyle)   #IMPLIED   -- bullet style --
        compact (compact)    #IMPLIED   -- reduced interitem
spacing --
        >

<!ELEMENT (DIR|MENU) - -  (LI)+ -(%block)>
<!ATTLIST DIR
        compact (compact) #IMPLIED
        >
<!ATTLIST MENU
```

```
        compact (compact) #IMPLIED
        >

<!-- <DIR>              Directory list                  -->
<!-- <DIR COMPACT>      Compact list style              -->
<!-- <MENU>             Menu list                       -->
<!-- <MENU COMPACT>     Compact list style              -->

<!-- The type attribute can be used to change the bullet style
     in unordered lists and the numbering style in ordered lists -->

<!ENTITY % LIStyle "CDATA" -- constrained to: "(%ULStyle|%OLStyle)"
-->

<!ELEMENT LI - O %flow -- list item -->
<!ATTLIST LI
        type    %LIStyle     #IMPLIED   -- list item style --
        value    NUMBER      #IMPLIED   -- reset sequence number --
        >

<!--================ Forms
===============================================-->

<!ELEMENT FORM - - %body.content -(FORM)>
<!ATTLIST FORM
        action %URL #IMPLIED  -- server-side form handler --
        method (%HTTP-Method) GET -- see HTTP specification --
        enctype %Content-Type; "application/x-www-form-urlencoded"
        >

<!ENTITY % InputType
        "(TEXT | PASSWORD | CHECKBOX | RADIO | SUBMIT
            | RESET | FILE | HIDDEN | IMAGE)">

<!ELEMENT INPUT - O EMPTY>
<!ATTLIST INPUT
        type %InputType TEXT     -- what kind of widget is needed --
        name  CDATA   #IMPLIED   -- required for all but submit and
reset --
        value CDATA   #IMPLIED   -- required for radio and
checkboxes --
        checked (checked) #IMPLIED -- for radio buttons and check
boxes --
        size CDATA    #IMPLIED   -- specific to each type of field
--
        maxlength NUMBER #IMPLIED -- max chars allowed in text
```

```
fields --
        src   %URL    #IMPLIED   -- for fields with background
images --
        align %IAlign #IMPLIED   -- vertical or horizontal
alignment --
        >

<!ELEMENT SELECT - - (OPTION+)>
<!ATTLIST SELECT
        name CDATA #REQUIRED
        size NUMBER #IMPLIED
        multiple (multiple) #IMPLIED
        >

<!ELEMENT OPTION - O (#PCDATA)*>
<!ATTLIST OPTION
        selected (selected) #IMPLIED
        value  CDATA  #IMPLIED -- defaults to element content --
        >

<!-- Multi-line text input field. -->

<!ELEMENT TEXTAREA - - (#PCDATA)*>
<!ATTLIST TEXTAREA
        name CDATA #REQUIRED
        rows NUMBER #REQUIRED
        cols NUMBER #REQUIRED
        >

<!--======================= Tables
=======================================-->

<!-- Widely deployed subset of the full table standard, see RFC 1942
     e.g. at http://www.ics.uci.edu/pub/ietf/html/rfc1942.txt -->

<!-- horizontal placement of table relative to window -->
<!ENTITY % Where "(left|center|right)">

<!-- horizontal alignment attributes for cell contents -->
<!ENTITY % cell.halign
        "align  (left|center|right) #IMPLIED"
        >

<!-- vertical alignment attributes for cell contents -->
<!ENTITY % cell.valign
        "valign  (top|middle|bottom)  #IMPLIED"
```

```
        >

<!ELEMENT table - - (caption?, tr+)>
<!ELEMENT tr - O (th|td)*>
<!ELEMENT (th|td) - O %body.content>

<!ATTLIST table                         -- table element --
        align     %Where;   #IMPLIED  -- table position relative to
window --
        width     %Length   #IMPLIED  -- table width relative to
window --
        border    %Pixels   #IMPLIED  -- controls frame width
around table --
        cellspacing %Pixels #IMPLIED  -- spacing between cells --
        cellpadding %Pixels #IMPLIED  -- spacing within cells --
        >

<!ELEMENT CAPTION - - (%text;)* -- table or figure caption -->
<!ATTLIST CAPTION
        align (top|bottom) #IMPLIED
        >

<!ATTLIST tr                            -- table row --
        %cell.halign;                   -- horizontal alignment in cells
--
        %cell.valign;                   -- vertical alignment in cells --
        >

<!ATTLIST (th|td)                       -- header or data cell --
        nowrap (nowrap)  #IMPLIED  -- suppress word wrap --
        rowspan NUMBER   1         -- number of rows spanned by
cell --
        colspan NUMBER   1         -- number of cols spanned by
cell --
        %cell.halign;                   -- horizontal alignment in cell
--
        %cell.valign;                   -- vertical alignment in cell --
        width   %Pixels  #IMPLIED  -- suggested width for cell --
        height  %Pixels  #IMPLIED  -- suggested height for cell --
        >

<!--================ Document Head
=========================================-->

<!-- %head.misc defined earlier on as "SCRIPT|STYLE|META|LINK" -->
```

```
<!ENTITY % head.content "TITLE & ISINDEX? & BASE?">

<!ELEMENT HEAD O O  (%head.content) +(%head.misc)>

<!ELEMENT TITLE - -  (#PCDATA)* -(%head.misc)
          -- The TITLE element is not considered part of the flow
of text.
             It should be displayed, for example as the page header
or
             window title.
          -->

<!ELEMENT ISINDEX - O EMPTY>
<!ATTLIST ISINDEX
        prompt CDATA #IMPLIED -- prompt message -->

<!--
    The BASE element gives an absolute URL for dereferencing
relative
    URLs, e.g.

          <BASE href="http://foo.com/index.html">
          ...
          <IMG SRC="images/bar.gif">

    The image is deferenced to

          http://foo.com/images/bar.gif

   In the absence of a BASE element the document URL should be used.
   Note that this is not necessarily the same as the URL used to
   request the document, as the base URL may be overridden by an
HTTP
   header accompanying the document.
-->

<!ELEMENT BASE - O EMPTY>
<!ATTLIST BASE
        href %URL  #REQUIRED
        >

<!ELEMENT META - O EMPTY -- Generic Metainformation -->
<!ATTLIST META
        http-equiv  NAME    #IMPLIED  -- HTTP response header name
--
        name        NAME    #IMPLIED  -- metainformation name
```

```
--
        content     CDATA   #REQUIRED -- associated information
--
        >

<!-- SCRIPT/STYLE are place holders for transition to next version
of HTML -->

<!ELEMENT STYLE  - - CDATA -- placeholder for style info -->
<!ELEMENT SCRIPT - - CDATA -- placeholder for script statements -->

<!--================ Document Structure
====================================-->

<!ENTITY % version.attr "VERSION CDATA #FIXED '%HTML.Version;'">

<![ %HTML.Deprecated [
    <!ENTITY % html.content "HEAD, BODY, PLAINTEXT?">
]]>

<!ENTITY % html.content "HEAD, BODY">

<!ELEMENT HTML O O  (%html.content)>
<!ATTLIST HTML
        %version.attr;
        >
```

Character Entities for ISO Latin-1

```
<!-- (C) International Organization for Standardization 1986
     Permission to copy in any form is granted for use with
     conforming SGML systems and applications as defined in
     ISO 8879, provided this notice is included in all copies.
     This has been extended for use with HTML to cover the full
     set of codes in the range 160-255 decimal.
-->
<!-- Character entity set. Typical invocation:
     <!ENTITY % ISOlat1 PUBLIC
       "ISO 8879-1986//ENTITIES Added Latin 1//EN//HTML">
     %ISOlat1;
-->
    <!ENTITY nbsp   CDATA " " -- no-break space -->
```

```
    <!ENTITY iexcl  CDATA "&#161;" -- inverted exclamation mark -->
    <!ENTITY cent   CDATA "&#162;" -- cent sign -->
    <!ENTITY pound  CDATA "&#163;" -- pound sterling sign -->
    <!ENTITY curren CDATA "&#164;" -- general currency sign -->
    <!ENTITY yen    CDATA "&#165;" -- yen sign -->
    <!ENTITY brvbar CDATA "&#166;" -- broken (vertical) bar -->
    <!ENTITY sect   CDATA "&#167;" -- section sign -->
    <!ENTITY uml    CDATA "&#168;" -- umlaut (dieresis) -->
    <!ENTITY copy   CDATA "&#169;" -- copyright sign -->
    <!ENTITY ordf   CDATA "&#170;" -- ordinal indicator, feminine --
>
    <!ENTITY laquo  CDATA "&#171;" -- angle quotation mark, left -->
    <!ENTITY not    CDATA "&#172;" -- not sign -->
    <!ENTITY shy    CDATA "&#173;" -- soft hyphen -->
    <!ENTITY reg    CDATA "&#174;" -- registered sign -->
    <!ENTITY macr   CDATA "&#175;" -- macron -->
    <!ENTITY deg    CDATA "&#176;" -- degree sign -->
    <!ENTITY plusmn CDATA "&#177;" -- plus-or-minus sign -->
    <!ENTITY sup2   CDATA "&#178;" -- superscript two -->
    <!ENTITY sup3   CDATA "&#179;" -- superscript three -->
    <!ENTITY acute  CDATA "&#180;" -- acute accent -->
    <!ENTITY micro  CDATA "&#181;" -- micro sign -->
    <!ENTITY para   CDATA "&#182;" -- pilcrow (paragraph sign) -->
    <!ENTITY middot CDATA "&#183;" -- middle dot -->
    <!ENTITY cedil  CDATA "&#184;" -- cedilla -->
    <!ENTITY sup1   CDATA "&#185;" -- superscript one -->
    <!ENTITY ordm   CDATA "&#186;" -- ordinal indicator, masculine
-->
    <!ENTITY raquo  CDATA "&#187;" -- angle quotation mark, right --
>
    <!ENTITY frac14 CDATA "&#188;" -- fraction one-quarter -->
    <!ENTITY frac12 CDATA "&#189;" -- fraction one-half -->
    <!ENTITY frac34 CDATA "&#190;" -- fraction three-quarters -->
    <!ENTITY iquest CDATA "&#191;" -- inverted question mark -->
    <!ENTITY Agrave CDATA "&#192;" -- capital A, grave accent -->
    <!ENTITY Aacute CDATA "&#193;" -- capital A, acute accent -->
    <!ENTITY Acirc  CDATA "&#194;" -- capital A, circumflex accent
-->
    <!ENTITY Atilde CDATA "&#195;" -- capital A, tilde -->
    <!ENTITY Auml   CDATA "&#196;" -- capital A, dieresis or umlaut
mark -->
    <!ENTITY Aring  CDATA "&#197;" -- capital A, ring -->
    <!ENTITY AElig  CDATA "&#198;" -- capital AE diphthong
(ligature) -->
    <!ENTITY Ccedil CDATA "&#199;" -- capital C, cedilla -->
    <!ENTITY Egrave CDATA "&#200;" -- capital E, grave accent -->
```

```
    <!ENTITY Eacute CDATA "&#201;" -- capital E, acute accent -->
    <!ENTITY Ecirc  CDATA "&#202;" -- capital E, circumflex accent
-->
    <!ENTITY Euml   CDATA "&#203;" -- capital E, dieresis or umlaut
mark -->
    <!ENTITY Igrave CDATA "&#204;" -- capital I, grave accent -->
    <!ENTITY Iacute CDATA "&#205;" -- capital I, acute accent -->
    <!ENTITY Icirc  CDATA "&#206;" -- capital I, circumflex accent
-->
    <!ENTITY Iuml   CDATA "&#207;" -- capital I, dieresis or umlaut
mark -->
    <!ENTITY ETH    CDATA "&#208;" -- capital Eth, Icelandic -->
    <!ENTITY Ntilde CDATA "&#209;" -- capital N, tilde -->
    <!ENTITY Ograve CDATA "&#210;" -- capital O, grave accent -->
    <!ENTITY Oacute CDATA "&#211;" -- capital O, acute accent -->
    <!ENTITY Ocirc  CDATA "&#212;" -- capital O, circumflex accent
-->
    <!ENTITY Otilde CDATA "&#213;" -- capital O, tilde -->
    <!ENTITY Ouml   CDATA "&#214;" -- capital O, dieresis or umlaut
mark -->
    <!ENTITY times  CDATA "&#215;" -- multiply sign -->
    <!ENTITY Oslash CDATA "&#216;" -- capital O, slash -->
    <!ENTITY Ugrave CDATA "&#217;" -- capital U, grave accent -->
    <!ENTITY Uacute CDATA "&#218;" -- capital U, acute accent -->
    <!ENTITY Ucirc  CDATA "&#219;" -- capital U, circumflex accent
-->
    <!ENTITY Uuml   CDATA "&#220;" -- capital U, dieresis or umlaut
mark -->
    <!ENTITY Yacute CDATA "&#221;" -- capital Y, acute accent -->
    <!ENTITY THORN  CDATA "&#222;" -- capital THORN, Icelandic -->
    <!ENTITY szlig  CDATA "&#223;" -- small sharp s, German (sz
ligature) -->
    <!ENTITY agrave CDATA "&#224;" -- small a, grave accent -->
    <!ENTITY aacute CDATA "&#225;" -- small a, acute accent -->
    <!ENTITY acirc  CDATA "&#226;" -- small a, circumflex accent -->
    <!ENTITY atilde CDATA "&#227;" -- small a, tilde -->
    <!ENTITY auml   CDATA "&#228;" -- small a, dieresis or umlaut
mark -->
    <!ENTITY aring  CDATA "&#229;" -- small a, ring -->
    <!ENTITY aelig  CDATA "&#230;" -- small ae diphthong (ligature)
-->
    <!ENTITY ccedil CDATA "&#231;" -- small c, cedilla -->
    <!ENTITY egrave CDATA "&#232;" -- small e, grave accent -->
    <!ENTITY eacute CDATA "&#233;" -- small e, acute accent -->
    <!ENTITY ecirc  CDATA "&#234;" -- small e, circumflex accent -->
    <!ENTITY euml   CDATA "&#235;" -- small e, dieresis or umlaut
```

```
mark -->
    <!ENTITY igrave CDATA "&#236;" -- small i, grave accent -->
    <!ENTITY iacute CDATA "&#237;" -- small i, acute accent -->
    <!ENTITY icirc  CDATA "&#238;" -- small i, circumflex accent -->
    <!ENTITY iuml   CDATA "&#239;" -- small i, dieresis or umlaut
mark -->
    <!ENTITY eth    CDATA "&#240;" -- small eth, Icelandic -->
    <!ENTITY ntilde CDATA "&#241;" -- small n, tilde -->
    <!ENTITY ograve CDATA "&#242;" -- small o, grave accent -->
    <!ENTITY oacute CDATA "&#243;" -- small o, acute accent -->
    <!ENTITY ocirc  CDATA "&#244;" -- small o, circumflex accent -->
    <!ENTITY otilde CDATA "&#245;" -- small o, tilde -->
    <!ENTITY ouml   CDATA "&#246;" -- small o, dieresis or umlaut
mark -->
    <!ENTITY divide CDATA "&#247;" -- divide sign -->
    <!ENTITY oslash CDATA "&#248;" -- small o, slash -->
    <!ENTITY ugrave CDATA "&#249;" -- small u, grave accent -->
    <!ENTITY uacute CDATA "&#250;" -- small u, acute accent -->
    <!ENTITY ucirc  CDATA "&#251;" -- small u, circumflex accent -->
    <!ENTITY uuml   CDATA "&#252;" -- small u, dieresis or umlaut
mark -->
    <!ENTITY yacute CDATA "&#253;" -- small y, acute accent -->
    <!ENTITY thorn  CDATA "&#254;" -- small thorn, Icelandic -->
    <!ENTITY yuml   CDATA "&#255;" -- small y, dieresis or umlaut
mark -->
```

Table of printable Latin-1 Character codes

0	32		64	@	96	‘	128	160		192	À	224	à
1	33	!	65	A	97	a	129	161	¡	193	Á	225	á
2	34	"	66	B	98	b	130	162	¢	194	Â	226	â
3	35	#	67	C	99	c	131	163	£	195	Ã	227	ã
4	36	$	68	D	100	d	132	164	¤	196	Ä	228	ä
5	37	%	69	E	101	e	133	165	¥	197	Å	229	å
6	38	&	70	F	102	f	134	166	¦	198	Æ	230	æ
7	39	’	71	G	103	g	135	167	§	199	Ç	231	ç
8	40	(	72	H	104	h	136	168	¨	200	È	232	è
9	41	)	73	I	105	i	137	169	©	201	É	233	é
10	42	*	74	J	106	j	138	170	ª	202	Ê	234	ê
11	43	+	75	K	107	k	139	171	«	203	Ë	235	ë
12	44	,	76	L	108	l	140	172	¬	204	Ì	236	ì
13	45	-	77	M	109	m	141	173	-	205	Í	237	í
14	46	.	78	N	110	n	142	174	®	206	Î	238	î
15	47	/	79	O	111	o	143	175	¯	207	Ï	239	ï
16	48	0	80	P	112	p	144	176	°	208	Ð	240	ð
17	49	1	81	Q	113	q	145	177	±	209	Ñ	241	ñ
18	50	2	82	R	114	r	146	178	²	210	Ò	242	ò
19	51	3	83	S	115	s	147	179	³	211	Ó	243	ó
20	52	4	84	T	116	t	148	180	´	212	Ô	244	ô
21	53	5	85	U	117	u	149	181	µ	213	Õ	245	õ
22	54	6	86	V	118	v	150	182	¶	214	Ö	246	ö
23	55	7	87	W	119	w	151	183	·	215	×	247	÷
24	56	8	88	X	120	x	152	184	¸	216	Ø	248	ø
25	57	9	89	Y	121	y	153	185	¹	217	Ù	249	ù
26	58	:	90	Z	122	z	154	186	º	218	Ú	250	ú
27	59	;	91	[	123	{	155	187	»	219	Û	251	û
28	60	<	92	\	124	|	156	188	¼	220	Ü	252	ü
29	61	=	93	]	125	}	157	189	½	221	Ý	253	ý
30	62	>	94	^	126	~	158	190	¾	222	Þ	254	þ
31	63	?	95	_	127		159	191	¿	223	ß	255	ÿ

Acknowledgements

The author would like to thank the members of the W3C HTML Editorial Review Board, members of the W3C staff, and the many other people who have contributed to this specification.

Further Reading

The World Wide Web Consortium

Further information on W3C activities and pointers to the status of work on HTML and HTTP etc. can be found at http://www.w3.org/. Further information on HTML in particular can be found at http://www.w3.org/pub/WWW/MarkUp/.

HTML 2.0 (RFC1866)

By Tim Berners-Lee and Dan Connolly, November 1995. Defines the Hypertext Markup Language Specification Version 2.0. Available from ftp://ds.internic.net/rfc/rfc1866.txt.

Form-based File Upload in HTML (RFC1867)

By E. Nebel and L. Masinter, November 1995. Describes extensions to HTML 2.0 (RFC1866) to support file upload from HTML forms. Available from ftp://ds.internic.net/rfc/rfc1867.txt.

HTML Tables (RFC1942)

By Dave Raggett, May 1996. This defines the HTML table model. It is a superset of the table model defined by HTML 3.2. Available from ftp://ds.internic.net/rfc/rfc1942.txt, or as a W3C working draft at http://www.w3.org/pub/WWW/TR/WD-tables.

A Lexical Analyzer for HTML and Basic SGML

By Dan Connolly, June 1996. Describes lexical considerations for parsing HTML documents. Available from http://www.w3.org/pub/WWW/TR/WD-html-lex

The Hypertext Transfer Protocol (HTTP)

Further information of HTTP can be found at: http://www.w3.org/pub/WWW/Protocols.

A Standard Default Color Space for the Internet - sRGB

By Michael Stokes, Mathew Anderson, Srinivasan Chandrasekar and Ricardo Motta, November 1996. Available from: http://www.w3.org/pub/WWW/Graphics/Color/sRGB.html

This provides a precise definition for RGB that allows sRGB images to be reproduced accurately on different platforms and media under varying ambient lighting conditions.

table of contents

XHTML™ 1.0: The Extensible HyperText Markup Language

A Reformulation of HTML 4 in XML 1.0

W3C Recommendation 26 January 2000

This version:
http://www.w3.org/TR/2000/REC-xhtml1-20000126
(Postscript version, PDF version, ZIP archive, or Gzip'd TAR archive)
Latest version:
http://www.w3.org/TR/xhtml1
Previous version:
http://www.w3.org/TR/1999/PR-xhtml1-19991210
Authors:
See acknowledgements.

Abstract

This specification defines XHTML 1.0, a reformulation of HTML 4 as an XML 1.0 application, and three DTDs corresponding to the ones defined by HTML 4. The semantics of the elements and their attributes are defined in the W3C Recommendation for HTML 4. These semantics provide the foundation for future extensibility of XHTML. Compatibility with existing HTML user agents is possible by following a small set of guidelines.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document has been produced as part of the W3C HTML Activity. The goals of the HTML Working Group *(members only)* are discussed in the HTML Working Group charter *(members only)*.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

Public discussion on HTML features takes place on the mailing list www-html@w3.org (archive).

Please report errors in this document to www-html-editor@w3.org.

The list of known errors in this specification is available at http://www.w3.org/2000/01/REC-xhtml1-20000126-errata.

Contents

1. What is XHTML?

XHTML is a family of current and future document types and modules that reproduce, subset, and extend HTML 4 [HTML]. XHTML family document types are XML based, and ultimately are designed to work in conjunction with XML-based user agents. The details of this family and its evolution are discussed in more detail in the section on Future Directions.

XHTML 1.0 (this specification) is the first document type in the XHTML family. It is a reformulation of the three HTML 4 document types as applications of XML 1.0 [XML]. It is intended to be used as a language for content that is both XML-conforming and, if some simple guidelines are followed, operates in HTML 4 conforming user agents. Developers who migrate their content to XHTML 1.0 will realize the following benefits:

- XHTML documents are XML conforming. As such, they are readily viewed, edited, and validated with standard XML tools.
- XHTML documents can be written to to operate as well or better than they did before in existing HTML 4-conforming user agents as well as in new, XHTML 1.0 conforming user agents.
- XHTML documents can utilize applications (e.g. scripts and applets) that rely upon either the HTML Document Object Model or the XML Document Object Model [DOM].
- As the XHTML family evolves, documents conforming to XHTML 1.0 will be more likely to interoperate within and among various XHTML environments.

The XHTML family is the next step in the evolution of the Internet. By

migrating to XHTML today, content developers can enter the XML world with all of its attendant benefits, while still remaining confident in their content's backward and future compatibility.

1.1 What is HTML 4?

HTML 4 [HTML] is an SGML (Standard Generalized Markup Language) application conforming to International Standard ISO 8879, and is widely regarded as the standard publishing language of the World Wide Web.

SGML is a language for describing markup languages, particularly those used in electronic document exchange, document management, and document publishing. HTML is an example of a language defined in SGML.

SGML has been around since the middle 1980's and has remained quite stable. Much of this stability stems from the fact that the language is both feature-rich and flexible. This flexibility, however, comes at a price, and that price is a level of complexity that has inhibited its adoption in a diversity of environments, including the World Wide Web.

HTML, as originally conceived, was to be a language for the exchange of scientific and other technical documents, suitable for use by non-document specialists. HTML addressed the problem of SGML complexity by specifying a small set of structural and semantic tags suitable for authoring relatively simple documents. In addition to simplifying the document structure, HTML added support for hypertext. Multimedia capabilities were added later.

In a remarkably short space of time, HTML became wildly popular and rapidly outgrew its original purpose. Since HTML's inception, there has been rapid invention of new elements for use within HTML (as a standard) and for adapting HTML to vertical, highly specialized, markets. This plethora of new elements has led to compatibility problems for documents across different platforms.

As the heterogeneity of both software and platforms rapidly proliferate, it is clear that the suitability of 'classic' HTML 4 for use on these platforms is somewhat limited.

1.2 What is XML?

XML™ is the shorthand for Extensible Markup Language, and is an acronym of Extensible Markup Language [XML].

XML was conceived as a means of regaining the power and flexibility of

SGML without most of its complexity. Although a restricted form of SGML, XML nonetheless preserves most of SGML's power and richness, and yet still retains all of SGML's commonly used features.

While retaining these beneficial features, XML removes many of the more complex features of SGML that make the authoring and design of suitable software both difficult and costly.

1.3 Why the need for XHTML?

The benefits of migrating to XHTML 1.0 are described above. Some of the benefits of migrating to XHTML in general are:

- Document developers and user agent designers are constantly discovering new ways to express their ideas through new markup. In XML, it is relatively easy to introduce new elements or additional element attributes. The XHTML family is designed to accommodate these extensions through XHTML modules and techniques for developing new XHTML-conforming modules (described in the forthcoming XHTML Modularization specification). These modules will permit the combination of existing and new feature sets when developing content and when designing new user agents.
- Alternate ways of accessing the Internet are constantly being introduced. Some estimates indicate that by the year 2002, 75% of Internet document viewing will be carried out on these alternate platforms. The XHTML family is designed with general user agent interoperability in mind. Through a new user agent and document profiling mechanism, servers, proxies, and user agents will be able to perform best effort content transformation. Ultimately, it will be possible to develop XHTML-conforming content that is usable by any XHTML-conforming user agent.

2. Definitions

2.1 Terminology

The following terms are used in this specification. These terms extend the definitions in [RFC2119] in ways based upon similar definitions in ISO/IEC 9945-1:1990 [POSIX.1]:

Implementation-defined
: A value or behavior is implementation-defined when it is left to the implementation to define [and document] the corresponding

requirements for correct document construction.

May

With respect to implementations, the word "may" is to be interpreted as an optional feature that is not required in this specification but can be provided. With respect to Document Conformance, the word "may" means that the optional feature must not be used. The term "optional" has the same definition as "may".

Must

In this specification, the word "must" is to be interpreted as a mandatory requirement on the implementation or on Strictly Conforming XHTML Documents, depending upon the context. The term "shall" has the same definition as "must".

Reserved

A value or behavior is unspecified, but it is not allowed to be used by Conforming Documents nor to be supported by a Conforming User Agents.

Should

With respect to implementations, the word "should" is to be interpreted as an implementation recommendation, but not a requirement. With respect to documents, the word "should" is to be interpreted as recommended programming practice for documents and a requirement for Strictly Conforming XHTML Documents.

Supported

Certain facilities in this specification are optional. If a facility is supported, it behaves as specified by this specification.

Unspecified

When a value or behavior is unspecified, the specification defines no portability requirements for a facility on an implementation even when faced with a document that uses the facility. A document that requires specific behavior in such an instance, rather than tolerating any behavior when using that facility, is not a Strictly Conforming XHTML Document.

2.2 General Terms

Attribute

An attribute is a parameter to an element declared in the DTD. An attribute's type and value range, including a possible default value, are defined in the DTD.

DTD

A DTD, or document type definition, is a collection of XML declarations that, as a collection, defines the legal structure, *elements*, and *attributes* that are available for use in a document that complies to the DTD.

Document

A document is a stream of data that, after being combined with any other streams it references, is structured such that it holds information contained within *elements* that are organized as defined in the associated *DTD*. See Document Conformance for more information.

Element
An element is a document structuring unit declared in the *DTD*. The element's content model is defined in the *DTD*, and additional semantics may be defined in the prose description of the element.

Facilities
Functionality includes *elements*, *attributes*, and the semantics associated with those *elements* and *attributes*. An implementation supporting that functionality is said to provide the necessary facilities.

Implementation
An implementation is a system that provides collection of *facilities* and services that supports this specification. See User Agent Conformance for more information.

Parsing
Parsing is the act whereby a *document* is scanned, and the information contained within the *document* is filtered into the context of the *elements* in which the information is structured.

Rendering
Rendering is the act whereby the information in a *document* is presented. This presentation is done in the form most appropriate to the environment (e.g. aurally, visually, in print).

User Agent
A user agent is an *implementation* that retrieves and processes XHTML documents. See User Agent Conformance for more information.

Validation
Validation is a process whereby *documents* are verified against the associated *DTD*, ensuring that the structure, use of *elements*, and use of *attributes* are consistent with the definitions in the *DTD*.

Well-formed
A *document* is well-formed when it is structured according to the rules defined in Section 2.1 of the XML 1.0 Recommendation [XML].

Basically, this definition states that elements, delimited by their start and end tags, are nested properly within one another.

3. Normative Definition of XHTML 1.0

3.1 Document Conformance

This version of XHTML provides a definition of strictly conforming XHTML documents, which are restricted to tags and attributes from the XHTML namespace. See Section 3.1.2 for information on using XHTML with other

namespaces, for instance, to include metadata expressed in RDF within XHTML documents.

3.1.1 Strictly Conforming Documents

A Strictly Conforming XHTML Document is a document that requires only the facilities described as mandatory in this specification. Such a document must meet all of the following criteria:

1. It must validate against one of the three DTDs found in Appendix A.

2. The root element of the document must be **<html>**.

3. The root element of the document must designate the XHTML namespace using the **xmlns** attribute [XMLNAMES]. The namespace for XHTML is defined to be **http://www.w3.org/1999/xhtml**.

4. There must be a DOCTYPE declaration in the document prior to the root element. The public identifier included in the DOCTYPE declaration must reference one of the three DTDs found in Appendix A using the respective Formal Public Identifier. The system identifier may be changed to reflect local system conventions.

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "DTD/xhtml1-strict.dtd">

<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//
EN"
     "DTD/xhtml1-transitional.dtd">

<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
     "DTD/xhtml1-frameset.dtd">
```

Here is an example of a minimal XHTML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:
lang="en" lang="en">
```

```
    <head>
      <title>Virtual Library</title>
    </head>
    <body>
      <p>Moved to <a href="http://vlib.org/">vlib.
org</a>.</p>
    </body>
  </html>
```

Note that in this example, the XML declaration is included. An XML declaration like the one above is not required in all XML documents. XHTML document authors are strongly encouraged to use XML declarations in all their documents. Such a declaration is required when the character encoding of the document is other than the default UTF-8 or UTF-16.

3.1.2 Using XHTML with other namespaces

The XHTML namespace may be used with other XML namespaces as per [XMLNAMES], although such documents are not strictly conforming XHTML 1.0 documents as defined above. Future work by W3C will address ways to specify conformance for documents involving multiple namespaces.

The following example shows the way in which XHTML 1.0 could be used in conjunction with the MathML Recommendation:

```
  <html xmlns="http://www.w3.org/1999/xhtml" xml:
  lang="en" lang="en">
    <head>
      <title>A Math Example</title>
    </head>
    <body>
      <p>The following is MathML markup:</p>
      <math xmlns="http://www.w3.org/1998/Math/
  MathML">
        <apply> <log/>
          <logbase>
            <cn> 3 </cn>
          </logbase>
          <ci> x </ci>
        </apply>
      </math>
    </body>
  </html>
```

The following example shows the way in which XHTML 1.0 markup could be

incorporated into another XML namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- initially, the default namespace is "books"
-->
<book xmlns='urn:loc.gov:books'
    xmlns:isbn='urn:ISBN:0-395-36341-6' xml:
lang="en" lang="en">
  <title>Cheaper by the Dozen</title>
  <isbn:number>1568491379</isbn:number>
  <notes>
    <!-- make HTML the default namespace for a
hypertext commentary -->
    <p xmlns='http://www.w3.org/1999/xhtml'>
        This is also available <a href="http://
www.w3.org/">online</a>.
    </p>
  </notes>
</book>
```

3.2 User Agent Conformance

A conforming user agent must meet all of the following criteria:

1. In order to be consistent with the XML 1.0 Recommendation [XML], the user agent must parse and evaluate an XHTML document for well-formedness. If the user agent claims to be a validating user agent, it must also validate documents against their referenced DTDs according to [XML].
2. When the user agent claims to support facilities defined within this specification or required by this specification through normative reference, it must do so in ways consistent with the facilities' definition.
3. When a user agent processes an XHTML document as generic XML, it shall only recognize attributes of type **ID** (e.g. the **id** attribute on most XHTML elements) as fragment identifiers.
4. If a user agent encounters an element it does not recognize, it must render the element's content.
5. If a user agent encounters an attribute it does not recognize, it must ignore the entire attribute specification (i.e., the attribute and its value).
6. If a user agent encounters an attribute value it doesn't recognize, it must use the default attribute value.
7. If it encounters an entity reference (other than one of the predefined entities) for which the User Agent has processed no declaration (which could happen if the declaration is in the external subset which the User Agent hasn't read), the entity reference should be rendered as the

characters (starting with the ampersand and ending with the semi-colon) that make up the entity reference.

8. When rendering content, User Agents that encounter characters or character entity references that are recognized but not renderable should display the document in such a way that it is obvious to the user that normal rendering has not taken place.
9. The following characters are defined in [XML] as whitespace characters:
    - Space ( )
    - Tab (	)
    - Carriage return (
)
    - Line feed (
)

    The XML processor normalizes different system's line end codes into one single line-feed character, that is passed up to the application. The XHTML user agent in addition, must treat the following characters as whitespace:

    - Form feed ()
    - Zero-width space ()

    In elements where the 'xml:space' attribute is set to 'preserve', the user agent must leave all whitespace characters intact (with the exception of leading and trailing whitespace characters, which should be removed). Otherwise, whitespace is handled according to the following rules:

    - All whitespace surrounding block elements should be removed.
    - Comments are removed entirely and do not affect whitespace handling. One whitespace character on either side of a comment is treated as two white space characters.
    - Leading and trailing whitespace inside a block element must be removed.
    - Line feed characters within a block element must be converted into a space (except when the 'xml:space' attribute is set to 'preserve').
    - A sequence of white space characters must be reduced to a single space character (except when the 'xml:space' attribute is set to 'preserve').
    - With regard to rendition, the User Agent should render the content in a manner appropriate to the language in which the content is written. In languages whose primary script is Latinate, the ASCII space character is typically used to encode both grammatical word boundaries and typographic whitespace; in languages whose script is related to Nagari (e.g., Sanskrit, Thai, etc.), grammatical boundaries may be encoded using the ZW

'space' character, but will not typically be represented by typographic whitespace in rendered output; languages using Arabiform scripts may encode typographic whitespace using a space character, but may also use the ZW space character to delimit 'internal' grammatical boundaries (what look like words in Arabic to an English eye frequently encode several words, e.g. 'kitAbuhum' = 'kitAbu-hum' = 'book them' == their book); and languages in the Chinese script tradition typically neither encode such delimiters nor use typographic whitespace in this way.

Whitespace in attribute values is processed according to [XML].

4. Differences with HTML 4

Due to the fact that XHTML is an XML application, certain practices that were perfectly legal in SGML-based HTML 4 [HTML] must be changed.

4.1 Documents must be well-formed

Well-formedness is a new concept introduced by [XML]. Essentially this means that all elements must either have closing tags or be written in a special form (as described below), and that all the elements must nest.

Although overlapping is illegal in SGML, it was widely tolerated in existing browsers.

CORRECT: nested elements.

** here is an emphasized paragraph. **

INCORRECT: overlapping elements

here is an emphasized paragraph.

4.2 Element and attribute names must be in lower case

XHTML documents must use lower case for all HTML element and attribute names. This difference is necessary because XML is case-sensitive e.g. and <LI> are different tags.

4.3 For non-empty elements, end tags are required

In SGML-based HTML 4 certain elements were permitted to omit the end tag; with the elements that followed implying closure. This omission is not permitted in XML-based XHTML. All elements other than those declared in the DTD as **`EMPTY`** must have an end tag.

CORRECT: terminated elements

```
<p>here is a paragraph.</p><p>here is another paragraph.</p>
```

INCORRECT: unterminated elements

```
<p>here is a paragraph.<p>here is another paragraph.
```

4.4 Attribute values must always be quoted

All attribute values must be quoted, even those which appear to be numeric.

CORRECT: quoted attribute values

```
<table rows="3">
```

INCORRECT: unquoted attribute values

```
<table rows=3>
```

4.5 Attribute Minimization

XML does not support attribute minimization. Attribute-value pairs must be written in full. Attribute names such as **`compact`** and **`checked`** cannot occur in elements without their value being specified.

CORRECT: unminimized attributes

```
<dl compact="compact">
```

INCORRECT: minimized attributes

```
<dl compact>
```

4.6 Empty Elements

Empty elements must either have an end tag or the start tag must end with **/>**. For instance, **
** or **<hr></hr>**. See HTML Compatibility Guidelines for information on ways to ensure this is backward compatible with HTML 4 user agents.

CORRECT: terminated empty tags

```
<br/><hr/>
```

INCORRECT: unterminated empty tags

```
<br><hr>
```

4.7 Whitespace handling in attribute values

In attribute values, user agents will strip leading and trailing whitespace from attribute values and map sequences of one or more whitespace characters (including line breaks) to a single inter-word space (an ASCII space character for western scripts). See Section 3.3.3 of [XML].

4.8 Script and Style elements

In XHTML, the script and style elements are declared as having **#PCDATA** content. As a result, **<** and **&** will be treated as the start of markup, and entities such as **<** and **&** will be recognized as entity references by the XML processor to **<** and **&** respectively. Wrapping the content of the script or style element within a **CDATA** marked section avoids the expansion of these entities.

```
<script>
 <![CDATA[
 ... unescaped script content ...
 ]]>
 </script>
```

CDATA sections are recognized by the XML processor and appear as nodes in the Document Object Model, see Section 1.3 of the DOM Level 1 Recommendation [DOM].

An alternative is to use external script and style documents.

4.9 SGML exclusions

SGML gives the writer of a DTD the ability to exclude specific elements from being contained within an element. Such prohibitions (called "exclusions") are not possible in XML.

For example, the HTML 4 Strict DTD forbids the nesting of an '**a**' element within another '**a**' element to any descendant depth. It is not possible to spell out such prohibitions in XML. Even though these prohibitions cannot be defined in the DTD, certain elements should not be nested. A summary of such elements and the elements that should not be nested in them is found in the normative Appendix B.

4.10 The elements with 'id' and 'name' attributes

HTML 4 defined the **name** attribute for the elements **a**, **applet**, **form**, **frame**, **iframe**, **img**, and **map**. HTML 4 also introduced the **id** attribute. Both of these attributes are designed to be used as fragment identifiers.

In XML, fragment identifiers are of type **ID**, and there can only be a single attribute of type **ID** per element. Therefore, in XHTML 1.0 the **id** attribute is defined to be of type **ID**. In order to ensure that XHTML 1.0 documents are well-structured XML documents, XHTML 1.0 documents MUST use the **id** attribute when defining fragment identifiers, even on elements that historically have also had a **name** attribute. See the HTML Compatibility Guidelines for information on ensuring such anchors are backwards compatible when serving XHTML documents as media type **text/html**.

Note that in XHTML 1.0, the **name** attribute of these elements is formally deprecated, and will be removed in a subsequent version of XHTML.

5. Compatibility Issues

Although there is no requirement for XHTML 1.0 documents to be compatible with existing user agents, in practice this is easy to accomplish. Guidelines for creating compatible documents can be found in Appendix C.

5.1 Internet Media Type

As of the publication of this recommendation, the general recommended MIME labeling for XML-based applications has yet to be resolved.

However, XHTML Documents which follow the guidelines set forth in Appendix C, "HTML Compatibility Guidelines" may be labeled with the Internet Media Type "text/html", as they are compatible with most HTML browsers. This document makes no recommendation about MIME labeling of other XHTML documents.

6. Future Directions

XHTML 1.0 provides the basis for a family of document types that will extend and subset XHTML, in order to support a wide range of new devices and applications, by defining modules and specifying a mechanism for combining these modules. This mechanism will enable the extension and sub-setting of XHTML 1.0 in a uniform way through the definition of new modules.

6.1 Modularizing HTML

As the use of XHTML moves from the traditional desktop user agents to other platforms, it is clear that not all of the XHTML elements will be required on all platforms. For example a hand held device or a cell-phone may only support a subset of XHTML elements.

The process of modularization breaks XHTML up into a series of smaller element sets. These elements can then be recombined to meet the needs of different communities.

These modules will be defined in a later W3C document.

6.2 Subsets and Extensibility

Modularization brings with it several advantages:

- It provides a formal mechanism for sub-setting XHTML.
- It provides a formal mechanism for extending XHTML.
- It simplifies the transformation between document types.
- It promotes the reuse of modules in new document types.

6.3 Document Profiles

A document profile specifies the syntax and semantics of a set of documents.

Conformance to a document profile provides a basis for interoperability guarantees. The document profile specifies the facilities required to process documents of that type, e.g. which image formats can be used, levels of scripting, style sheet support, and so on.

For product designers this enables various groups to define their own standard profile.

For authors this will obviate the need to write several different versions of documents for different clients.

For special groups such as chemists, medical doctors, or mathematicians this allows a special profile to be built using standard HTML elements plus a group of elements geared to the specialist's needs.

Appendix A. DTDs

This appendix is normative.

These DTDs and entity sets form a normative part of this specification. The complete set of DTD files together with an XML declaration and SGML Open Catalog is included in the zip file for this specification.

A.1 Document Type Definitions

These DTDs approximate the HTML 4 DTDs. It is likely that when the DTDs are modularized, a method of DTD construction will be employed that corresponds more closely to HTML 4.

- XHTML-1.0-Strict
- XHTML-1.0-Transitional
- XHTML-1.0-Frameset

A.2 Entity Sets

The XHTML entity sets are the same as for HTML 4, but have been modified to be valid XML 1.0 entity declarations. Note the entity for the Euro currency sign (`&euro;` or `&#8364;` or `&#x20AC;`) is defined as part of the special characters.

- Latin-1 characters
- Special characters
- Symbols

Appendix B. Element Prohibitions

This appendix is normative.

The following elements have prohibitions on which elements they can contain (see Section 4.9). This prohibition applies to all depths of nesting, i.e. it contains all the descendant elements.

a
: cannot contain other **a** elements.

pre
: cannot contain the **img**, **object**, **big**, **small**, **sub**, or **sup** elements.

button
: cannot contain the **input**, **select**, **textarea**, **label**, **button**, **form**, **fieldset**, **iframe** or **isindex** elements.

label
: cannot contain other **label** elements.

form
: cannot contain other **form** elements.

Appendix C. HTML Compatibility Guidelines

This appendix is informative.

This appendix summarizes design guidelines for authors who wish their XHTML documents to render on existing HTML user agents.

C.1 Processing Instructions

Be aware that processing instructions are rendered on some user agents. However, also note that when the XML declaration is not included in a document, the document can only use the default character encodings UTF-8 or UTF-16.

C.2 Empty Elements

Include a space before the trailing `/` and `>` of empty elements, e.g. `<br />`, `<hr />` and `<img src="karen.jpg" alt="Karen" />`. Also, use the minimized tag syntax for empty elements, e.g. `<br />`, as the alternative syntax `<br></br>` allowed by XML gives uncertain results in many existing user agents.

C.3 Element Minimization and Empty Element Content

Given an empty instance of an element whose content model is not `EMPTY` (for example, an empty title or paragraph) do not use the minimized form (e.g. use `<p> </p>` and not `<p />`).

C.4 Embedded Style Sheets and Scripts

Use external style sheets if your style sheet uses `<` or `&` or `]]>` or `--`. Use external scripts if your script uses `<` or `&` or `]]>` or `--`. Note that XML parsers are permitted to silently remove the contents of comments. Therefore, the historical practice of "hiding" scripts and style sheets within comments to make the documents backward compatible is likely to not work as expected in XML-based implementations.

C.5 Line Breaks within Attribute Values

Avoid line breaks and multiple whitespace characters within attribute values. These are handled inconsistently by user agents.

C.6 Isindex

Don't include more than one `isindex` element in the document `head`. The `isindex` element is deprecated in favor of the `input` element.

C.7 The `lang` and `xml:lang` Attributes

Use both the `lang` and `xml:lang` attributes when specifying the language of an element. The value of the `xml:lang` attribute takes precedence.

C.8 Fragment Identifiers

In XML, URIs [RFC2396] that end with fragment identifiers of the form **"#foo"** do not refer to elements with an attribute **name="foo"**; rather, they refer to elements with an attribute defined to be of type **ID**, e.g., the **id** attribute in HTML 4. Many existing HTML clients don't support the use of **ID**-type attributes in this way, so identical values may be supplied for both of these attributes to ensure maximum forward and backward compatibility (e.g., **...**).

Further, since the set of legal values for attributes of type **ID** is much smaller than for those of type **CDATA**, the type of the **name** attribute has been changed to **NMTOKEN**. This attribute is constrained such that it can only have the same values as type **ID**, or as the **Name** production in XML 1.0 Section 2.5, production 5. Unfortunately, this constraint cannot be expressed in the XHTML 1.0 DTDs. Because of this change, care must be taken when converting existing HTML documents. The values of these attributes must be unique within the document, valid, and any references to these fragment identifiers (both internal and external) must be updated should the values be changed during conversion.

Finally, note that XHTML 1.0 has deprecated the **name** attribute of the **a**, **applet**, **form**, **frame**, **iframe**, **img**, and **map** elements, and it will be removed from XHTML in subsequent versions.

C.9 Character Encoding

To specify a character encoding in the document, use both the encoding attribute specification on the xml declaration (e.g. **<?xml version="1.0" encoding="EUC-JP"?>**) and a meta http-equiv statement (e.g. **<meta http-equiv="Content-type" content='text/html; charset="EUC-JP"' />**). The value of the encoding attribute of the xml processing instruction takes precedence.

C.10 Boolean Attributes

Some HTML user agents are unable to interpret boolean attributes when these appear in their full (non-minimized) form, as required by XML 1.0. Note this problem doesn't affect user agents compliant with HTML 4. The following attributes are involved: **compact**, **nowrap**, **ismap**, **declare**, **noshade**, **checked**, **disabled**, **readonly**, **multiple**, **selected**, **noresize**, **defer**.

C.11 Document Object Model and XHTML

The Document Object Model level 1 Recommendation [DOM] defines

document object model interfaces for XML and HTML 4. The HTML 4 document object model specifies that HTML element and attribute names are returned in upper-case. The XML document object model specifies that element and attribute names are returned in the case they are specified. In XHTML 1.0, elements and attributes are specified in lower-case. This apparent difference can be addressed in two ways:

1. Applications that access XHTML documents served as Internet media type **`text/html`** via the DOM can use the HTML DOM, and can rely upon element and attribute names being returned in upper-case from those interfaces.
2. Applications that access XHTML documents served as Internet media types **`text/xml`** or **`application/xml`** can also use the XML DOM. Elements and attributes will be returned in lower-case. Also, some XHTML elements may or may not appear in the object tree because they are optional in the content model (e.g. the **`tbody`** element within **`table`**). This occurs because in HTML 4 some elements were permitted to be minimized such that their start and end tags are both omitted (an SGML feature). This is not possible in XML. Rather than require document authors to insert extraneous elements, XHTML has made the elements optional. Applications need to adapt to this accordingly.

C.12 Using Ampersands in Attribute Values

When an attribute value contains an ampersand, it must be expressed as a character entity reference (e.g. "**`&`**"). For example, when the **`href`** attribute of the **`a`** element refers to a CGI script that takes parameters, it must be expressed as **`http://my.site.dom/cgi-bin/myscript.pl?class=guest&name=user`** rather than as **`http://my.site.dom/cgi-bin/myscript.pl?class=guest&name=user`**.

C.13 Cascading Style Sheets (CSS) and XHTML

The Cascading Style Sheets level 2 Recommendation [CSS2] defines style properties which are applied to the parse tree of the HTML or XML document. Differences in parsing will produce different visual or aural results, depending on the selectors used. The following hints will reduce this effect for documents which are served without modification as both media types:

1. CSS style sheets for XHTML should use lower case element and attribute names.
2. In tables, the tbody element will be inferred by the parser of an HTML user agent, but not by the parser of an XML user agent. Therefore you

should always explicitly add a tbody element if it is referred to in a CSS selector.

3. Within the XHTML name space, user agents are expected to recognize the "id" attribute as an attribute of type ID. Therefore, style sheets should be able to continue using the shorthand "#" selector syntax even if the user agent does not read the DTD.
4. Within the XHTML name space, user agents are expected to recognize the "class" attribute. Therefore, style sheets should be able to continue using the shorthand "." selector syntax.
5. CSS defines different conformance rules for HTML and XML documents; be aware that the HTML rules apply to XHTML documents delivered as HTML and the XML rules apply to XHTML documents delivered as XML.

Appendix D. Acknowledgements

This appendix is informative.

This specification was written with the participation of the members of the W3C HTML working group:

Steven Pemberton, CWI (HTML Working Group Chair)
Murray Altheim, Sun Microsystems
Daniel Austin, AskJeeves (CNET: The Computer Network through July 1999)
Frank Boumphrey, HTML Writers Guild
John Burger, Mitre
Andrew W. Donoho, IBM
Sam Dooley, IBM
Klaus Hofrichter, GMD
Philipp Hoschka, W3C
Masayasu Ishikawa, W3C
Warner ten Kate, Philips Electronics
Peter King, Phone.com
Paula Klante, JetForm
Shin'ichi Matsui, Panasonic (W3C visiting engineer through September 1999)
Shane McCarron, Applied Testing and Technology (The Open Group through August 1999)
Ann Navarro, HTML Writers Guild
Zach Nies, Quark
Dave Raggett, W3C/HP (W3C lead for HTML)
Patrick Schmitz, Microsoft
Sebastian Schnitzenbaumer, Stack Overflow
Peter Stark, Phone.com

Chris Wilson, Microsoft
Ted Wugofski, Gateway 2000
Dan Zigmond, WebTV Networks

Appendix E. References

This appendix is informative.

[CSS2]
"Cascading Style Sheets, level 2 (CSS2) Specification", B. Bos, H. W. Lie, C. Lilley, I. Jacobs, 12 May 1998.
Latest version available at: http://www.w3.org/TR/REC-CSS2

[DOM]
"Document Object Model (DOM) Level 1 Specification", Lauren Wood *et al.*, 1 October 1998.
Latest version available at: http://www.w3.org/TR/REC-DOM-Level-1

[HTML]
"HTML 4.01 Specification", D. Raggett, A. Le Hors, I. Jacobs, 24 December 1999.
Latest version available at: http://www.w3.org/TR/html401

[POSIX.1]
"ISO/IEC 9945-1:1990 Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language]", Institute of Electrical and Electronics Engineers, Inc, 1990.

[RFC2046]
"RFC2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", N. Freed and N. Borenstein, November 1996.
Available at http://www.ietf.org/rfc/rfc2046.txt. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.

[RFC2119]
"RFC2119: Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997.
Available at: http://www.ietf.org/rfc/rfc2119.txt

[RFC2376]
"RFC2376: XML Media Types", E. Whitehead, M. Murata, July 1998.
Available at: http://www.ietf.org/rfc/rfc2376.txt

[RFC2396]
"RFC2396: Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, August 1998.
This document updates RFC1738 and RFC1808.
Available at: http://www.ietf.org/rfc/rfc2396.txt

[XML]

"Extensible Markup Language (XML) 1.0 Specification", T. Bray, J. Paoli, C. M. Sperberg-McQueen, 10 February 1998.
Latest version available at: http://www.w3.org/TR/REC-xml

[XMLNAMES]

"Namespaces in XML", T. Bray, D. Hollander, A. Layman, 14 January 1999.
XML namespaces provide a simple method for qualifying names used in XML documents by associating them with namespaces identified by URI.
Latest version available at: http://www.w3.org/TR/REC-xml-names

---

table of contents

Extensible Markup Language (XML) 1.0 (Second Edition)

W3C Recommendation 6 October 2000

This version:
http://www.w3.org/TR/2000/REC-xml-20001006 (XHTML, XML, PDF, XHTML review version with color-coded revision indicators)
Latest version:
http://www.w3.org/TR/REC-xml
Previous versions:
http://www.w3.org/TR/2000/WD-xml-2e-20000814
http://www.w3.org/TR/1998/REC-xml-19980210
Editors:
Tim Bray, Textuality and Netscape <tbray@textuality.com>
Jean Paoli, Microsoft <jeanpa@microsoft.com>
C. M. Sperberg-McQueen, University of Illinois at Chicago and Text Encoding Initiative <cmsmcq@uic.edu>
Eve Maler, Sun Microsystems, Inc. <eve.maler@east.sun.com> - Second Edition

---

Abstract

The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

Status of this Document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document specifies a syntax created by subsetting an existing, widely used international text processing standard (Standard Generalized Markup Language, ISO 8879:1986(E) as amended and corrected) for use on the World Wide Web. It is a product of the W3C XML Activity, details of which can be found at http://www.w3.org/XML. The English version of this specification is the only normative version. However, for translations of this document, see http://www.w3.org/XML/#trans. A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

This second edition is *not* a new version of XML (first published 10 February 1998); it merely incorporates the changes dictated by the first-edition errata (available at http://www.w3.org/XML/xml-19980210-errata) as a convenience to readers. The errata list for this second edition is available at http://www.w3.org/XML/xml-V10-2e-errata.

Please report errors in this document to xml-editor@w3.org; archives are available.

Note:

C. M. Sperberg-McQueen's affiliation has changed since the publication of the first edition. He is now at the World Wide Web Consortium, and can be contacted at cmsmcq@w3.org.

Table of Contents

Appendices

---

1 Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

[Definition: A software module called an **XML processor** is used to read XML documents and provide access to their content and structure.] [Definition: It is assumed that an XML processor is doing its work on behalf of another module, called the **application**.] This specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C. The membership of the XML Working Group is given in an appendix. Dan Connolly served as the WG's contact with the W3C.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

This specification, together with associated standards (Unicode and ISO/IEC 10646 for characters, Internet RFC 1766 for language identification tags, ISO 639 for language name codes, and ISO 3166 for country name codes), provides all the information necessary to understand XML Version 1.0 and construct computer programs to process it.

This version of the XML specification may be distributed freely, as long as all text and legal notices remain intact.

1.2 Terminology

The terminology used to describe XML documents is defined in the body of this specification. The terms defined in the following list are used in building those definitions and in describing the actions of an XML processor:

may

[Definition: Conforming documents and XML processors are permitted to but need not behave as described.]

must

[Definition: Conforming documents and XML processors are required to behave as described; otherwise they are in error.]

error

[Definition: A violation of the rules of this specification; results are undefined. Conforming software may detect and report an error and may recover from it.]

fatal error

[Definition: An error which a conforming XML processor must detect and report to the application. After encountering a fatal error, the processor may continue processing the data to search for further errors and may report such errors to the application. In order to support correction of errors, the processor may make unprocessed data from the document (with intermingled character data and markup) available to the application. Once a fatal error is detected, however, the processor must not continue normal processing (i.e., it must not continue to pass character data and information about the document's logical structure to the application in the normal way).]

at user option

[Definition: Conforming software may or must (depending on the modal verb in the sentence) behave as described; if it does, it must provide users a means to enable or disable the behavior described.]

validity constraint

[Definition: A rule which applies to all valid XML documents. Violations of validity constraints are errors; they must, at user option, be reported by validating XML processors.]

well-formedness constraint

[Definition: A rule which applies to all well-formed XML documents. Violations of well-formedness constraints are fatal errors.]

match

[Definition: (Of strings or names:) Two strings or names being compared must be identical. Characters with multiple possible representations in ISO/IEC 10646 (e.g. characters with both precomposed and base+diacritic forms) match only if they have the same representation in both strings. No case folding is performed. (Of strings and rules in the grammar:) A string matches a grammatical production if it belongs to the language generated by that production. (Of content and content models:) An element matches its declaration when it conforms in the fashion described in the constraint **[VC: Element Valid]**.]

for compatibility

[Definition: Marks a sentence describing a feature of XML included solely to ensure that XML remains compatible with SGML.]

for interoperability

[Definition: Marks a sentence describing a non-binding recommendation included to increase the chances that XML documents can be processed by the existing installed base of SGML processors which predate the WebSGML Adaptations Annex to ISO 8879.]

2 Documents

[Definition: A data object is an **XML document** if it is well-formed, as defined in this specification. A well-formed XML document may in addition be valid if it meets certain further constraints.]

Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The logical and physical structures must nest properly, as described in **4.3.2 Well-Formed Parsed Entities**.

2.1 Well-Formed XML Documents

[Definition: A textual object is a **well-formed** XML document if:]

1. Taken as a whole, it matches the production labeled document.
2. It meets all the well-formedness constraints given in this specification.
3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed.

Document

```
[1]  document  ::=  prolog element Misc*
```

Matching the document production implies that:

1. It contains one or more elements.
2. [Definition: There is exactly one element, called the **root**, or document element, no part of which appears in the content of any other element.] For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

[Definition: As a consequence of this, for each non-root element `C` in the document, there is one other element `P` in the document such that `C` is in the content of `P`, but is not in the content of any other element that is in the content of `P`. `P` is referred to as the **parent** of `C`, and `C` as a **child** of `P`.]

2.2 Characters

[Definition: A parsed entity contains **text**, a sequence of characters, which may represent markup or character data.] [Definition: A **character** is an atomic unit of text as specified by ISO/IEC 10646 [ISO/IEC 10646] (see also [ISO/IEC 10646-2000]). Legal characters are tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646. The versions of these standards cited in **A.1 Normative References** were current at the time this document was prepared. New characters may be added to these standards by amendments or new editions. Consequently, XML processors must accept any character in the range specified for Char. The use of "compatibility characters", as defined in section 6.8 of [Unicode] (see also D21 in section 3.6 of [Unicode3]), is discouraged.]

Character Range

```
[2]  Char  ::=  #x9 | #xA | #xD | [#x20-#xD7FF] |        /* any Unicode character, excluding the
                [#xE000-#xFFFD] | [#x10000-#x10FFFF]     surrogate blocks, FFFE, and FFFF. */
```

The mechanism for encoding character code points into bit patterns may vary from entity to entity. All XML processors must accept the UTF-8 and UTF-16 encodings of 10646; the mechanisms for signaling which of the two is in use, or for bringing other encodings into play, are discussed later, in **4.3.3 Character Encoding in Entities**.

2.3 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.

S (white space) consists of one or more space (#x20) characters, carriage returns, line feeds, or tabs.

White Space

```
[3]  S  ::=  (#x20 | #x9 | #xD | #xA)+
```

Characters are classified for convenience as letters, digits, or other characters. A letter consists of an alphabetic or syllabic base character or an ideographic character. Full definitions of the specific characters in each class are given in **B Character Classes**.

[Definition: A **Name** is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters.] Names beginning with the string "`xml`", or any string which would match `(('X'|'x') ('M'|'m') ('L'|'l'))`, are reserved for standardization in this or future versions of this specification.

> **Note:**
>
> The Namespaces in XML Recommendation [XML Names] assigns a meaning to names containing colon characters. Therefore, authors should not use the colon in XML names except for namespace purposes, but XML processors must accept the colon as a name character.

An Nmtoken (name token) is any mixture of name characters.

Names and Tokens

```
[4]  NameChar  ::=  Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
[5]  Name      ::=  (Letter | '_' | ':') (NameChar)*
[6]  Names     ::=  Name (S Name)*
[7]  Nmtoken   ::=  (NameChar)+
[8]  Nmtokens  ::=  Nmtoken (S Nmtoken)*
```

Literal data is any quoted string not containing the quotation mark used as a delimiter for that string. Literals are used for specifying the content of internal entities (EntityValue), the values of attributes (AttValue), and external identifiers (SystemLiteral). Note that a SystemLiteral can be parsed without scanning for markup.

Literals

```
[9]   EntityValue    ::=  '"' ([^%&"] | PEReference | Reference)* '"'
                          |  "'" ([^%&'] | PEReference | Reference)* "'"
[10]  AttValue       ::=  '"' ([^<&"] | Reference)* '"'
                          |  "'" ([^<&'] | Reference)* "'"
[11]  SystemLiteral  ::=  ('"' [^"]* '"') | ("'" [^']* "'")
[12]  PubidLiteral   ::=  '"' PubidChar* '"' | "'" (PubidChar - "'")* "'"
[13]  PubidChar      ::=  #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]
```

> **Note:**
>
> Although the EntityValue production allows the definition of an entity consisting of a single explicit < in the literal (e.g., `<!ENTITY mylt "<">`), it is strongly advised to avoid this practice since any reference to that entity will cause a well-formedness error.

2.4 Character Data and Markup

Text consists of intermingled character data and markup. [Definition: **Markup** takes the form of start-tags, end-tags, empty-element tags, entity references, character references, comments, CDATA section delimiters, document type declarations, processing instructions, XML declarations, text declarations, and any white space that is at the top level of the document entity (that is, outside the document element and not inside any other markup).]

[Definition: All text that is not markup constitutes the **character data** of the document.]

The ampersand character (&) and the left angle bracket (<) may appear in their literal form *only* when used as markup delimiters, or within a comment, a processing instruction, or a CDATA section. If they are needed elsewhere, they must be escaped using either numeric character references or the strings "`&`" and "`<`" respectively. The right angle bracket (>) may be represented using the string "`>`", and must, for compatibility, be escaped using "`>`" or a character reference when it appears in the string "`]]>`" in content, when that string is not marking the end of a CDATA section.

In the content of elements, character data is any string of characters which does not contain the start-delimiter of any markup. In a CDATA section, character data is any string of characters not including the CDATA-section-close delimiter, "`]]>`".

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') may be represented as "`&apos;`", and the double-quote character (") as "`&quot;`".

Character Data

```
[14]  CharData  ::=  [^<&]* - ([^<&]* ']]>' [^<&]*)
```

2.5 Comments

[Definition: **Comments** may appear anywhere in a document outside other markup; in addition, they may appear within the document type declaration at places allowed by the grammar. They are not part of the document's character data; an XML processor may, but need not, make it possible for an application to retrieve the text of comments. For compatibility, the string "`--`" (double-hyphen) must not occur within comments.] Parameter entity references are not recognized within comments.

Comments

```
[15]  Comment  ::=  '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

An example of a comment:

```
<!-- declarations for <head> & <body> -->
```

Note that the grammar does not allow a comment ending in `--->`. The following example is *not* well-formed.

```
<!-- B+, B, or B--->
```

2.6 Processing Instructions

[Definition: **Processing instructions** (PIs) allow documents to contain instructions for applications.]

Processing Instructions

```
[16]  PI        ::=  '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17]  PITarget  ::=  Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

PIs are not part of the document's character data, but must be passed through to the application. The PI begins with a target (PITarget) used to identify the application to which the instruction is directed. The target names "`XML`", "`xml`", and so on are reserved for standardization in this or future versions of this specification. The XML Notation mechanism may be used for formal declaration of PI targets. Parameter entity references are not recognized within processing instructions.

2.7 CDATA Sections

[Definition: **CDATA sections** may occur anywhere character data may occur; they are used to escape blocks of text

containing characters which would otherwise be recognized as markup. CDATA sections begin with the string "<![CDATA[" and end with the string "]]>":]

CDATA Sections

```
[18]  CDSect   ::=  CDStart CData CDEnd
[19]  CDStart  ::=  '<![CDATA['
[20]  CData    ::=  (Char* - (Char* ']]>'
                    Char*))
[21]  CDEnd    ::=  ']]>'
```

Within a CDATA section, only the CDEnd string is recognized as markup, so that left angle brackets and ampersands may occur in their literal form; they need not (and cannot) be escaped using "<" and "&". CDATA sections cannot nest.

An example of a CDATA section, in which "<greeting>" and "</greeting>" are recognized as character data, not markup:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

2.8 Prolog and Document Type Declaration

[Definition: XML documents should begin with an **XML declaration** which specifies the version of XML being used.] For example, the following is a complete XML document, well-formed but not valid:

```
<?xml version="1.0"?> <greeting>Hello, world!</greeting>
```

and so is this:

```
<greeting>Hello, world!</greeting>
```

The version number "1.0" should be used to indicate conformance to this version of this specification; it is an error for a document to use the value "1.0" if it does not conform to this version of this specification. It is the intent of the XML working group to give later versions of this specification numbers other than "1.0", but this intent does not indicate a commitment to produce any future versions of XML, nor if any are produced, to use any particular numbering scheme. Since future versions are not ruled out, this construct is provided as a means to allow the possibility of automatic version recognition, should it become necessary. Processors may signal an error if they receive documents labeled with versions they do not support.

The function of the markup in an XML document is to describe its storage and logical structure and to associate attribute-value pairs with its logical structures. XML provides a mechanism, the document type declaration, to define constraints on the logical structure and to support the use of predefined storage units. [Definition: An XML document is **valid** if it has an associated document type declaration and if the document complies with the constraints expressed in it.]

The document type declaration must appear before the first element in the document.

Prolog

```
[22]  prolog       ::=  XMLDecl? Misc* (doctypedecl Misc*)?
[23]  XMLDecl      ::=  '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24]  VersionInfo  ::=  S 'version' Eq ("'" VersionNum "'" | '"' VersionNum '"')/* */
[25]  Eq           ::=  S? '=' S?
[26]  VersionNum   ::=  ([a-zA-Z0-9_.:] | '-')+
```

```
[27]  Misc          ::=  Comment | PI | S
```

[Definition: The XML **document type declaration** contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or **DTD**. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. The DTD for a document consists of both subsets taken together.]

[Definition: A **markup declaration** is an element type declaration, an attribute-list declaration, an entity declaration, or a notation declaration.] These declarations may be contained in whole or in part within parameter entities, as described in the well-formedness and validity constraints below. For further information, see **4 Physical Structures**.

Document Type Definition

```
[28]   doctypedecl  ::=  '<!DOCTYPE' S Name (S ExternalID)? S?
                         ('[' (markupdecl | DeclSep)* ']' S?)? '>'
                                                    [VC: Root Element Type]
                                                    [WFC: External Subset]
                                                    /* */
[28a]  DeclSep      ::=  PEReference | S            [WFC: PE Between
                                                    Declarations]
                                                    /* */
[29]   markupdecl   ::=  elementdecl | AttlistDecl | EntityDecl |
                         NotationDecl | PI | Comment
                                                    [VC: Proper Declaration/PE
                                                    Nesting]
                                                    [WFC: PEs in Internal Subset]
```

Note that it is possible to construct a well-formed document containing a doctypedecl that neither points to an external subset nor contains an internal subset.

The markup declarations may be made up in whole or in part of the replacement text of parameter entities. The productions later in this specification for individual nonterminals (elementdecl, AttlistDecl, and so on) describe the declarations *after* all the parameter entities have been included.

Parameter entity references are recognized anywhere in the DTD (internal and external subsets and external parameter entities), except in literals, processing instructions, comments, and the contents of ignored conditional sections (see **3.4 Conditional Sections**). They are also recognized in entity value literals. The use of parameter entities in the internal subset is restricted as described below.

Validity constraint: Root Element Type

The Name in the document type declaration must match the element type of the root element.

Validity constraint: Proper Declaration/PE Nesting

Parameter-entity replacement text must be properly nested with markup declarations. That is to say, if either the first character or the last character of a markup declaration (markupdecl above) is contained in the replacement text for a parameter-entity reference, both must be contained in the same replacement text.

Well-formedness constraint: PEs in Internal Subset

In the internal DTD subset, parameter-entity references can occur only where markup declarations can occur, not within markup declarations. (This does not apply to references that occur in external parameter entities or to the external subset.)

Well-formedness constraint: External Subset

The external subset, if any, must match the production for extSubset.

Well-formedness constraint: PE Between Declarations

The replacement text of a parameter entity reference in a DeclSep must match the production extSubsetDecl.

Like the internal subset, the external subset and any external parameter entities referenced in a DeclSep must consist of a series of complete markup declarations of the types allowed by the non-terminal symbol markupdecl, interspersed with white space or parameter-entity references. However, portions of the contents of the external subset or of these external parameter entities may conditionally be ignored by using the conditional section construct; this is not allowed in the internal subset.

External Subset

```
[30]  extSubset      ::=  TextDecl? extSubsetDecl
[31]  extSubsetDecl  ::=  ( markupdecl | conditionalSect | DeclSep)* /* */
```

The external subset and external parameter entities also differ from the internal subset in that in them, parameter-entity references are permitted *within* markup declarations, not only *between* markup declarations.

An example of an XML document with a document type declaration:

```
<?xml version="1.0"?> <!DOCTYPE greeting SYSTEM "hello.dtd"> <greeting>Hello, world!
</greeting>
```

The system identifier "`hello.dtd`" gives the address (a URI reference) of a DTD for the document.

The declarations can also be given locally, as in this example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

If both the external and internal subsets are used, the internal subset is considered to occur before the external subset. This has the effect that entity and attribute-list declarations in the internal subset take precedence over those in the external subset.

2.9 Standalone Document Declaration

Markup declarations can affect the content of the document, as passed from an XML processor to an application; examples are attribute defaults and entity declarations. The standalone document declaration, which may appear as a component of the XML declaration, signals whether or not there are such declarations which appear external to the document entity or in parameter entities. [Definition: An **external markup declaration** is defined as a markup declaration occurring in the external subset or in a parameter entity (external or internal, the latter being included because non-validating processors are not required to read them).]

Standalone Document Declaration

```
[32]  SDDecl  ::=  S 'standalone' Eq (("'" ('yes' | 'no')     [VC: Standalone Document
                   "'") | ('"' ('yes' | 'no') '"'))          Declaration]
```

In a standalone document declaration, the value "yes" indicates that there are no external markup declarations which affect the information passed from the XML processor to the application. The value "no" indicates that there are or may be such external markup declarations. Note that the standalone document declaration only denotes the presence of external *declarations*; the presence, in a document, of references to external *entities*, when those entities are internally declared, does not change its standalone status.

If there are no external markup declarations, the standalone document declaration has no meaning. If there are external markup declarations but there is no standalone document declaration, the value "no" is assumed.

Any XML document for which `standalone="no"` holds can be converted algorithmically to a standalone document, which may be desirable for some network delivery applications.

Validity constraint: Standalone Document Declaration

The standalone document declaration must have the value "no" if any external markup declarations contain declarations of:

- attributes with default values, if elements to which these attributes apply appear in the document without specifications of values for these attributes, or
- entities (other than `amp`, `lt`, `gt`, `apos`, `quot`), if references to those entities appear in the document, or
- attributes with values subject to *normalization*, where the attribute appears in the document with a value which will change as a result of normalization, or
- element types with element content, if white space occurs directly within any instance of those types.

An example XML declaration with a standalone document declaration:

```
<?xml version="1.0" standalone='yes'?>
```

2.10 White Space Handling

In editing XML documents, it is often convenient to use "white space" (spaces, tabs, and blank lines) to set apart the markup for greater readability. Such white space is typically not intended for inclusion in the delivered version of the document. On the other hand, "significant" white space that should be preserved in the delivered version is common, for example in poetry and source code.

An XML processor must always pass all characters in a document that are not markup through to the application. A validating XML processor must also inform the application which of these characters constitute white space appearing in element content.

A special attribute named `xml:space` may be attached to an element to signal an intention that in that element, white space should be preserved by applications. In valid documents, this attribute, like any other, must be declared if it is used. When declared, it must be given as an enumerated type whose values are one or both of "default" and "preserve". For example:

```
<!ATTLIST poem  xml:space (default|preserve) 'preserve'>

<!-- -->
<!ATTLIST pre xml:space (preserve) #FIXED 'preserve'>
```

The value "default" signals that applications' default white-space processing modes are acceptable for this element; the value "preserve" indicates the intent that applications preserve all the white space. This declared intent is considered to apply to all elements within the content of the element where it is specified, unless overriden with another instance of the `xml:space` attribute.

The root element of any document is considered to have signaled no intentions as regards application space handling, unless it provides a value for this attribute or the attribute is declared with a default value.

2.11 End-of-Line Handling

XML parsed entities are often stored in computer files which, for editing convenience, are organized into lines. These lines are typically separated by some combination of the characters carriage-return (#xD) and line-feed (#xA).

To simplify the tasks of applications, the characters passed to an application by the XML processor must be as if the XML processor normalized all line breaks in external parsed entities (including the document entity) on input, before parsing, by translating both the two-character sequence #xD #xA and any #xD that is not followed by #xA to a single #xA character.

2.12 Language Identification

In document processing, it is often useful to identify the natural or formal language in which the content is written. A special attribute named `xml:lang` may be inserted in documents to specify the language used in the contents and attribute values of any element in an XML document. In valid documents, this attribute, like any other, must be declared if it is used. The values of the attribute are language identifiers as defined by [IETF RFC 1766], *Tags for the Identification of Languages*, or its successor on the IETF Standards Track.

> **Note:**
>
> [IETF RFC 1766] tags are constructed from two-letter language codes as defined by [ISO 639], from two-letter country codes as defined by [ISO 3166], or from language identifiers registered with the Internet Assigned Numbers Authority [IANA-LANGCODES]. It is expected that the successor to [IETF RFC 1766] will introduce three-letter language codes for languages not presently covered by [ISO 639].

(Productions 33 through 38 have been removed.)

For example:

```
<p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
  <l>Habe nun, ach! Philosophie,</l>
  <l>Juristerei, und Medizin</l>
  <l>und leider auch Theologie</l>
  <l>durchaus studiert mit heißem Bemüh'n.</l>
</sp>
```

The intent declared with `xml:lang` is considered to apply to all attributes and content of the element where it is specified, unless overridden with an instance of `xml:lang` on another element within that content.

A simple declaration for `xml:lang` might take the form

```
xml:lang NMTOKEN #IMPLIED
```

but specific default values may also be given, if appropriate. In a collection of French poems for English students, with glosses and notes in English, the `xml:lang` attribute might be declared this way:

```
<!ATTLIST poem   xml:lang NMTOKEN 'fr'>
<!ATTLIST gloss  xml:lang NMTOKEN 'en'>
<!ATTLIST note   xml:lang NMTOKEN 'en'>
```

3 Logical Structures

[Definition: Each XML document contains one or more **elements**, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, sometimes called its "generic identifier" (GI), and may have a set of attribute specifications.] Each attribute specification has a name and a value.

Element

```
[39]  element  ::=  EmptyElemTag
                  | STag content ETag [WFC: Element Type Match]
                                      [VC: Element Valid]
```

This specification does not constrain the semantics, use, or (beyond syntax) names of the element types and attributes, except that names beginning with a match to `(('X'|'x')('M'|'m')('L'|'l'))` are reserved for standardization in this or future versions of this specification.

Well-formedness constraint: Element Type Match

The Name in an element's end-tag must match the element type in the start-tag.

Validity constraint: Element Valid

An element is valid if there is a declaration matching elementdecl where the Name matches the element type, and one of the following holds:

1. The declaration matches **EMPTY** and the element has no content.
2. The declaration matches children and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space (characters matching the nonterminal S) between the start-tag and the first child element, between child elements, or between the last child element and the end-tag. Note that a CDATA section containing only white space does not match the nonterminal S, and hence cannot appear in these positions.
3. The declaration matches Mixed and the content consists of character data and child elements whose types match names in the content model.
4. The declaration matches **ANY**, and the types of any child elements have been declared.

3.1 Start-Tags, End-Tags, and Empty-Element Tags

[Definition: The beginning of every non-empty XML element is marked by a **start-tag**.]

Start-tag

```
[40]  STag       ::=  '<' Name (S Attribute)* S? '>'  [WFC: Unique Att Spec]
[41]  Attribute  ::=  Name Eq AttValue                [VC: Attribute Value Type]
                                                      [WFC: No External Entity References]
                                                      [WFC: No < in Attribute Values]
```

The Name in the start- and end-tags gives the element's **type**. [Definition: The Name-AttValue pairs are referred to as the **attribute specifications** of the element], [Definition: with the Name in each pair referred to as the **attribute name**] and [Definition: the content of the AttValue (the text between the ' or " delimiters) as the **attribute value**.]Note that the order of attribute specifications in a start-tag or empty-element tag is not significant.

Well-formedness constraint: Unique Att Spec

No attribute name may appear more than once in the same start-tag or empty-element tag.

Validity constraint: Attribute Value Type

The attribute must have been declared; the value must be of the type declared for it. (For attribute types, see **3.3 Attribute-List Declarations**.)

Well-formedness constraint: No External Entity References

Attribute values cannot contain direct or indirect entity references to external entities.

Well-formedness constraint: No < in Attribute Values

The replacement text of any entity referred to directly or indirectly in an attribute value must not contain a <.

An example of a start-tag:

```
    <termdef id="dt-dog" term="dog">
```

[Definition: The end of every element that begins with a start-tag must be marked by an **end-tag** containing a name that echoes the element's type as given in the start-tag:]

End-tag

[42] ETag ::= '</' Name S? '>'

An example of an end-tag:

```
    </termdef>
```

[Definition: The text between the start-tag and end-tag is called the element's **content**:]

Content of Elements

[43] content ::= CharData? ((element | Reference | CDSect | PI | Comment) CharData?) /* */
*

[Definition: An element with no content is said to be **empty**.] The representation of an empty element is either a start-tag immediately followed by an end-tag, or an empty-element tag. [Definition: An **empty-element tag** takes a special form:]

Tags for Empty Elements

[44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>' [WFC: Unique Att Spec]

Empty-element tags may be used for any element which has no content, whether or not it is declared using the keyword **EMPTY**. For interoperability, the empty-element tag should be used, and should only be used, for elements which are declared EMPTY.

Examples of empty elements:

```
    <IMG align="left"
     src="http://www.w3.org/Icons/WWW/w3c_home" />
    <br></br>
    <br/>
```

3.2 Element Type Declarations

The element structure of an XML document may, for validation purposes, be constrained using element type and attribute-list declarations. An element type declaration constrains the element's content.

Element type declarations often constrain which element types can appear as children of the element. At user option, an XML processor may issue a warning when a declaration mentions an element type for which no declaration is provided, but this is not an error.

[Definition: An **element type declaration** takes the form:]

Element Type Declaration

[45] elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>' [VC: Unique Element Type Declaration]

```
[46]  contentspec  ::=  'EMPTY' | 'ANY' | Mixed | children
```

where the Name gives the element type being declared.

Validity constraint: Unique Element Type Declaration

No element type may be declared more than once.

Examples of element type declarations:

```
<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA|emph)* >
<!ELEMENT %name.para; %content.para; >
<!ELEMENT container ANY>
```

3.2.1 Element Content

[Definition: An element type has **element content** when elements of that type must contain only child elements (no character data), optionally separated by white space (characters matching the nonterminal S).][Definition: In this case, the constraint includes a **content model**, a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear.] The grammar is built on content particles (cps), which consist of names, choice lists of content particles, or sequence lists of content particles:

Element-content Models

```
[47]  children  ::=  (choice | seq) ('?' | '*' | '+')?
[48]  cp        ::=  (Name | choice | seq) ('?' | '*' | '+')?
[49]  choice    ::=  '(' S? cp ( S? '|' S? cp )+ S? ')'      /* */
                                                             /* */
                                                             [VC: Proper Group/PE Nesting]
[50]  seq       ::=  '(' S? cp ( S? ',' S? cp )* S? ')'      /* */
                                                             [VC: Proper Group/PE Nesting]
```

where each Name is the type of an element which may appear as a child. Any content particle in a choice list may appear in the element content at the location where the choice list appears in the grammar; content particles occurring in a sequence list must each appear in the element content in the order given in the list. The optional character following a name or list governs whether the element or the content particles in the list may occur one or more (+), zero or more (*), or zero or one times (?). The absence of such an operator means that the element or content particle must appear exactly once. This syntax and meaning are identical to those used in the productions in this specification.

The content of an element matches a content model if and only if it is possible to trace out a path through the content model, obeying the sequence, choice, and repetition operators and matching each element in the content against an element type in the content model. For compatibility, it is an error if an element in the document can match more than one occurrence of an element type in the content model. For more information, see **E Deterministic Content Models**.

Validity constraint: Proper Group/PE Nesting

Parameter-entity replacement text must be properly nested with parenthesized groups. That is to say, if either of the opening or closing parentheses in a choice, seq, or Mixed construct is contained in the replacement text for a parameter entity, both must be contained in the same replacement text.

For interoperability, if a parameter-entity reference appears in a choice, seq, or Mixed construct, its replacement text should contain at least one non-blank character, and neither the first nor last non-blank character of the replacement text should be a connector (`|` or `,`).

Examples of element-content models:

```
    <!ELEMENT spec (front, body, back?)>
    <!ELEMENT div1 (head, (p | list | note)*, div2*)>
    <!ELEMENT dictionary-body (%div.mix; | %dict.mix;)*>
```

3.2.2 Mixed Content

[Definition: An element type has **mixed content** when elements of that type may contain character data, optionally interspersed with child elements.] In this case, the types of the child elements may be constrained, but not their order or their number of occurrences:

Mixed-content Declaration

```
[51]  Mixed  ::=  '(' S? '#PCDATA' (S? '|' S? Name)* S? ')
                  *'
                  | '(' S? '#PCDATA' S? ')'                [VC: Proper Group/PE Nesting]
                                                           [VC: No Duplicate Types]
```

where the Names give the types of elements that may appear as children. The keyword **#PCDATA** derives historically from the term "parsed character data."

Validity constraint: No Duplicate Types

The same name must not appear more than once in a single mixed-content declaration.

Examples of mixed content declarations:

```
    <!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
    <!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >
    <!ELEMENT b (#PCDATA)>
```

3.3 Attribute-List Declarations

Attributes are used to associate name-value pairs with elements. Attribute specifications may appear only within start-tags and empty-element tags; thus, the productions used to recognize them appear in **3.1 Start-Tags, End-Tags, and Empty-Element Tags**. Attribute-list declarations may be used:

- To define the set of attributes pertaining to a given element type.
- To establish type constraints for these attributes.
- To provide default values for attributes.

[Definition: **Attribute-list declarations** specify the name, data type, and default value (if any) of each attribute associated with a given element type:]

Attribute-list Declaration

```
[52]  AttlistDecl  ::=  '<!ATTLIST' S Name AttDef* S? '>'
[53]  AttDef       ::=  S Name S AttType S DefaultDecl
```

The Name in the AttlistDecl rule is the type of an element. At user option, an XML processor may issue a warning if attributes are declared for an element type not itself declared, but this is not an error. The Name in the AttDef rule is the name of the attribute.

When more than one AttlistDecl is provided for a given element type, the contents of all those provided are merged. When more than one definition is provided for the same attribute of a given element type, the first declaration is binding and later declarations are ignored. For interoperability, writers of DTDs may choose to provide at most one attribute-list declaration for a given element type, at most one attribute definition for a given attribute name in an attribute-list

declaration, and at least one attribute definition in each attribute-list declaration. For interoperability, an XML processor may at user option issue a warning when more than one attribute-list declaration is provided for a given element type, or more than one attribute definition is provided for a given attribute, but this is not an error.

3.3.1 Attribute Types

XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The string type may take any literal string as a value; the tokenized types have varying lexical and semantic constraints. The validity constraints noted in the grammar are applied after the attribute value has been normalized as described in **3.3 Attribute-List Declarations**.

Attribute Types

[54]	AttType	::=	StringType | TokenizedType | EnumeratedType	
[55]	StringType	::=	'CDATA'	
[56]	TokenizedType	::=	'ID'	[VC: ID] [VC: One ID per Element Type] [VC: ID Attribute Default]
			| 'IDREF'	[VC: IDREF]
			| 'IDREFS'	[VC: IDREF]
			| 'ENTITY'	[VC: Entity Name]
			| 'ENTITIES'	[VC: Entity Name]
			| 'NMTOKEN'	[VC: Name Token]
			| 'NMTOKENS'	[VC: Name Token]

Validity constraint: ID

Values of type **ID** must match the Name production. A name must not appear more than once in an XML document as a value of this type; i.e., ID values must uniquely identify the elements which bear them.

Validity constraint: One ID per Element Type

No element type may have more than one ID attribute specified.

Validity constraint: ID Attribute Default

An ID attribute must have a declared default of **#IMPLIED** or **#REQUIRED**.

Validity constraint: IDREF

Values of type **IDREF** must match the Name production, and values of type **IDREFS** must match Names; each Name must match the value of an ID attribute on some element in the XML document; i.e. **IDREF** values must match the value of some ID attribute.

Validity constraint: Entity Name

Values of type **ENTITY** must match the Name production, values of type **ENTITIES** must match Names; each Name must match the name of an unparsed entity declared in the DTD.

Validity constraint: Name Token

Values of type **NMTOKEN** must match the Nmtoken production; values of type **NMTOKENS** must match Nmtokens.

[Definition: **Enumerated attributes** can take one of a list of values provided in the declaration]. There are two kinds of enumerated types:

Enumerated Attribute Types

```
[57]  EnumeratedType  ::=  NotationType | Enumeration
[58]  NotationType    ::=  'NOTATION' S '(' S? Name (S? '|'      [VC: Notation Attributes]
                           S? Name)* S? ')'
                                                                 [VC: One Notation Per Element
                                                                 Type]
                                                                 [VC: No Notation on Empty Element]
[59]  Enumeration     ::=  '(' S? Nmtoken (S? '|' S? Nmtoken)    [VC: Enumeration]
                           * S? ')'
```

A **NOTATION** attribute identifies a notation, declared in the DTD with associated system and/or public identifiers, to be used in interpreting the element to which the attribute is attached.

Validity constraint: Notation Attributes

Values of this type must match one of the *notation* names included in the declaration; all notation names in the declaration must be declared.

Validity constraint: One Notation Per Element Type

No element type may have more than one **NOTATION** attribute specified.

Validity constraint: No Notation on Empty Element

For compatibility, an attribute of type **NOTATION** must not be declared on an element declared **EMPTY**.

Validity constraint: Enumeration

Values of this type must match one of the Nmtoken tokens in the declaration.

For interoperability, the same Nmtoken should not occur more than once in the enumerated attribute types of a single element type.

3.3.2 Attribute Defaults

An attribute declaration provides information on whether the attribute's presence is required, and if not, how an XML processor should react if a declared attribute is absent in a document.

Attribute Defaults

```
[60]  DefaultDecl  ::=  '#REQUIRED' | '#IMPLIED'
                        | (('#FIXED' S)? AttValue)  [VC: Required Attribute]
                                                    [VC: Attribute Default Legal]
                                                    [WFC: No < in Attribute Values]
                                                    [VC: Fixed Attribute Default]
```

In an attribute declaration, **#REQUIRED** means that the attribute must always be provided, **#IMPLIED** that no default value is provided. [Definition: If the declaration is neither **#REQUIRED** nor **#IMPLIED**, then the AttValue value contains the declared **default** value; the **#FIXED** keyword states that the attribute must always have the default value. If a default value is declared, when an XML processor encounters an omitted attribute, it is to behave as though the attribute were present with the declared default value.]

Validity constraint: Required Attribute

If the default declaration is the keyword **#REQUIRED**, then the attribute must be specified for all elements of the type in the attribute-list declaration.

Validity constraint: Attribute Default Legal

The declared default value must meet the lexical constraints of the declared attribute type.

Validity constraint: Fixed Attribute Default

If an attribute has a default value declared with the **#FIXED** keyword, instances of that attribute must match the default value.

Examples of attribute-list declarations:

```
<!ATTLIST termdef
          id      ID      #REQUIRED
          name    CDATA   #IMPLIED>
<!ATTLIST list
          type    (bullets|ordered|glossary)  "ordered">
<!ATTLIST form
          method  CDATA   #FIXED "POST">
```

3.3.3 Attribute-Value Normalization

Before the value of an attribute is passed to the application or checked for validity, the XML processor must normalize the attribute value by applying the algorithm below, or by using some other method such that the value passed to the application is the same as that produced by the algorithm.

1. All line breaks must have been normalized on input to #xA as described in **2.11 End-of-Line Handling**, so the rest of this algorithm operates on text normalized in this way.
2. Begin with a normalized value consisting of the empty string.
3. For each character, entity reference, or character reference in the unnormalized attribute value, beginning with the first and continuing to the last, do the following:
   - For a character reference, append the referenced character to the normalized value.
   - For an entity reference, recursively apply step 3 of this algorithm to the replacement text of the entity.
   - For a white space character (#x20, #xD, #xA, #x9), append a space character (#x20) to the normalized value.
   - For another character, append the character to the normalized value.

If the attribute type is not CDATA, then the XML processor must further process the normalized attribute value by discarding any leading and trailing space (#x20) characters, and by replacing sequences of space (#x20) characters by a single space (#x20) character.

Note that if the unnormalized attribute value contains a character reference to a white space character other than space (#x20), the normalized value contains the referenced character itself (#xD, #xA or #x9). This contrasts with the case where the unnormalized value contains a white space character (not a reference), which is replaced with a space character (#x20) in the normalized value and also contrasts with the case where the unnormalized value contains an entity reference whose replacement text contains a white space character; being recursively processed, the white space character is replaced with a space character (#x20) in the normalized value.

All attributes for which no declaration has been read should be treated by a non-validating processor as if declared **CDATA**.

Following are examples of attribute normalization. Given the following declarations:

```
<!ENTITY d "&#xD;">
<!ENTITY a "&#xA;">
<!ENTITY da "&#xD;&#xA;">
```

the attribute specifications in the left column below would be normalized to the character sequences of the middle column if the attribute a is declared **NMTOKENS** and to those of the right columns if a is declared **CDATA**.

Attribute specification	a is NMTOKENS	a is CDATA

a=" xyz"	x y z	#x20 #x20 x y z
a="&d;&d;A&a;&a;B&da;"	A #x20 B	#x20 #x20 A #x20 #x20 B #x20 #x20
a="

A

B
" | #xD #xD A #xA #xA B #xD #xA | #xD #xD A #xA #xA B #xD #xD |

Note that the last example is invalid (but well-formed) if a is declared to be of type **NMTOKENS**.

3.4 Conditional Sections

[Definition: **Conditional sections** are portions of the document type declaration external subset which are included in, or excluded from, the logical structure of the DTD based on the keyword which governs them.]

Conditional Section

```
[61]  conditionalSect      ::=  includeSect | ignoreSect
[62]  includeSect          ::=  '<![' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'   /* */
                                                                  [VC: Proper Conditional Section/PE Nesting]
[63]  ignoreSect           ::=  '<![' S? 'IGNORE' S? '[' ignoreSectContents* ']]>'   /* */
                                                                  [VC: Proper Conditional Section/PE Nesting]
[64]  ignoreSectContents   ::=  Ignore ('<![' ignoreSectContents ']]>' Ignore)*
[65]  Ignore               ::=  Char* - (Char* ('<![' | ']]>') Char*)
```

Validity constraint: Proper Conditional Section/PE Nesting

If any of the "<![", "[", or "]]>" of a conditional section is contained in the replacement text for a parameter-entity reference, all of them must be contained in the same replacement text.

Like the internal and external DTD subsets, a conditional section may contain one or more complete declarations, comments, processing instructions, or nested conditional sections, intermingled with white space.

If the keyword of the conditional section is **INCLUDE**, then the contents of the conditional section are part of the DTD. If the keyword of the conditional section is **IGNORE**, then the contents of the conditional section are not logically part of the DTD. If a conditional section with a keyword of **INCLUDE** occurs within a larger conditional section with a keyword of **IGNORE**, both the outer and the inner conditional sections are ignored. The contents of an ignored conditional section are parsed by ignoring all characters after the "[" following the keyword, except conditional section starts "<![" and ends "]]>", until the matching conditional section end is found. Parameter entity references are not recognized in this process.

If the keyword of the conditional section is a parameter-entity reference, the parameter entity must be replaced by its content before the processor decides whether to include or ignore the conditional section.

An example:

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >

<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

4 Physical Structures

[Definition: An XML document may consist of one or many storage units. These are called **entities**; they all have **content** and are all (except for the document entity and the external DTD subset) identified by entity **name**.] Each XML document has one entity called the document entity, which serves as the starting point for the XML processor and may contain the whole document.

Entities may be either parsed or unparsed. [Definition: A **parsed entity's** contents are referred to as its replacement text; this text is considered an integral part of the document.]

[Definition: An **unparsed entity** is a resource whose contents may or may not be text, and if text, may be other than XML. Each unparsed entity has an associated notation, identified by name. Beyond a requirement that an XML processor make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.]

Parsed entities are invoked by name using entity references; unparsed entities by name, given in the value of **ENTITY** or **ENTITIES** attributes.

[Definition: **General entities** are entities for use within the document content. In this specification, general entities are sometimes referred to with the unqualified term *entity* when this leads to no ambiguity.] [Definition: **Parameter entities** are parsed entities for use within the DTD.] These two types of entities use different forms of reference and are recognized in different contexts. Furthermore, they occupy different namespaces; a parameter entity and a general entity with the same name are two distinct entities.

4.1 Character and Entity References

[Definition: A **character reference** refers to a specific character in the ISO/IEC 10646 character set, for example one not directly accessible from available input devices.]

Character Reference

```
[66]  CharRef  ::=  '&#' [0-9]+ ';'
                  | '&#x' [0-9a-fA-F]+ ';' [WFC: Legal Character]
```

Well-formedness constraint: Legal Character

Characters referred to using character references must match the production for Char.

If the character reference begins with "&#x", the digits and letters up to the terminating `;` provide a hexadecimal representation of the character's code point in ISO/IEC 10646. If it begins just with "&#", the digits up to the terminating `;` provide a decimal representation of the character's code point.

[Definition: An **entity reference** refers to the content of a named entity.] [Definition: References to parsed general entities use ampersand (&) and semicolon (`;`) as delimiters.] [Definition: **Parameter-entity references** use percent-sign (%) and semicolon (`;`) as delimiters.]

Entity Reference

```
[67]  Reference  ::=  EntityRef | CharRef
```

[68]	EntityRef	::= '&' Name ';'	[WFC: Entity Declared] [VC: Entity Declared] [WFC: Parsed Entity] [WFC: No Recursion]
[69]	PEReference	::= '%' Name ';'	[VC: Entity Declared] [WFC: No Recursion] [WFC: In DTD]

Well-formedness constraint: Entity Declared

In a document without any DTD, a document with only an internal DTD subset which contains no parameter entity references, or a document with "`standalone='yes'`", for an entity reference that does not occur within the external subset or a parameter entity, the Name given in the entity reference must match that in an *entity declaration* that does not occur within the external subset or a parameter entity, except that well-formed documents need not declare any of the following entities: `amp`, `lt`, `gt`, `apos`, `quot`. The declaration of a general entity must precede any reference to it which appears in a default value in an attribute-list declaration.

Note that if entities are declared in the external subset or in external parameter entities, a non-validating processor is *not obligated to* read and process their declarations; for such documents, the rule that an entity must be declared is a well-formedness constraint only if *standalone='yes'*.

Validity constraint: Entity Declared

In a document with an external subset or external parameter entities with "`standalone='no'`", the Name given in the entity reference must match that in an *entity declaration*. For interoperability, valid documents should declare the entities `amp`, `lt`, `gt`, `apos`, `quot`, in the form specified in **4.6 Predefined Entities**. The declaration of a parameter entity must precede any reference to it. Similarly, the declaration of a general entity must precede any attribute-list declaration containing a default value with a direct or indirect reference to that general entity.

Well-formedness constraint: Parsed Entity

An entity reference must not contain the name of an unparsed entity. Unparsed entities may be referred to only in attribute values declared to be of type **ENTITY** or **ENTITIES**.

Well-formedness constraint: No Recursion

A parsed entity must not contain a recursive reference to itself, either directly or indirectly.

Well-formedness constraint: In DTD

Parameter-entity references may only appear in the DTD.

Examples of character and entity references:

```
Type <key>less-than</key> (&#x3C;) to save options.
This document was prepared on &docdate; and
is classified &security-level;.
```

Example of a parameter-entity reference:

```
<!-- declare the parameter entity "ISOLat2"... -->
<!ENTITY % ISOLat2
         SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ... now reference it. -->
%ISOLat2;
```

4.2 Entity Declarations

[Definition: Entities are declared thus:]

Entity Declaration

```
[70]  EntityDecl  ::=  GEDecl | PEDecl
[71]  GEDecl      ::=  '<!ENTITY' S Name S EntityDef S? '>'
[72]  PEDecl      ::=  '<!ENTITY' S '%' S Name S PEDef S? '>'
[73]  EntityDef   ::=  EntityValue | (ExternalID NDataDecl?)
[74]  PEDef       ::=  EntityValue | ExternalID
```

The Name identifies the entity in an entity reference or, in the case of an unparsed entity, in the value of an **ENTITY** or **ENTITIES** attribute. If the same entity is declared more than once, the first declaration encountered is binding; at user option, an XML processor may issue a warning if entities are declared multiple times.

4.2.1 Internal Entities

[Definition: If the entity definition is an EntityValue, the defined entity is called an **internal entity**. There is no separate physical storage object, and the content of the entity is given in the declaration.] Note that some processing of entity and character references in the literal entity value may be required to produce the correct replacement text: see **4.5 Construction of Internal Entity Replacement Text**.

An internal entity is a parsed entity.

Example of an internal entity declaration:

```
<!ENTITY Pub-Status "This is a pre-release of the
 specification.">
```

4.2.2 External Entities

[Definition: If the entity is not internal, it is an **external entity**, declared as follows:]

External Entity Declaration

```
[75]  ExternalID  ::=  'SYSTEM' S SystemLiteral
                       | 'PUBLIC' S PubidLiteral S
                       SystemLiteral
[76]  NDataDecl   ::=  S 'NDATA' S Name                    [VC: Notation Declared]
```

If the NDataDecl is present, this is a general unparsed entity; otherwise it is a parsed entity.

Validity constraint: Notation Declared

The Name must match the declared name of a notation.

[Definition: The SystemLiteral is called the entity's **system identifier**. It is a URI reference (as defined in [IETF RFC 2396], updated by [IETF RFC 2732]), meant to be dereferenced to obtain input for the XML processor to construct the entity's replacement text.] It is an error for a fragment identifier (beginning with a # character) to be part of a system identifier. Unless otherwise provided by information outside the scope of this specification (e.g. a special XML element type defined by a particular DTD, or a processing instruction defined by a particular application specification), relative URIs are relative to the location of the resource within which the entity declaration occurs. A URI might thus be relative to the document entity, to the entity containing the external DTD subset, or to some other external parameter entity.

URI references require encoding and escaping of certain characters. The disallowed characters include all non-ASCII characters, plus the excluded characters listed in Section 2.4 of [IETF RFC 2396], except for the number sign (#) and

percent sign (%) characters and the square bracket characters re-allowed in [IETF RFC 2732]. Disallowed characters must be escaped as follows:

1. Each disallowed character is converted to UTF-8 [IETF RFC 2279] as one or more bytes.
2. Any octets corresponding to a disallowed character are escaped with the URI escaping mechanism (that is, converted to *%HH*, where HH is the hexadecimal notation of the byte value).
3. The original character is replaced by the resulting character sequence.

[Definition: In addition to a system identifier, an external identifier may include a **public identifier**.] An XML processor attempting to retrieve the entity's content may use the public identifier to try to generate an alternative URI reference. If the processor is unable to do so, it must use the URI reference specified in the system literal. Before a match is attempted, all strings of white space in the public identifier must be normalized to single space characters (#x20), and leading and trailing white space must be removed.

Examples of external entity declarations:

```
<!ENTITY open-hatch
         SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
         SYSTEM "../grafix/OpenHatch.gif"
         NDATA gif >
```

4.3 Parsed Entities

4.3.1 The Text Declaration

External parsed entities should each begin with a **text declaration**.

Text Declaration

```
[77]  TextDecl  ::=  '<?xml' VersionInfo? EncodingDecl S? '?>'
```

The text declaration must be provided literally, not by reference to a parsed entity. No text declaration may appear at any position other than the beginning of an external parsed entity. The text declaration in an external parsed entity is not considered part of its replacement text.

4.3.2 Well-Formed Parsed Entities

The document entity is well-formed if it matches the production labeled document. An external general parsed entity is well-formed if it matches the production labeled extParsedEnt. All external parameter entities are well-formed by definition.

Well-Formed External Parsed Entity

```
[78]  extParsedEnt  ::=  TextDecl? content
```

An internal general parsed entity is well-formed if its replacement text matches the production labeled content. All internal parameter entities are well-formed by definition.

A consequence of well-formedness in entities is that the logical and physical structures in an XML document are properly nested; no start-tag, end-tag, empty-element tag, element, comment, processing instruction, character reference, or entity reference can begin in one entity and end in another.

4.3.3 Character Encoding in Entities

Each external parsed entity in an XML document may use a different encoding for its characters. All XML processors must be able to read entities in both the UTF-8 and UTF-16 encodings. The terms "UTF-8" and "UTF-16" in this

specification do not apply to character encodings with any other labels, even if the encodings or labels are very similar to UTF-8 or UTF-16.

Entities encoded in UTF-16 must begin with the Byte Order Mark described by Annex F of [ISO/IEC 10646], Annex H of [ISO/IEC 10646-2000], section 2.4 of [Unicode], and section 2.7 of [Unicode3] (the ZERO WIDTH NO-BREAK SPACE character, #xFEFF). This is an encoding signature, not part of either the markup or the character data of the XML document. XML processors must be able to use this character to differentiate between UTF-8 and UTF-16 encoded documents.

Although an XML processor is required to read only entities in the UTF-8 and UTF-16 encodings, it is recognized that other encodings are used around the world, and it may be desired for XML processors to read entities that use them. In the absence of external character encoding information (such as MIME headers), parsed entities which are stored in an encoding other than UTF-8 or UTF-16 must begin with a text declaration (see **4.3.1 The Text Declaration**) containing an encoding declaration:

Encoding Declaration

```
[80]  EncodingDecl  ::=  S 'encoding' Eq ('"' EncName '"' |
                         "'" EncName "'" )
[81]  EncName       ::=  [A-Za-z] ([A-Za-z0-9._] | '-')*      /* Encoding name contains only
                                                               Latin characters */
```

In the document entity, the encoding declaration is part of the XML declaration. The EncName is the name of the encoding used.

In an encoding declaration, the values "`UTF-8`", "`UTF-16`", "`ISO-10646-UCS-2`", and "`ISO-10646-UCS-4`" should be used for the various encodings and transformations of Unicode / ISO/IEC 10646, the values "`ISO-8859-1`", "`ISO-8859-2`", ... "`ISO-8859-`*n*" (where *n* is the part number) should be used for the parts of ISO 8859, and the values "`ISO-2022-JP`", "`Shift_JIS`", and "`EUC-JP`" should be used for the various encoded forms of JIS X-0208-1997. It is recommended that character encodings registered (as *charset*s) with the Internet Assigned Numbers Authority [IANA-CHARSETS], other than those just listed, be referred to using their registered names; other encodings should use names starting with an "x-" prefix. XML processors should match character encoding names in a case-insensitive way and should either interpret an IANA-registered name as the encoding registered at IANA for that name or treat it as unknown (processors are, of course, not required to support all IANA-registered encodings).

In the absence of information provided by an external transport protocol (e.g. HTTP or MIME), it is an error for an entity including an encoding declaration to be presented to the XML processor in an encoding other than that named in the declaration, or for an entity which begins with neither a Byte Order Mark nor an encoding declaration to use an encoding other than UTF-8. Note that since ASCII is a subset of UTF-8, ordinary ASCII entities do not strictly need an encoding declaration.

It is a fatal error for a TextDecl to occur other than at the beginning of an external entity.

It is a fatal error when an XML processor encounters an entity with an encoding that it is unable to process. It is a fatal error if an XML entity is determined (via default, encoding declaration, or higher-level protocol) to be in a certain encoding but contains octet sequences that are not legal in that encoding. It is also a fatal error if an XML entity contains no encoding declaration and its content is not legal UTF-8 or UTF-16.

Examples of text declarations containing encoding declarations:

```
<?xml encoding='UTF-8'?>
<?xml encoding='EUC-JP'?>
```

4.4 XML Processor Treatment of Entities and References

The table below summarizes the contexts in which character references, entity references, and invocations of unparsed entities might appear and the required behavior of an XML processor in each case. The labels in the leftmost column describe the recognition context:

Reference in Content

as a reference anywhere after the start-tag and before the end-tag of an element; corresponds to the nonterminal content.

Reference in Attribute Value

as a reference within either the value of an attribute in a start-tag, or a default value in an attribute declaration; corresponds to the nonterminal AttValue.

Occurs as Attribute Value

as a Name, not a reference, appearing either as the value of an attribute which has been declared as type **ENTITY**, or as one of the space-separated tokens in the value of an attribute which has been declared as type **ENTITIES**.

Reference in Entity Value

as a reference within a parameter or internal entity's literal entity value in the entity's declaration; corresponds to the nonterminal EntityValue.

Reference in DTD

as a reference within either the internal or external subsets of the DTD, but outside of an EntityValue, AttValue, PI, Comment, SystemLiteral, PubidLiteral, or the contents of an ignored conditional section (see **3.4 Conditional Sections**).

.

	Entity Type				Character
	Parameter	Internal General	External Parsed General	Unparsed	
Reference in Content	*Not recognized*	*Included*	*Included if validating*	*Forbidden*	*Included*
Reference in Attribute Value	*Not recognized*	*Included in literal*	*Forbidden*	*Forbidden*	*Included*
Occurs as Attribute Value	*Not recognized*	*Forbidden*	*Forbidden*	*Notify*	*Not recognized*
Reference in EntityValue	*Included in literal*	*Bypassed*	*Bypassed*	*Forbidden*	*Included*
Reference in DTD	*Included as PE*	*Forbidden*	*Forbidden*	*Forbidden*	*Forbidden*

4.4.1 Not Recognized

Outside the DTD, the % character has no special significance; thus, what would be parameter entity references in the DTD are not recognized as markup in content. Similarly, the names of unparsed entities are not recognized except when they appear in the value of an appropriately declared attribute.

4.4.2 Included

[Definition: An entity is **included** when its replacement text is retrieved and processed, in place of the reference itself, as though it were part of the document at the location the reference was recognized.] The replacement text may contain both character data and (except for parameter entities) markup, which must be recognized in the usual way. (The string "`AT&T;`" expands to "`AT&T;`" and the remaining ampersand is not recognized as an entity-reference delimiter.) A character reference is **included** when the indicated character is processed in place of the reference itself.

4.4.3 Included If Validating

When an XML processor recognizes a reference to a parsed entity, in order to validate the document, the processor must include its replacement text. If the entity is external, and the processor is not attempting to validate the XML document, the processor may, but need not, include the entity's replacement text. If a non-validating processor does not include the replacement text, it must inform the application that it recognized, but did not read, the entity.

This rule is based on the recognition that the automatic inclusion provided by the SGML and XML entity mechanism, primarily designed to support modularity in authoring, is not necessarily appropriate for other applications, in particular document browsing. Browsers, for example, when encountering an external parsed entity reference, might choose to provide a visual indication of the entity's presence and retrieve it for display only on demand.

4.4.4 Forbidden

The following are forbidden, and constitute fatal errors:

- the appearance of a reference to an unparsed entity.
- the appearance of any character or general-entity reference in the DTD except within an EntityValue or AttValue.
- a reference to an external entity in an attribute value.

4.4.5 Included in Literal

When an entity reference appears in an attribute value, or a parameter entity reference appears in a literal entity value, its replacement text is processed in place of the reference itself as though it were part of the document at the location the reference was recognized, except that a single or double quote character in the replacement text is always treated as a normal data character and will not terminate the literal. For example, this is well-formed:

```
<!--  -->
<!ENTITY % YN '"Yes"' >
<!ENTITY WhatHeSaid "He said %YN;" >
```

while this is not:

```
<!ENTITY EndAttr "27'" >
<element attribute='a-&EndAttr;>
```

4.4.6 Notify

When the name of an unparsed entity appears as a token in the value of an attribute of declared type **ENTITY** or **ENTITIES**, a validating processor must inform the application of the system and public (if any) identifiers for both the entity and its associated notation.

4.4.7 Bypassed

When a general entity reference appears in the EntityValue in an entity declaration, it is bypassed and left as is.

4.4.8 Included as PE

Just as with external parsed entities, parameter entities need only be *included if validating*. When a parameter-entity reference is recognized in the DTD and included, its replacement text is enlarged by the attachment of one leading and one following space (#x20) character; the intent is to constrain the replacement text of parameter entities to contain an integral number of grammatical tokens in the DTD. This behavior does not apply to parameter entity references within entity values; these are described in **4.4.5 Included in Literal**.

4.5 Construction of Internal Entity Replacement Text

In discussing the treatment of internal entities, it is useful to distinguish two forms of the entity's value. [Definition: The

literal entity value is the quoted string actually present in the entity declaration, corresponding to the non-terminal EntityValue.] [Definition: The **replacement text** is the content of the entity, after replacement of character references and parameter-entity references.]

The literal entity value as given in an internal entity declaration (EntityValue) may contain character, parameter-entity, and general-entity references. Such references must be contained entirely within the literal entity value. The actual replacement text that is included as described above must contain the *replacement text* of any parameter entities referred to, and must contain the character referred to, in place of any character references in the literal entity value; however, general-entity references must be left as-is, unexpanded. For example, given the following declarations:

```
<!ENTITY % pub    "&#xc9;ditions Gallimard" >
<!ENTITY   rights "All rights reserved" >
<!ENTITY   book   "La Peste: Albert Camus,
&#xA9; 1947 %pub;. &rights;" >
```

then the replacement text for the entity "`book`" is:

```
La Peste: Albert Camus,
© 1947 Éditions Gallimard. &rights;
```

The general-entity reference "`&rights;`" would be expanded should the reference "`&book;`" appear in the document's content or an attribute value.

These simple rules may have complex interactions; for a detailed discussion of a difficult example, see **D Expansion of Entity and Character References**.

4.6 Predefined Entities

[Definition: Entity and character references can both be used to **escape** the left angle bracket, ampersand, and other delimiters. A set of general entities (`amp`, `lt`, `gt`, `apos`, `quot`) is specified for this purpose. Numeric character references may also be used; they are expanded immediately when recognized and must be treated as character data, so the numeric character references "`&#60;`" and "`&#38;`" may be used to escape < and & when they occur in character data.]

All XML processors must recognize these entities whether they are declared or not. For interoperability, valid XML documents should declare these entities, like any others, before using them. If the entities `lt` or `amp` are declared, they must be declared as internal entities whose replacement text is a character reference to the respective character (less-than sign or ampersand) being escaped; the double escaping is required for these entities so that references to them produce a well-formed result. If the entities `gt`, `apos`, or `quot` are declared, they must be declared as internal entities whose replacement text is the single character being escaped (or a character reference to that character; the double escaping here is unnecessary but harmless). For example:

```
<!ENTITY lt     "&#38;#60;">
<!ENTITY gt     "&#62;">
<!ENTITY amp    "&#38;#38;">
<!ENTITY apos   "&#39;">
<!ENTITY quot   "&#34;">
```

4.7 Notation Declarations

[Definition: **Notations** identify by name the format of unparsed entities, the format of elements which bear a notation attribute, or the application to which a processing instruction is addressed.]

[Definition: **Notation declarations** provide a name for the notation, for use in entity and attribute-list declarations and in attribute specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.]

Notation Declarations

```
[82]  NotationDecl  ::=  '<!NOTATION' S Name S (ExternalID |          [VC: Unique Notation Name]
                         PublicID) S? '>'
[83]  PublicID      ::=  'PUBLIC' S PubidLiteral
```

Validity constraint: Unique Notation Name

Only one notation declaration can declare a given Name.

XML processors must provide applications with the name and external identifier(s) of any notation declared and referred to in an attribute value, attribute definition, or entity declaration. They may additionally resolve the external identifier into the system identifier, file name, or other information needed to allow the application to call a processor for data in the notation described. (It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.)

4.8 Document Entity

[Definition: The **document entity** serves as the root of the entity tree and a starting-point for an XML processor.] This specification does not specify how the document entity is to be located by an XML processor; unlike other entities, the document entity has no name and might well appear on a processor input stream without any identification at all.

5 Conformance

5.1 Validating and Non-Validating Processors

Conforming XML processors fall into two classes: validating and non-validating.

Validating and non-validating processors alike must report violations of this specification's well-formedness constraints in the content of the document entity and any other parsed entities that they read.

[Definition: **Validating processors** must, at user option, report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification.] To accomplish this, validating XML processors must read and process the entire DTD and all external parsed entities referenced in the document.

Non-validating processors are required to check only the document entity, including the entire internal DTD subset, for well-formedness. [Definition: While they are not required to check the document for validity, they are required to **process** all the declarations they read in the internal DTD subset and in any parameter entity that they read, up to the first reference to a parameter entity that they do *not* read; that is to say, they must use the information in those declarations to *normalize* attribute values, *include* the replacement text of internal entities, and supply *default attribute values*.] Except when `standalone="yes"`, they must not process entity declarations or attribute-list declarations encountered after a reference to a parameter entity that is not read, since the entity may have contained overriding declarations.

5.2 Using XML Processors

The behavior of a validating XML processor is highly predictable; it must read every piece of a document and report all well-formedness and validity violations. Less is required of a non-validating processor; it need not read any part of the document other than the document entity. This has two effects that may be important to users of XML processors:

- Certain well-formedness errors, specifically those that require reading external entities, may not be detected by a non-validating processor. Examples include the constraints entitled *Entity Declared*, *Parsed Entity*, and *No Recursion*, as well as some of the cases described as *forbidden* in **4.4 XML Processor Treatment of Entities and References**.
- The information passed from the processor to the application may vary, depending on whether the processor reads parameter and external entities. For example, a non-validating processor may not *normalize* attribute values, *include* the replacement text of internal entities, or supply *default attribute values*, where doing so depends on having read declarations in external or parameter entities.

For maximum reliability in interoperating between different XML processors, applications which use non-validating

processors should not rely on any behaviors not required of such processors. Applications which require facilities such as the use of default attributes or internal entities which are declared in external entities should use validating XML processors.

6 Notation

The formal grammar of XML is given in this specification using a simple Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one symbol, in the form

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are the start symbol of a regular language, otherwise with an initial lower case letter. Literal strings are quoted.

Within the expression on the right-hand side of a rule, the following expressions are used to match strings of one or more characters:

#xN

where N is a hexadecimal integer, the expression matches the character in ISO/IEC 10646 whose canonical (UCS-4) code value, when interpreted as an unsigned binary number, has the value indicated. The number of leading zeros in the #xN form is insignificant; the number of leading zeros in the corresponding code value is governed by the character encoding in use and is not significant for XML.

[a-zA-Z], [#xN-#xN]

matches any Char with a value in the range(s) indicated (inclusive).

[abc], [#xN#xN#xN]

matches any Char with a value among the characters enumerated. Enumerations and ranges can be mixed in one set of brackets.

[^a-z], [^#xN-#xN]

matches any Char with a value *outside* the range indicated.

[^abc], [^#xN#xN#xN]

matches any Char with a value not among the characters given. Enumerations and ranges of forbidden values can be mixed in one set of brackets.

"string"

matches a literal string matching that given inside the double quotes.

'string'

matches a literal string matching that given inside the single quotes.

These symbols may be combined to match more complex patterns as follows, where A and B represent simple expressions:

(expression)

expression is treated as a unit and may be combined as described in this list.

A?

matches A or nothing; optional A.

A B

matches A followed by B. This operator has higher precedence than alternation; thus `A B | C D` is identical to `(A B) | (C D)`.

A | B

matches A or B but not both.

A - B

matches any string that matches A but does not match B.

A+

matches one or more occurrences of A.Concatenation has higher precedence than alternation; thus `A+ | B+` is identical to `(A+) | (B+)`.

A*

matches zero or more occurrences of A. Concatenation has higher precedence than alternation; thus `A* | B*` is identical to `(A*) | (B*)`.

Other notations used in the productions are:

/* ... */

comment.

[wfc: ...]

well-formedness constraint; this identifies by name a constraint on well-formed documents associated with a production.

[vc: ...]

validity constraint; this identifies by name a constraint on valid documents associated with a production.

A References

A.1 Normative References

IANA-CHARSETS
(Internet Assigned Numbers Authority) *Official Names for Character Sets*, ed. Keld Simonsen et al. See ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets.

IETF RFC 1766
IETF (Internet Engineering Task Force). *RFC 1766: Tags for the Identification of Languages*, ed. H. Alvestrand. 1995. (See http://www.ietf.org/rfc/rfc1766.txt.)

ISO/IEC 10646
ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).

ISO/IEC 10646-2000
ISO (International Organization for Standardization). *ISO/IEC 10646-1:2000. Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 2000.

Unicode
The Unicode Consortium. *The Unicode Standard, Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press, 1996.

Unicode3
The Unicode Consortium. *The Unicode Standard, Version 3.0.* Reading, Mass.: Addison-Wesley Developers Press, 2000. ISBN 0-201-61633-5.

A.2 Other References

Aho/Ullman
Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Reading: Addison-Wesley, 1986, rpt. corr. 1988.

Berners-Lee et al.
Berners-Lee, T., R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax and Semantics*. 1997. (Work in progress; see updates to RFC1738.)

Brüggemann-Klein
Brüggemann-Klein, Anne. Formal Models in Document Processing. Habilitationsschrift. Faculty of Mathematics at the University of Freiburg, 1993. (See ftp://ftp.informatik.uni-freiburg.de/documents/papers/brueggem/habil.ps.)

Brüggemann-Klein and Wood
Brüggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages*. Universität Freiburg, Institut für Informatik, Bericht 38, Oktober 1991. Extended abstract in A. Finkel, M. Jantzen, Hrsg., STACS 1992, S. 173-184. Springer-Verlag, Berlin 1992. Lecture Notes in Computer Science 577. Full version titled *One-Unambiguous Regular Languages* in Information and Computation 140 (2): 229-253, February 1998.

Clark
James Clark. Comparison of SGML and XML. See http://www.w3.org/TR/NOTE-sgml-xml-971215.

IANA-LANGCODES
(Internet Assigned Numbers Authority) *Registry of Language Tags*, ed. Keld Simonsen et al. (See http://www.isi.edu/in-notes/iana/assignments/languages/.)

IETF RFC2141
IETF (Internet Engineering Task Force). *RFC 2141: URN Syntax*, ed. R. Moats. 1997. (See http://www.ietf.org/rfc/rfc2141.txt.)

IETF RFC 2279
IETF (Internet Engineering Task Force). *RFC 2279: UTF-8, a transformation format of ISO 10646*, ed. F. Yergeau, 1998. (See http://www.ietf.org/rfc/rfc2279.txt.)

IETF RFC 2376
IETF (Internet Engineering Task Force). *RFC 2376: XML Media Types*. ed. E. Whitehead, M. Murata. 1998. (See http://www.ietf.org/rfc/rfc2376.txt.)

IETF RFC 2396
IETF (Internet Engineering Task Force). *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*. T. Berners-Lee, R. Fielding, L. Masinter. 1998. (See http://www.ietf.org/rfc/rfc2396.txt.)

IETF RFC 2732
IETF (Internet Engineering Task Force). *RFC 2732: Format for Literal IPv6 Addresses in URL's*. R. Hinden, B. Carpenter, L. Masinter. 1999. (See http://www.ietf.org/rfc/rfc2732.txt.)

IETF RFC 2781
IETF (Internet Engineering Task Force). *RFC 2781: UTF-16, an encoding of ISO 10646*, ed. P. Hoffman, F. Yergeau. 2000. (See http://www.ietf.org/rfc/rfc2781.txt.)

ISO 639
(International Organization for Standardization). *ISO 639:1988 (E). Code for the representation of names of languages.* [Geneva]: International Organization for Standardization, 1988.

ISO 3166
(International Organization for Standardization). *ISO 3166-1:1997 (E). Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes* [Geneva]: International Organization for Standardization, 1997.

ISO 8879
ISO (International Organization for Standardization). *ISO 8879:1986(E). Information processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML).* First edition -- 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO/IEC 10744
ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology -- Hypermedia/Time-based Structuring Language (HyTime).* [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe.* [Geneva]: International Organization for Standardization, 1996.

WEBSGML
ISO (International Organization for Standardization). *ISO 8879:1986 TC2. Information technology -- Document Description and Processing Languages.* [Geneva]: International Organization for Standardization, 1998. (See http://www.sgmlsource.com/8879rev/n0029.htm.)

XML Names
Tim Bray, Dave Hollander, and Andrew Layman, editors. *Namespaces in XML*. Textuality, Hewlett-Packard, and Microsoft. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/REC-xml-names/.)

B Character Classes

Following the characteristics defined in the Unicode standard, characters are classed as base characters (among others, these contain the alphabetic characters of the Latin alphabet), ideographic characters, and combining characters (among others, this class contains most diacritics) Digits and extenders are also distinguished.

Characters

[84] Letter ::= BaseChar | Ideographic

[85] BaseChar ::= [#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-#x00FF] | [#x0100-#x0131] | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E] | [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5] | [#x01FA-#x0217] | [#x0250-#x02A8] | [#x02BB-#x02C1] | #x0386 | [#x0388-#x038A] | #x038C | [#x038E-#x03A1] | [#x03A3-#x03CE] | [#x03D0-#x03D6] | #x03DA | #x03DC | #x03DE | #x03E0 | [#x03E2-#x03F3] | [#x0401-#x040C] | [#x040E-#x044F] | [#x0451-#x045C] | [#x045E-#x0481] | [#x0490-#x04C4] | [#x04C7-#x04C8] | [#x04CB-#x04CC] | [#x04D0-#x04EB] | [#x04EE-#x04F5] | [#x04F8-#x04F9] | [#x0531-#x0556] | #x0559 | [#x0561-#x0586] | [#x05D0-#x05EA] | [#x05F0-#x05F2] | [#x0621-#x063A] | [#x0641-#x064A] | [#x0671-#x06B7] | [#x06BA-#x06BE] | [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5 | [#x06E5-#x06E6] | [#x0905-#x0939] | #x093D | [#x0958-#x0961] | [#x0985-#x098C] | [#x098F-#x0990] | [#x0993-#x09A8] | [#x09AA-#x09B0] | #x09B2 | [#x09B6-#x09B9] | [#x09DC-#x09DD] | [#x09DF-#x09E1] | [#x09F0-#x09F1] | [#x0A05-#x0A0A] | [#x0A0F-#x0A10] | [#x0A13-#x0A28] | [#x0A2A-#x0A30] | [#x0A32-#x0A33] | [#x0A35-#x0A36] | [#x0A38-#x0A39] | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74] | [#x0A85-#x0A8B] | #x0A8D | [#x0A8F-#x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0] | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0 | [#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28] | [#x0B2A-#x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39] | #x0B3D | [#x0B5C-#x0B5D] | [#x0B5F-#x0B61] | [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-#x0B95] | [#x0B99-#x0B9A] | #x0B9C | [#x0B9E-#x0B9F] | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA] | [#x0BAE-#x0BB5] | [#x0BB7-#x0BB9] | [#x0C05-#x0C0C] | [#x0C0E-#x0C10] | [#x0C12-#x0C28] | [#x0C2A-#x0C33] | [#x0C35-#x0C39] | [#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90] | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3] | [#x0CB5-#x0CB9] | #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C] | [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39] | [#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33] | [#x0E40-#x0E45] | [#x0E81-#x0E82] | #x0E84 | [#x0E87-#x0E88] | #x0E8A | #x0E8D | [#x0E94-#x0E97] | [#x0E99-#x0E9F] | [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7 | [#x0EAA-#x0EAB] | [#x0EAD-#x0EAE] | #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-#x0F69] | [#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103] | [#x1105-#x1107] | #x1109 | [#x110B-#x110C] | [#x110E-#x1112] | #x113C | #x113E | #x1140 | #x114C | #x114E | #x1150 | [#x1154-#x1155] | #x1159 | [#x115F-#x1161] | #x1163 | #x1165 | #x1167 | #x1169 | [#x116D-#x116E] | [#x1172-#x1173] | #x1175 | #x119E | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8] | #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9 | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15] | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D] | [#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D | [#x1F5F-#x1F7D] | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE | [#x1FC2-#x1FC4] | [#x1FC6-#x1FCC] | [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC] | [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126 | [#x212A-#x212B] | #x212E | [#x2180-#x2182] | [#x3041-#x3094] | [#x30A1-#x30FA] | [#x3105-#x312C] | [#xAC00-#xD7A3]

[86] Ideographic ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]

```
[87]  CombiningChar  ::=  [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486] | [#x0591-
                          #x05A1] | [#x05A3-#x05B9] | [#x05BB-#x05BD] | #x05BF | [#x05C1-
                          #x05C2] | #x05C4 | [#x064B-#x0652] | #x0670 | [#x06D6-#x06DC]
                          | [#x06DD-#x06DF] | [#x06E0-#x06E4] | [#x06E7-#x06E8] | [#x06EA-
                          #x06ED] | [#x0901-#x0903] | #x093C | [#x093E-#x094C] | #x094D
                          | [#x0951-#x0954] | [#x0962-#x0963] | [#x0981-#x0983] | #x09BC
                          | #x09BE | #x09BF | [#x09C0-#x09C4] | [#x09C7-#x09C8] | [#x09CB-
                          #x09CD] | #x09D7 | [#x09E2-#x09E3] | #x0A02 | #x0A3C | #x0A3E
                          | #x0A3F | [#x0A40-#x0A42] | [#x0A47-#x0A48] | [#x0A4B-#x0A4D]
                          | [#x0A70-#x0A71] | [#x0A81-#x0A83] | #x0ABC | [#x0ABE-#x0AC5]
                          | [#x0AC7-#x0AC9] | [#x0ACB-#x0ACD] | [#x0B01-#x0B03] | #x0B3C
                          | [#x0B3E-#x0B43] | [#x0B47-#x0B48] | [#x0B4B-#x0B4D] | [#x0B56-
                          #x0B57] | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-#x0BC8]
                          | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03] | [#x0C3E-#x0C44]
                          | [#x0C46-#x0C48] | [#x0C4A-#x0C4D] | [#x0C55-#x0C56] | [#x0C82-
                          #x0C83] | [#x0CBE-#x0CC4] | [#x0CC6-#x0CC8] | [#x0CCA-#x0CCD]
                          | [#x0CD5-#x0CD6] | [#x0D02-#x0D03] | [#x0D3E-#x0D43] | [#x0D46-
                          #x0D48] | [#x0D4A-#x0D4D] | #x0D57 | #x0E31 | [#x0E34-#x0E3A]
                          | [#x0E47-#x0E4E] | #x0EB1 | [#x0EB4-#x0EB9] | [#x0EBB-#x0EBC]
                          | [#x0EC8-#x0ECD] | [#x0F18-#x0F19] | #x0F35 | #x0F37 | #x0F39
                          | #x0F3E | #x0F3F | [#x0F71-#x0F84] | [#x0F86-#x0F8B] | [#x0F90-
                          #x0F95] | #x0F97 | [#x0F99-#x0FAD] | [#x0FB1-#x0FB7] | #x0FB9
                          | [#x20D0-#x20DC] | #x20E1 | [#x302A-#x302F] | #x3099 | #x309A
[88]  Digit          ::=  [#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9] | [#x0966-
                          #x096F] | [#x09E6-#x09EF] | [#x0A66-#x0A6F] | [#x0AE6-#x0AEF]
                          | [#x0B66-#x0B6F] | [#x0BE7-#x0BEF] | [#x0C66-#x0C6F] | [#x0CE6-
                          #x0CEF] | [#x0D66-#x0D6F] | [#x0E50-#x0E59] | [#x0ED0-#x0ED9]
                          | [#x0F20-#x0F29]
[89]  Extender       ::=  #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46 | #x0EC6
                          | #x3005 | [#x3031-#x3035] | [#x309D-#x309E] | [#x30FC-#x30FE]
```

The character classes defined here can be derived from the Unicode 2.0 character database as follows:

- Name start characters must have one of the categories Ll, Lu, Lo, Lt, Nl.
- Name characters other than Name-start characters must have one of the categories Mc, Me, Mn, Lm, or Nd.
- Characters in the compatibility area (i.e. with character code greater than #xF900 and less than #xFFFE) are not allowed in XML names.
- Characters which have a font or compatibility decomposition (i.e. those with a "compatibility formatting tag" in field 5 of the database -- marked by field 5 beginning with a "<") are not allowed.
- The following characters are treated as name-start characters rather than name characters, because the property file classifies them as Alphabetic: [#x02BB-#x02C1], #x0559, #x06E5, #x06E6.
- Characters #x20DD-#x20E0 are excluded (in accordance with Unicode 2.0, section 5.14).
- Character #x00B7 is classified as an extender, because the property list so identifies it.
- Character #x0387 is added as a name character, because #x00B7 is its canonical equivalent.
- Characters ':' and '_' are allowed as name-start characters.
- Characters '-' and '.' are allowed as name characters.

C XML and SGML (Non-Normative)

XML is designed to be a subset of SGML, in that every XML document should also be a conforming SGML document. For a detailed comparison of the additional restrictions that XML places on documents beyond those of SGML, see [Clark].

D Expansion of Entity and Character References (Non-Normative)

This appendix contains some examples illustrating the sequence of entity- and character-reference recognition and expansion, as specified in **4.4 XML Processor Treatment of Entities and References**.

If the DTD contains the declaration

```
<!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>" >
```

then the XML processor will recognize the character references when it parses the entity declaration, and resolve them before storing the following string as the value of the entity "example":

```
<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>
```

A reference in the document to "&example;" will cause the text to be reparsed, at which time the start- and end-tags of the p element will be recognized and the three references will be recognized and expanded, resulting in a p element with the following content (all data, no delimiters or markup):

```
An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;).
```

A more complex example will illustrate the rules and their effects fully. In the following example, the line numbers are solely for reference.

```
1 <?xml version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60;!ENTITY tricky "error-prone" >' >
6 %xx;
7 ]>
8 <test>This sample shows a &tricky; method.</test>
```

This produces the following:

- in line 4, the reference to character 37 is expanded immediately, and the parameter entity "xx" is stored in the symbol table with the value "%zz;". Since the replacement text is not rescanned, the reference to parameter entity "zz" is not recognized. (And it would be an error if it were, since "zz" is not yet declared.)
- in line 5, the character reference "<" is expanded immediately and the parameter entity "zz" is stored with the replacement text "<!ENTITY tricky "error-prone" >", which is a well-formed entity declaration.
- in line 6, the reference to "xx" is recognized, and the replacement text of "xx" (namely "%zz;") is parsed. The reference to "zz" is recognized in its turn, and its replacement text ("<!ENTITY tricky "error-prone" >") is parsed. The general entity "tricky" has now been declared, with the replacement text "error-prone".
- in line 8, the reference to the general entity "tricky" is recognized, and it is expanded, so the full content of the test element is the self-describing (and ungrammatical) string *This sample shows a error-prone method.*

E Deterministic Content Models (Non-Normative)

As noted in **3.2.1 Element Content**, it is required that content models in element type declarations be deterministic. This requirement is for compatibility with SGML (which calls deterministic content models "unambiguous"); XML processors built using SGML systems may flag non-deterministic content models as errors.

For example, the content model ((b, c) | (b, d)) is non-deterministic, because given an initial b the XML processor cannot know which b in the model is being matched without looking ahead to see which element follows the b. In this case, the two references to b can be collapsed into a single reference, making the model read (b, (c | d)). An initial b now clearly matches only a single name in the content model. The processor doesn't need to look ahead to see what follows; either c or d would be accepted.

More formally: a finite state automaton may be constructed from the content model using the standard algorithms, e.g. algorithm 3.5 in section 3.9 of Aho, Sethi, and Ullman [Aho/Ullman]. In many such algorithms, a follow set is constructed for each position in the regular expression (i.e., each leaf node in the syntax tree for the regular expression); if any position has a follow set in which more than one following position is labeled with the same element type name, then the content model is in error and may be reported as an error.

Algorithms exist which allow many but not all non-deterministic content models to be reduced automatically to equivalent deterministic models; see Brüggemann-Klein 1991 [Brüggemann-Klein].

F Autodetection of Character Encodings (Non-Normative)

The XML encoding declaration functions as an internal label on each entity, indicating which character encoding is in use. Before an XML processor can read the internal label, however, it apparently has to know what character encoding is in use--which is what the internal label is trying to indicate. In the general case, this is a hopeless situation. It is not entirely hopeless in XML, however, because XML limits the general case in two ways: each implementation is assumed to support only a finite set of character encodings, and the XML encoding declaration is restricted in position and content in order to make it feasible to autodetect the character encoding in use in each entity in normal cases. Also, in many cases other sources of information are available in addition to the XML data stream itself. Two cases may be distinguished, depending on whether the XML entity is presented to the processor without, or with, any accompanying (external) information. We consider the first case first.

F.1 Detection Without External Encoding Information

Because each XML entity not accompanied by external encoding information and not in UTF-8 or UTF-16 encoding *must* begin with an XML encoding declaration, in which the first characters must be '`<?xml`', any conforming processor can detect, after two to four octets of input, which of the following cases apply. In reading this list, it may help to know that in UCS-4, '<' is "`#x0000003C`" and '?' is "`#x0000003F`", and the Byte Order Mark required of UTF-16 data streams is "`#xFEFF`". The notation ## is used to denote any byte value except that two consecutive ##s cannot be both 00.

With a Byte Order Mark:

`00 00 FE FF`	UCS-4, big-endian machine (1234 order)
`FF FE 00 00`	UCS-4, little-endian machine (4321 order)
`00 00 FF FE`	UCS-4, unusual octet order (2143)
`FE FF 00 00`	UCS-4, unusual octet order (3412)
`FE FF ## ##`	UTF-16, big-endian
`FF FE ## ##`	UTF-16, little-endian
`EF BB BF`	UTF-8

Without a Byte Order Mark:

`00 00 00 3C`
`3C 00 00 00`
`00 00 3C 00`
`00 3C 00 00`
	UCS-4 or other encoding with a 32-bit code unit and ASCII characters encoded as ASCII values, in respectively big-endian (1234), little-endian (4321) and two unusual byte orders (2143 and 3412). The encoding declaration must be read to determine which of UCS-4 or other supported 32-bit encodings applies.

`00 3C 00 3F`	UTF-16BE or big-endian ISO-10646-UCS-2 or other encoding with a 16-bit code unit in big-endian order and ASCII characters encoded as ASCII values (the encoding declaration must be read to determine which)
`3C 00 3F 00`	UTF-16LE or little-endian ISO-10646-UCS-2 or other encoding with a 16-bit code unit in little-endian order and ASCII characters encoded as ASCII values (the encoding declaration must be read to determine which)
`3C 3F 78 6D`	UTF-8, ISO 646, ASCII, some part of ISO 8859, Shift-JIS, EUC, or any other 7-bit, 8-bit, or mixed-width encoding which ensures that the characters of ASCII have their normal positions, width, and values; the actual encoding declaration must be read to detect which of these applies, but since all of these encodings use the same bit patterns for the relevant ASCII characters, the encoding declaration itself may be read reliably
`4C 6F A7 94`	EBCDIC (in some flavor; the full encoding declaration must be read to tell which code page is in use)

Other	UTF-8 without an encoding declaration, or else the data stream is mislabeled (lacking a required encoding declaration), corrupt, fragmentary, or enclosed in a wrapper of some kind

Note:

In cases above which do not require reading the encoding declaration to determine the encoding, section 4.3.3 still requires that the encoding declaration, if present, be read and that the encoding name be checked to match the actual encoding of the entity. Also, it is possible that new character encodings will be invented that will make it necessary to use the encoding declaration to determine the encoding, in cases where this is not required at present.

This level of autodetection is enough to read the XML encoding declaration and parse the character-encoding identifier, which is still necessary to distinguish the individual members of each family of encodings (e.g. to tell UTF-8 from 8859, and the parts of 8859 from each other, or to distinguish the specific EBCDIC code page in use, and so on).

Because the contents of the encoding declaration are restricted to characters from the ASCII repertoire (however encoded), a processor can reliably read the entire encoding declaration as soon as it has detected which family of encodings is in use. Since in practice, all widely used character encodings fall into one of the categories above, the XML encoding declaration allows reasonably reliable in-band labeling of character encodings, even when external sources of information at the operating-system or transport-protocol level are unreliable. Character encodings such as UTF-7 that make overloaded usage of ASCII-valued bytes may fail to be reliably detected.

Once the processor has detected the character encoding in use, it can act appropriately, whether by invoking a separate input routine for each case, or by calling the proper conversion function on each character of input.

Like any self-labeling system, the XML encoding declaration will not work if any software changes the entity's character set or encoding without updating the encoding declaration. Implementors of character-encoding routines should be careful to ensure the accuracy of the internal and external information used to label the entity.

F.2 Priorities in the Presence of External Encoding Information

The second possible case occurs when the XML entity is accompanied by encoding information, as in some file systems and some network protocols. When multiple sources of information are available, their relative priority and the preferred method of handling conflict should be specified as part of the higher-level protocol used to deliver XML. In particular, please refer to [IETF RFC 2376] or its successor, which defines the `text/xml` and `application/xml` MIME types and provides some useful guidance. In the interests of interoperability, however, the following rule is recommended.

- If an XML entity is in a file, the Byte-Order Mark and encoding declaration are used (if present) to determine the character encoding.

G W3C XML Working Group (Non-Normative)

This specification was prepared and approved for publication by the W3C XML Working Group (WG). WG approval of this specification does not necessarily imply that all WG members voted for its approval. The current and former members of the XML WG are:

- Jon Bosak, Sun (*Chair*)
- James Clark (*Technical Lead*)
- Tim Bray, Textuality and Netscape (*XML Co-editor*)
- Jean Paoli, Microsoft (*XML Co-editor*)
- C. M. Sperberg-McQueen, U. of Ill. (*XML Co-editor*)
- Dan Connolly, W3C (*W3C Liaison*)
- Paula Angerstein, Texcel
- Steve DeRose, INSO
- Dave Hollander, HP
- Eliot Kimber, ISOGEN
- Eve Maler, ArborText
- Tom Magliery, NCSA
- Murray Maloney, SoftQuad, Grif SA, Muzmo and Veo Systems
- MURATA Makoto (FAMILY Given), Fuji Xerox Information Systems
- Joel Nava, Adobe
- Conleth O'Connell, Vignette
- Peter Sharpe, SoftQuad

- John Tigue, DataChannel

H W3C XML Core Group (Non-Normative)

The second edition of this specification was prepared by the W3C XML Core Working Group (WG). The members of the WG at the time of publication of this edition were:

- Paula Angerstein, Vignette
- Daniel Austin, Ask Jeeves
- Tim Boland
- Allen Brown, Microsoft
- Dan Connolly, W3C (*Staff Contact*)
- John Cowan, Reuters Limited
- John Evdemon, XMLSolutions Corporation
- Paul Grosso, Arbortext (*Co-Chair*)
- Arnaud Le Hors, IBM (*Co-Chair*)
- Eve Maler, Sun Microsystems (*Second Edition Editor*)
- Jonathan Marsh, Microsoft
- MURATA Makoto (FAMILY Given), IBM
- Mark Needleman, Data Research Associates
- David Orchard, Jamcracker
- Lew Shannon, NCR
- Richard Tobin, University of Edinburgh
- Daniel Veillard, W3C
- Dan Vint, Lexica
- Norman Walsh, Sun Microsystems
- François Yergeau, Alis Technologies (*Errata List Editor*)
- Kongyi Zhou, Oracle

I Production Notes (Non-Normative)

This Second Edition was encoded in the XMLspec DTD (which has documentation available). The HTML versions were produced with a combination of the xmlspec.xsl, diffspec.xsl, and REC-xml-2e.xsl XSLT stylesheets. The PDF version was produced with the html2ps facility and a distiller program.

REC-xml-names-19990114

Namespaces in XML

World Wide Web Consortium 14-January-1999

This version:

http://www.w3.org/TR/1999/REC-xml-names-19990114

http://www.w3.org/TR/1999/REC-xml-names-19990114/xml-names.xml

http://www.w3.org/TR/1999/REC-xml-names-19990114/Overview.html

Latest version:

http://www.w3.org/TR/REC-xml-names

Previous version:

http://www.w3.org/TR/1998/PR-xml-names-19981117

Editors:

Tim Bray (Textuality) <tbray@textuality.com>

Dave Hollander (Hewlett-Packard Company) <dmh@corp.hp.com>

Andrew Layman (Microsoft) <andrewl@microsoft.com>

Status of this document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

The list of known errors in this specification is available at http://www.w3.org/XML/xml-names-19990114-errata.

Please report errors in this document to xml-names-editor@w3.org.

Abstract

XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.

Table of Contents

Appendices

---

1. Motivation and Summary

We envision applications of Extensible Markup Language (XML) where a single XML document may contain elements and attributes (here referred to as a "markup vocabulary") that are defined for and used by multiple software modules. One motivation for this is modularity; if such a markup vocabulary exists which is well-understood and for which there is useful software available, it is better to re-use this markup rather than re-invent it.

Such documents, containing multiple markup vocabularies, pose problems of recognition and collision. Software modules need to be able to recognize the tags and attributes which they are designed to process, even in the face of "collisions" occurring when markup intended for some other software package uses the same element type or attribute name.

These considerations require that document constructs should have universal names, whose scope extends beyond their containing document. This specification describes a mechanism, *XML namespaces*, which accomplishes this.

[Definition:] An **XML namespace** is a collection of names, identified by a URI reference [RFC2396], which are used in XML documents as element types and attribute names. XML namespaces differ from the "namespaces" conventionally used in computing disciplines in that the XML version has internal structure and is not, mathematically speaking, a set. These issues are discussed in "A. The Internal Structure of XML Namespaces".

[Definition:] URI references which identify namespaces are considered **identical** when they are exactly the same character-for-character. Note that URI references which are not identical in this sense may in fact be functionally equivalent. Examples include URI references which differ only in case, or which are in external entities which have different effective base URIs.

Names from XML namespaces may appear as qualified names, which contain a single colon, separating the name into a namespace prefix and a local part. The prefix, which is mapped to a URI reference, selects a namespace. The combination of the universally managed URI namespace and the document's own namespace produces identifiers that are universally unique. Mechanisms are provided for prefix scoping and defaulting.

URI references can contain characters not allowed in names, so cannot be used directly as namespace prefixes. Therefore, the namespace prefix serves as a proxy for a URI reference. An attribute-based syntax described below is used to declare the association of the namespace prefix with a URI reference; software which supports this namespace proposal must recognize and act on these declarations and prefixes.

1.1 A Note on Notation and Usage

Note that many of the nonterminals in the productions in this specification are defined not here but in the XML specification [XML]. When nonterminals defined here have the same names as nonterminals defined in the XML specification, the productions here in all cases match a subset of the strings matched by the corresponding ones there.

In this document's productions, the `NSC` is a "Namespace Constraint", one of the rules that documents conforming to this specification must follow.

Note that all Internet domain names used in examples, with the exception of `w3.org`, are selected at random and should not be taken as having any import.

2. Declaring Namespaces

[Definition:] A namespace is **declared** using a family of reserved attributes. Such an attribute's name must either be `xmlns` or have `xmlns:` as a prefix. These attributes, like any other XML attributes, may be provided directly or by default.

Attribute Names for Namespace Declaration

```
[1]       NSAttName :: PrefixedAttName
                    =
                       | DefaultAttName
[2] PrefixedAttName :: 'xmlns:' NCName                [ NSC: Leading
                    =                                   "XML" ]
[3]  DefaultAttName :: 'xmlns'
                    =
[4]           NCName :: (Letter | '_') (NCNameChar)  / An XML Name,
                    =  *                           * minus the
                                                      ":" */
[5]       NCNameChar :: Letter | Digit | '.' | '-'
                    =  | '_' | CombiningChar
                       | Extender
```

[Definition:] The attribute's value, a URI reference, is the **namespace name** identifying the namespace. The namespace name, to serve its intended purpose, should have the characteristics of uniqueness and persistence. It is not a goal that it be directly usable for retrieval of a schema (if any exists). An example of a syntax that is designed with these goals in mind is that for Uniform Resource Names [RFC2141]. However, it should be noted that ordinary URLs can be managed in such a way as to achieve these same goals.

[Definition:] If the attribute name matches `PrefixedAttName`, then the `NCName` gives the **namespace prefix**, used to associate element and attribute names with the namespace name in the attribute value in the scope of the element to which the declaration is attached. In such declarations, the namespace name may not be empty.

[Definition:] If the attribute name matches `DefaultAttName`, then the namespace name in the attribute value is that of the **default namespace** in the scope of the element to which the declaration is attached. In such a default declaration, the attribute value may be empty. Default namespaces and overriding of declarations are discussed in "5. Applying Namespaces to Elements and Attributes".

An example namespace declaration, which associates the namespace prefix `edi` with the namespace name `http://ecommerce.org/schema`:

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <!-- the "edi" prefix is bound to http://ecommerce.org/schema
       for the "x" element and contents -->
</x>
```

Namespace Constraint: Leading "XML"

Prefixes beginning with the three-letter sequence `x`, `m`, `l`, in any case combination, are reserved for use

by XML and XML-related specifications.

3. Qualified Names

[Definition:] In XML documents conforming to this specification, some names (constructs corresponding to the nonterminal Name) may be given as **qualified names**, defined as follows:

Qualified Name

```
[6]     QName ::= (Prefix ':')? LocalPart
[7]    Prefix ::= NCName
[8] LocalPart ::= NCName
```

The Prefix provides the namespace prefix part of the qualified name, and must be associated with a namespace URI reference in a namespace declaration. [Definition:] The LocalPart provides the **local part** of the qualified name.

Note that the prefix functions *only* as a placeholder for a namespace name. Applications should use the namespace name, not the prefix, in constructing names whose scope extends beyond the containing document.

4. Using Qualified Names

In XML documents conforming to this specification, element types are given as qualified names, as follows:

Element Types

```
 [9]          STag ::= '<' QName (S Attribute)* S? '>'      [ NSC: Prefix Declared ]
[10]          ETag ::= '</' QName S? '>'                    [ NSC: Prefix Declared ]
[11] EmptyElemTag ::= '<' QName (S Attribute)* S? '/>'      [ NSC: Prefix Declared ]
```

An example of a qualified name serving as an element type:

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <!-- the 'price' element's namespace is http://ecommerce.org/
schema -->
  <edi:price units='Euro'>32.18</edi:price>
</x>
```

Attributes are either namespace declarations or their names are given as qualified names:

Attribute

```
[12] Attribute :: NSAttName Eq AttValue
                =
                   | QName Eq AttValue   [ NSC: Prefix Declared ]
```

An example of a qualified name serving as an attribute name:

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <!-- the 'taxClass' attribute's namespace is http://ecommerce.
org/schema -->
  <lineItem edi:taxClass="exempt">Baby food</lineItem>
</x>
```

Namespace Constraint: Prefix Declared
The namespace prefix, unless it is `xml` or `xmlns`, must have been declared in a namespace declaration attribute in either the start-tag of the element where the prefix is used or in an an ancestor element (i.e. an element in whose content the prefixed markup occurs). The prefix `xml` is by definition bound to the namespace name `http://www.w3.org/XML/1998/namespace`. The prefix `xmlns` is used only for namespace bindings and is not itself bound to any namespace name.

This constraint may lead to operational difficulties in the case where the namespace declaration attribute is provided, not directly in the XML document entity, but via a default attribute declared in an external entity. Such declarations may not be read by software which is based on a non-validating XML processor. Many XML applications, presumably including namespace-sensitive ones, fail to require validating processors. For correct operation with such applications, namespace declarations must be provided either directly or via default attributes declared in the internal subset of the DTD.

Element names and attribute types are also given as qualified names when they appear in declarations in the DTD:

Qualified Names in Declarations

```
[13] doctypedecl :: '<!DOCTYPE' S QName (S ExternalID)? S?
                 = ('[' (markupdecl | PEReference | S)* ']' S?)?
                   '>'
[14] elementdecl :: '<!ELEMENT' S QName S contentspec S? '>'
                 =
[15]          cp :: (QName | choice | seq) ('?' | '*' | '+')?
                 =
[16]       Mixed :: '(' S? '#PCDATA' (S? '|' S? QName)* S? ')*'
                 =
                   | '(' S? '#PCDATA' S? ')'
[17] AttlistDecl :: '<!ATTLIST' S QName AttDef* S? '>'
                 =
[18]      AttDef :: S (QName | NSAttName) S AttType S DefaultDecl
                 =
```

5. Applying Namespaces to Elements and Attributes

5.1 Namespace Scoping

The namespace declaration is considered to apply to the element where it is specified and to all elements within the content of that element, unless overridden by another namespace declaration with the same NSAttName part:

```
<?xml version="1.0"?>
<!-- all elements here are explicitly in the HTML namespace -->
<html:html xmlns:html='http://www.w3.org/TR/REC-html40'>
  <html:head><html:title>Frobnostication</html:title></html:head>
  <html:body><html:p>Moved to
    <html:a href='http://frob.com'>here.</html:a></html:p></html:
body>
</html:html>
```

Multiple namespace prefixes can be declared as attributes of a single element, as shown in this example:

```
<?xml version="1.0"?>
<!-- both namespace prefixes are available throughout -->
<bk:book xmlns:bk='urn:loc.gov:books'
         xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <bk:title>Cheaper by the Dozen</bk:title>
    <isbn:number>1568491379</isbn:number>
</bk:book>
```

5.2 Namespace Defaulting

A default namespace is considered to apply to the element where it is declared (if that element has no namespace prefix), and to all elements with no prefix within the content of that element. If the URI reference in a default namespace declaration is empty, then unprefixed elements in the scope of the declaration are not considered to be in any namespace. Note that default namespaces do not apply directly to attributes.

```
<?xml version="1.0"?>
<!-- elements are in the HTML namespace, in this case by default --
>
<html xmlns='http://www.w3.org/TR/REC-html40'>
  <head><title>Frobnostication</title></head>
  <body><p>Moved to
    <a href='http://frob.com'>here</a>.</p></body>
</html>
```

```
<?xml version="1.0"?>
<!-- unprefixed element types are from "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <title>Cheaper by the Dozen</title>
    <isbn:number>1568491379</isbn:number>
</book>
```

A larger example of namespace scoping:

```
<?xml version="1.0"?>
<!-- initially, the default namespace is "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <title>Cheaper by the Dozen</title>
    <isbn:number>1568491379</isbn:number>
    <notes>
      <!-- make HTML the default namespace for some commentary -->
      <p xmlns='urn:w3-org-ns:HTML'>
          This is a <i>funny</i> book!
      </p>
    </notes>
</book>
```

The default namespace can be set to the empty string. This has the same effect, within the scope of the declaration, of there being no default namespace.

```
<?xml version='1.0'?>
<Beers>
  <!-- the default namespace is now that of HTML -->
  <table xmlns='http://www.w3.org/TR/REC-html40'>
   <th><td>Name</td><td>Origin</td><td>Description</td></th>
   <tr>
     <!-- no default namespace inside table cells -->
     <td><brandName xmlns="">Huntsman</brandName></td>
     <td><origin xmlns="">Bath, UK</origin></td>
     <td>
       <details xmlns=""><class>Bitter</class><hop>Fuggles</hop>
         <pro>Wonderful hop, light alcohol, good summer beer</pro>
         <con>Fragile; excessive variance pub to pub</con>
         </details>
        </td>
     </tr>
    </table>
  </Beers>
```

5.3 Uniqueness of Attributes

In XML documents conforming to this specification, no tag may contain two attributes which:

1. have identical names, or
2. have qualified names with the same local part and with prefixes which have been bound to namespace names that are identical.

For example, each of the `bad` start-tags is illegal in the following:

```
<!-- http://www.w3.org is bound to n1 and n2 -->
<x xmlns:n1="http://www.w3.org"
   xmlns:n2="http://www.w3.org" >
  <bad a="1"     a="2" />
  <bad n1:a="1"  n2:a="2" />
</x>
```

However, each of the following is legal, the second because the default namespace does not apply to attribute names:

```
<!-- http://www.w3.org is bound to n1 and is the default -->
<x xmlns:n1="http://www.w3.org"
   xmlns="http://www.w3.org" >
  <good a="1"     b="2" />
  <good a="1"     n1:a="2" />
</x>
```

6. Conformance of Documents

In XML documents which conform to this specification, element types and attribute names must match the production for `QName` and must satisfy the "Namespace Constraints".

An XML document conforms to this specification if all other tokens in the document which are required, for XML conformance, to match the XML production for `Name`, match this specification's production for `NCName`.

The effect of conformance is that in such a document:

- All element types and attribute names contain either zero or one colon.
- No entity names, PI targets, or notation names contain any colons.

Strictly speaking, attribute values declared to be of types `ID`, `IDREF(S)`, `ENTITY(IES)`, and `NOTATION` are also `Names`, and thus should be colon-free. However, the declared type of attribute values is only available to processors which read markup declarations, for example validating processors. Thus, unless the use of a validating processor has been specified, there can be no assurance that the contents of attribute values have been checked for conformance to this specification.

Appendices

A. The Internal Structure of XML Namespaces (Non-Normative)

A.1 The Insufficiency of the Traditional Namespace

In the computing disciplines, the term "namespace" conventionally refers to a *set* of names, i.e. a collection containing no duplicates. However, treating the names used in XML markup as such a namespace would greatly impair their usefulness. The primary use of such names in XML documents is to enable identification of logical structures in documents by software modules such as query processors, stylesheet-driven rendering engines, and schema-driven validators. Consider the following example:

```
<section><title>Book-Signing Event</title>
<signing>
  <author title="Mr" name="Vikram Seth" />
  <book title="A Suitable Boy" price="$22.95" /></signing>
<signing>
  <author title="Dr" name="Oliver Sacks" />
  <book title="The Island of the Color-Blind" price="$12.95" /></
signing>
</section>
```

In this example, there are three occurrences of the name `title` within markup, and the name alone clearly provides insufficient information to allow correct processing by a software module.

Another problematic area comes from the use of "global" attributes, as illustrated by this example, a fragment of an XML document which is to be displayed using a CSS stylesheet:

```
<RESERVATION>
 <NAME HTML:CLASS="largeSansSerif">Layman, A</NAME>
 <SEAT CLASS="Y" HTML:CLASS="reallyImportant">33B</SEAT>
 <DEPARTURE>1997-05-24T07:55:00+1</DEPARTURE></RESERVATION>
```

In this case, the `CLASS` attribute, which describes the fare basis and takes values such as "J", "Y", and "C", is distinct at all semantic levels from the `HTML:CLASS` attribute, which is used to simulate syntactic richness in HTML, as a means of overcoming the limited element repertoire by subclassing.

XML 1.0 does not provide a built-in way to declare "global" attributes; items such as the HTML `CLASS` attribute are global only in their prose description and their interpretation by HTML applications. However, such attributes, an important distinguishing feature of which is that their names are unique, are commonly observed to occur in a variety of applications.

A.2 XML Namespace Partitions

In order to support the goal of making both qualified and unqualified names useful in meeting their intended purpose, we identify the names appearing in an XML namespace as belonging to one of several disjoint traditional (i.e. set-structured) namespaces, called namespace partitions. The partitions are:

The All Element Types Partition
: All element types in an XML namespace appear in this partition. Each has a unique local part; the combination of the namespace name and the local part uniquely identifies the element type.

The Global Attribute Partition
: This partition contains the names of all attributes which are defined, in this namespace, to be global. The only required characteristic of a global attribute is that its name be unique in the global attribute partition. This specification makes no assertions as to the proper usage of such attributes. The combination of the namespace name and the attribute name uniquely identifies the global attribute.

The Per-Element-Type Partitions
: Each type in the All Element Types Partition has an associated namespace in which appear the names of the unqualified attributes that are provided for that element. This is a traditional namespace because the appearance of duplicate attribute names on an element is forbidden by XML 1.0. The combination of the attribute name with the element's type and namespace name uniquely identifies each unqualified attribute.

In XML documents conforming to this specification, the names of all qualified (prefixed) attributes are assigned to the global attribute partition, and the names of all unqualified attributes are assigned to the appropriate per-element-type partition.

A.3 Expanded Element Types and Attribute Names

For convenience in specifying rules and in making comparisons, we define an expanded form, expressed here in XML element syntax, for each element type and attribute name in an XML document.

[Definition:] An **expanded element type** is expressed as an empty XML element of type `ExpEType`. It has a required `type` attribute which gives the type's `LocalPart`, and an optional `ns` attribute which, if the element is qualified, gives its namespace name.

[Definition:] An **expanded attribute name** is expressed as an empty XML element of type `ExpAName`. It has a required `name` attribute which gives the name. If the attribute is global, it has a required `ns` attribute which gives the namespace name; otherwise, it has a required attribute `eltype` which gives the type of the attached element, and an optional attribute `elns` which gives the namespace name, if known, of the attached element.

Slight variations on the examples given above will illustrate the working of expanded element types

and attribute names. The following two fragments are each followed by a table showing the expansion of the names:

```
<!-- 1 --> <section xmlns='urn:com:books-r-us'>
<!-- 2 -->   <title>Book-Signing Event</title>
<!-- 3 -->   <signing>
<!-- 4 -->     <author title="Mr" name="Vikram Seth" />
<!-- 5 -->     <book title="A Suitable Boy" price="$22.95" />
             </signing>
           </section>
```

The names would expand as follows:

Line	Name	Expanded
1	section	<ExpEType type="section" ns="urn:com:books-r-us" />
2	title	<ExpEType type="title" ns="urn:com:books-r-us" />
3	signing	<ExpEType type="signing" ns="urn:com:books-r-us" />
4	author	<ExpEType type="author" ns="urn:com:books-r-us" />
4	title	<ExpAName name='title' eltype="author" elns="urn:com:books-r-us" />
4	name	<ExpAName name='name' eltype="author" elns="urn:com:books-r-us" />
5	book	<ExpEType type="book" ns="urn:com:books-r-us" />
5	title	<ExpAName name='title' eltype="book" elns="urn:com:books-r-us" />
5	price	<ExpAName name='price' eltype="book" elns="urn:com:books-r-us" />

```
<!-- 1 --> <RESERVATION xmlns:HTML="http://www.w3.org/TR/REC-
html40">
<!-- 2 --> <NAME HTML:CLASS="largeSansSerif">Layman, A</NAME>
<!-- 3 --> <SEAT CLASS="Y" HTML:CLASS="largeMonotype">33B</SEAT>
<!-- 4 --> <HTML:A HREF='/cgi-bin/ResStatus'>Check Status</HTML:A>
<!-- 5 --> <DEPARTURE>1997-05-24T07:55:00+1</DEPARTURE></
RESERVATION>
```

1	RESERVATION	<ExpEType type="RESERVATION" />

2	NAME	<ExpEType type="NAME" />
2	HTML:CLASS	<ExpAName name="CLASS" ns=http://www.w3.org/TR/REC-html40 />
3	SEAT	<ExpEType type="SEAT" />
3	CLASS	<ExpAName name="CLASS" eltype="SEAT">
3	HTML:CLASS	<ExpAName name="CLASS" ns="http://www.w3.org/TR/REC-html40" />
4	HTML:A	<ExpEType type="A" ns="http://www.w3.org/TR/REC-html40" />
4	HREF	<ExpAName name="HREF" eltype="A" elns="http://www.w3.org/TR/REC-html40" />
5	DEPARTURE	<ExpEType type="DEPARTURE" />

A.4 Unique Expanded Attribute Names

The constraint expressed by "5.3 Uniqueness of Attributes" above may straightforwardly be implemented by requiring that no element have two attributes whose expanded names are equivalent, i. e. have the same attribute-value pairs.

B. Acknowledgements (Non-Normative)

This work reflects input from a very large number of people, including especially the members of the World Wide Web Consortium XML Working Group and Special Interest Group and the participants in the W3C Metadata Activity. The contributions of Charles Frankston of Microsoft were particularly valuable.

C. References

RFC2141

IETF (Internet Engineering Task Force) *RFC 2141: URN Syntax*, ed. R. Moats. May 1997.

RFC2396

IETF (Internet Engineering Task Force) *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*, eds. T. Berners-Lee, R. Fielding, L. Masinter. August 1998.

XML

Extensible Markup Language (XML) 1.0, eds. Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. 10 February 1998. Available at http://www.w3.org/TR/REC-xml.

| Translations

XML in 10 points

***XML, XLink, Namespace, DTD, Schema, CSS, XHTML* ... If you are new to XML, it may be hard to know where to begin. This summary in 10 points attempts to capture enough of the basic concepts to enable a beginner to see the forest through the trees. And if you are giving a presentation on XML, why not start with these 10 points?**

---

1. XML is for structuring data

Structured data includes things like spreadsheets, address books, configuration parameters, financial transactions, and technical drawings. XML is a set of rules (you may also think of them as guidelines or conventions) for designing text formats that let you structure your data. XML is not a programming language, and you don't have to be a programmer to use it or learn it. XML makes it easy for a computer to generate data, read data, and ensure that the data structure is unambiguous. XML avoids common pitfalls in language design: it is extensible, platform-independent, and it supports internationalization and localization. XML is fully Unicode-compliant.

2. XML looks a bit like HTML

Like HTML, XML makes use of *tags* (words bracketed by '<' and '>') and *attributes* (of the form `name="value"`). While HTML specifies what each tag and attribute means, and often how the text between them will look in a browser, XML uses the tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that reads it. In other words, if you see "<p>" in an XML file, do not assume it is a paragraph. Depending on the context, it may be a price, a parameter, a person, a p... (and who says it has to be a word with a "p"?).

3. XML is text, but isn't meant to be read

Programs that produce spreadsheets, address books, and other structured data often store that data on disk, using either a binary or text format. One advantage of a text format is that it allows people, if necessary, to look at the data without the program that produced it; in a pinch, you can read a text format with your favorite text editor. Text formats also allow developers to more easily debug applications. Like HTML, XML files are text files that people shouldn't have to read, but may when the need arises. Compared to HTML, the rules for XML files allow fewer variations. A forgotten tag, or an attribute without quotes makes an XML file unusable, while in HTML such practice is often explicitly allowed. The official XML specification forbids applications from trying to second-guess the creator of a broken XML file; if the file is broken, an application has to stop right there and report an error.

4. XML is verbose by design

Since XML is a text format and it uses tags to delimit the data, XML files are nearly always larger than comparable binary formats. That was a conscious decision by the designers of XML. The advantages of a text format are evident (see point 3), and the disadvantages can usually be compensated at a different level. Disk space is less expensive than it used to be, and compression programs like zip and gzip can compress files very well and very fast. In addition, communication protocols such as modem protocols and HTTP/1.1, the core protocol of the Web, can compress data on the fly, saving bandwidth as effectively as a binary format.

5. XML is a family of technologies

XML 1.0 is the specification that defines what "tags" and "attributes" are. Beyond XML 1.0, "the XML family" is a growing set of modules that offer useful services to accomplish important and frequently demanded tasks. *XLink* describes a standard way to add hyperlinks to an XML file. *XPointer* is a syntax in development for pointing to parts of an XML document. An XPointer is a bit like a URL, but instead of pointing to

documents on the Web, it points to pieces of data inside an XML file. *CSS*, the style sheet language, is applicable to XML as it is to HTML. *XSL* is the advanced language for expressing style sheets. It is based on *XSLT*, a transformation language used for rearranging, adding and deleting tags and attributes. The *DOM* is a standard set of function calls for manipulating XML (and HTML) files from a programming language. *XML Schemas 1* and *2* help developers to precisely define the structures of their own XML-based formats. There are several more modules and tools available or under development. Keep an eye on W3C's technical reports page.

6. XML is new, but not that new

Development of XML started in 1996 and it has been a W3C Recommendation since February 1998, which may make you suspect that this is rather immature technology. In fact, the technology isn't very new. Before XML there was SGML, developed in the early '80s, an ISO standard since 1986, and widely used for large documentation projects. The development of HTML started in 1990. The designers of XML simply took the best parts of SGML, guided by the experience with HTML, and produced something that is no less powerful than SGML, and vastly more regular and simple to use. Some evolutions, however, are hard to distinguish from revolutions... And it must be said that while SGML is mostly used for technical documentation and much less for other kinds of data, with XML it is exactly the opposite.

7. XML leads HTML to XHTML

There is an important XML application that is a document format: W3C's XHTML, the successor to HTML. XHTML has many of the same elements as HTML. The syntax has been changed slightly to conform to the rules of XML. A format that is "XML-based" inherits the syntax from XML and restricts it in certain ways (e.g, XHTML allows "<p>", but not "<r>"); it also adds meaning to that syntax (XHTML says that "<p>" stands for "paragraph", and not for "price", "person", or anything else).

8. XML is modular

XML allows you to define a new document format by combining and reusing other formats. Since two formats developed independently may have elements or attributes with the same name, care must be taken when combining those formats (does "<p>" mean "paragraph" from this format or "person" from that one?). To eliminate name confusion when combining formats, XML provides a *namespace* mechanism. XSL and RDF are good examples of XML-based formats that use namespaces. *XML Schema* is designed to mirror this support for modularity at the level of defining XML document structures, by making it easy to combine two schemas to produce a third which covers a merged document structure.

9. XML is the basis for RDF and the Semantic Web

W3C's Resource Description Framework (RDF) is an XML text format that supports resource description and metadata applications, such as music playlists, photo collections, and bibliographies. For example, RDF might let you identify people in a Web photo album using information from a personal contact list; then your mail client could automatically start a message to those people stating that their photos are on the Web. Just as HTML integrated documents, images, menu systems, and forms applications to launch the original Web, RDF provides tools to integrate even more, to make the Web a little bit more into a Semantic Web. Just like people need to have agreement on the meanings of the words they employ in their communication, computers need mechanisms for agreeing on the meanings of terms in order to communicate effectively. Formal descriptions of terms in a certain area (shopping or manufacturing, for example) are called ontologies and are a necessary part of the Semantic Web. RDF, ontologies, and the representation of meaning so that computers can help people do work are all topics of the Semantic Web Activity.

10. XML is license-free, platform-independent and well-supported

By choosing XML as the basis for a project, you gain access to a large and growing community of tools (one of which may already do what you need!) and engineers experienced in the technology. Opting for XML is a bit like choosing SQL for databases: you still have to build your own database and your own programs and procedures that manipulate it, but there are many tools available and many people who can help you. And since XML

is license-free, you can build your own software around it without paying anybody anything. The large and growing support means that you are also not tied to a single vendor. *XML isn't always the best solution, but it is always worth considering.*

W3C Communications Team, w3t-comm@w3.org
Revised 13 Nov. 2001 (last update: $Date: 2003/06/02 19:08:30 $)
Created 27 Mar 1999 by Bert Bos
(Previous version)

Economist.com PREMIUM CONTENT

LAST WORD

The Babbage of the web

Dec 7th 2000
From The Economist Technology Quarterly

Ted Nelson imagined hypertext in 1960—but his vision failed to become a reality. Now the web has eaten his lunch. But Mr Nelson hopes that his innovative ideas will yet prevail

---

The article you requested is premium content. To read it, please log in, or pick a payment or subscription option, on the right.

An Economist.com gift voucher is the ideal gift, allowing the recipient to enjoy all *Economist* articles for an entire year. Buy your gift subscription now and get 25% off our normal subscription rate. Click here.

Print subscriptions to *The Economist* include a free subscription to Economist.com for the term of your print subscription (more about print subscriptions). If you **already subscribe** to *The Economist* in print, click here to activate your free online access.

Print subscriptions to *The Economist* include a free subscription to Economist.com for the term of your print subscription (more about print subscriptions). If you **already subscribe** to *The Economist* in print, click here to activate your free online access.

Select an option below to continue:

Subscribe

Get unlimited site access. Sign up for:

- ○ a month, $19.95 (automatic renewal)
- ◉ a year, $89 (automatic renewal)
- ○ a print subscription (complimentary web access)

Subscribe

Pay Per View

Buy access to view the individual article:

- ○ 1 article-credit, $2.95
- ◉ Use credits already on my account

Go

Current Subscribers

Log in to view the article.

E-mail address

Password

Remember my password on this computer? ◉ Yes ○ No

Log in

- Expired website subscription? Click here to renew.
- Current print subscriber? Activate your free account.

[Mirrored from: http://dlar.fcit.monash.edu.au/~dfoott/2800/lect08.html]

COT1800 Public Networks
Lecture 8 SGML

Standard Generalised Markup Language

`Found IT,' the Mouse replied rather crossly: `of course you know what "it" means.'

`I know what "it" means well enough, when I find a thing,' said the Duck: `it's generally a frog or a worm. The question is, what did the archbishop find?'

Banal Markup

All printed texts are encoded. There are conventions that are applied to printed text to indicate all sorts of functions and operations within the text. Often these relate to the speech origin of text, with marks to indicate breathing, pauses for effect, and so on.

Punctuation marks, use of capitalization, disposition of letters around the page, even the spaces between words, might be regarded as a kind of markup, the function of which is to help the human reader determine where words and concepts end or to identify broad structural features.

Document Markup

- Document markup is
  - the process of adding codes to a document to identify the structure or the format in which it is to appear.
  - a communication form that has existed for many years. Until the computerization of the printing industry, markup was primarily done by a copy editor writing instructions on a manuscript for a typesetter to follow.
    These included instructions on the type of font to be used, the size to be used, whether bold or italics were to be used, to indicate underlining if it were necesary, to indicate spacing, hyphenation, indentation and so on.
    Every facet of the appearance of the printed work was represented in code to the typesetter.

Editorial Markup to Typesetting

Over a period of time, a standard set of symbols was developed and used by copy editors to communicate with typesetters.

As typesetting functions became computerized, text formatting languages were written. A typesetter would convert the copy editor's markup into the appropriate markup for the text formatting language being used.

Word Processing

One of the earliest applications of computing was to the setting up of text. Because the rules were able to be codified they leant themselves to a computing operation. In particular the typesetter's tasks, though requiring considerable human skill, were able to be performed easily and quickly by machine. Early text or type setting programs were the parent of word processors.
Some general features of word processors [as mark-up applications] include

- Each word processing program had/has its own [proprietary] method of markup.
  For a long time this meant that documents prepared using one word processing tool were not able to be read by others. Often documents were migrated in a "lowest common denominator" form, often as ASCII code with formatting lost, and then reformatted in the new environment.
- Most electronic devices which store text for later recall and output use some form of markup.
  Even when text is stored as unformatted ASCII code there are still some markup elements retained - spaces, commas and other punctuation, line-feed and carriage-return characters, at least at the end of paragraphs, beginning and end of document markers, etc.
- The markup may or may not be apparent to the user.
- The markup may be visible, hidden, entered by the user, or automatically generated.
- In some cases document markup takes the form of alphanumeric text characters and in other cases it is stored as binary data.
- In most cases, some form of delimiter is used to indicate the beginning of the markup and optionally the end of the markup.
- Although the power of a word processing program to format a document is impressive, it is also a burden when one wants to switch from one platform [software or hardware] to another.

Conversion Utilities

- To move between two word processors 2 convertors are needed, but for 4 the number is 12 [the general formula is **n*(n-1)**]
- If a common [intermediate] standard is accepted the number of convertors, using the common language, is **2n**
  Clearly there are significant advantages in the existence and use of a common base encoding system, even if only in terms of moving documents between platforms while retaining the inherent structure of the document.

The Generic Coding Concept

Historically, electronic manuscripts contained control codes or macros that caused the document to be formatted in a particular way ('specific coding').

In contrast, generic coding, which began in the late 1960s, uses descriptive tags(for example, 'heading', rather than 'format-17').

3 Important Stages

William Tunnicliffe, chairman of the Graphic Communications Association (GCA) Composition Committee, during a meeting at the Canadian Government Printing Office in September 1967 emphasised the separation of the information content of documents from their format.

Also in the late 1960s, a New York book designer named Stanley Rice proposed the idea of a universal catalog of parameterized 'editorial structure' tags.

Norman Scharpf, director of the GCA, recognized the significance of these trends, and established a generic coding project in the Composition Committee

Charles Goldfarb

In 1969, Charles Goldfarb was leading an IBM research project on integrated law office information systems.

Goldfarb, with Edward Mosher and Raymond Lorie invented the Generalized Markup Language (GML) as a means of allowing the text editing, formatting, and information retrieval subsystems to share documents.

GML

GML (which, not coincidentally, comprises the initials of its three inventors) was based on the generic coding ideas of Rice and Tunnicliffe.

Instead of a simple tagging scheme, however, GML introduced the concept of a formally-defined document type with an explicit nested element structure.

Development of SGML as an International Standard

In 1978, the American National Standards Institute (ANSI) committee on Information Processing established the Computer Languages for the Processing of Text committee

Goldfarb was asked to join the committee and eventually to lead a project for a text description language standard based on GML.

International Standard

- In 1985, a draft proposal for an international standard was published
- The international SGML Users' Group was founded in the the UK by Joan Smith, who became its first president.
- A draft international standard was published in October 1985, and approved in 1986 (ISO 8879:1986).
  Other national standard bodies have also adopted the standard, usually unaltered from the ISO standard. Australia is no exception.

Early Applications of SGML

- Two early applications were developed with much broad participation:
  - the Electronic Manuscript Project of the Association of American Publishers (AAP),
  - the documentation component of the Computer-aided Acquisition and Logistic Support (CALS) initiative of the US Department of Defense.

3 Elements of SGML

You can break a typical document into three layers: structure , content, and style.

SGML separates these three aspects, but deals mainly with the relationship between structure [DTD] and content [tagged material].

Structure - the DTD [Document Type Definition]

- describes the structure of a document, much like a database schema describes the types of information it handles and the relationships between fields.
- provides a framework for the elements that constitute a document.
- A DTD also specifies rules for the relationships between elements; eg. "a chapter heading must be the first element after the start of a chapter"

Content

- The information itself: content includes titles, paragraphs, lists, tables, graphics, and audio.
- Identifying the content's position within the DTD structure is called "tagging."
- Creating an SGML document involves inserting tags around content. These tags mark the beginning and end of each part of the structure.
  There are, of course, authoring tools that enable tags to be "automatically" added to the text.

SGML Declaration

- Declares the Document to be an SGML document
- Specifies if a particular syntax has been used
- Defines character sets that are used

Advantages of SGML

- Machine/platform independence
- Portability over time
- Directly appropriate for information retrieval and hypertext retrieval
- International standard and character sets

Tags

Many tags can be declared but typical are:

- <abstract>
- <author>
- <cit> for citation
- <hdref> for cross reference to heading
- for a list of items
- <top1> for a top level topic etc

What it looks like

This should look fairly familiar to you. HTML, the markup that is used in WWW documents, is a specific instance of SGML - a limited DTD is available, and inherent in each Web browsing software application, and the markup uses a limited set of SGML tags.

<text> <tPage>
<dTitle type=main>PARADISE LOST</dTitle>
<byLine>by
<dAuthor>John Milton</dAuthor>
</byLine>
</tPage>
</front>
<body>

<head>BOOK I.</head>
<l> Of Mans First Disobedience, and the Fruit
<l>Of that Forbidden Tree, whose mortal tast
<l>Brought Death into the World, and all our woe,

A typical DTD segment

This is from the Text Encoding Initiative Document Type Definition, a specific set of DTDs which have developed to handle the majority of literary modes [novels, poetry, drama, etc]. The intention is

that it may not be necessary to transmit the DTD with the document if it is simply announced in the header that the TEI DTD is used, and the TEI DTD is widely distributed.
The TEI DTDs allow the production of proper scholarly editions of works, complete with alternate readings, and allow the document to be presented as an authorised edition would appear. This is moving electronic publishing into the realms of being able to match printed editions. The material presented on screen [and, if necessary, printed] will look like the recognised print edition.

<!DOCTYPE ota SYSTEM "ota.dtd" [
<!ENTITY % OTAents system "unixlat0.dtd">
%OTAents
<!ENTITY file1 SYSTEM "plost.1827">
]>
<ota n="1827" crdate="1993-02-02" update="1993-07-19">
<header>
<fileDesc>
<titlStmt>
<title>Milton's "Paradise Lost": electronic edition</title>
<edStmt>
Public Domain TEI edition prepared at the Oxford Text Archive
<extent>
Filesize uncompressed: 498 Kbytes.<pubStmt>

UNIVERSITY OF FLORIDA
Cooperative Extension Service
Institute of Food and Agricultural Sciences

edis

Brief History of Document Markup[1]

Dennis G. Watson[2]

INTRODUCTION

Document markup is the process of adding codes to a document to identify the structure of a document or the format in which it is to appear. Document markup is a communication form that has existed for many years. Until the computerization of the printing industry, markup was primarily done by a copy editor writing instructions on a manuscript for a typesetter to follow (Figure 1). Over a period of time, a standard set of symbols was developed and used by copy editors to communicate with typesetters.

As typesetting functions became computerized, text formatting languages were written. A typesetter would convert the copy editor's markup into the appropriate markup for the text formatting language being used. Figure 2 is an example of the commands used by a typesetting system to print the chapter heading of a document.

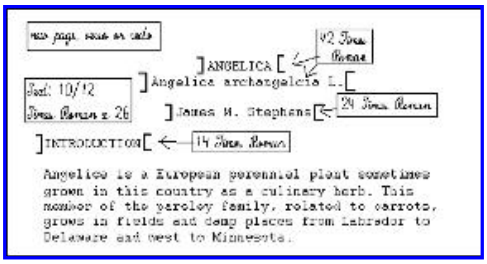


Figure 1. Portion of a marked-up page with instructions for a typesetter.

The style of the document described in Figure 2 calls for the chapter heading to be one line below the normal starting point, the font to be changed to 16 point roman, and the heading to be centered and printed in bold. Since many text formatting languages were developed, many different instruction sets exist.

As computers became widely available, authors began using word processing software to write and edit their documents. Each word processing program had its own method of markup. Most electronic devices which store text for later recall and output use some form of markup. The markup may or may not be apparent to the user. The markup may be visible, hidden, entered by the user, or automatically generated. In some cases document markup takes the form of alphanumeric text characters and in other cases it is stored as binary data. In most cases, some form of delimiter is used to indicate the beginning of the markup and optionally the end of the markup. Although the power of a word processing program to format a document is impressive, it is also a burden when one wants to switch to the next generation computer and software. When upgrading equipment and software, old electronic documents must be converted to the new format. Sometimes this is handled automatically by the new software. At other times, it is not and an author must reenter formatting information, if not the entire text.

Two categories of document markup are specific markup and generic markup. Specific markup uses instructions which are specific to the software being used to prepare or output a document. Generalized markup describes the structure of a document. For example, specific markup might include marking a heading as centered and bold. With generic markup, a first level heading could simply be marked as "head1."

```
.sp
.bf roman16
.bd .be Chapter 1. Introduction
```

Figure 2. Sample commands for a text formatting language which describe the appearance of a chapter heading.

Figure 3. Sample commands for a text formatting language which describe the appearance of a chapter heading.

Text formatting languages are another example of specific markup (Figure 2). The commands used by the language are specific to the text processing software and usually have to be reentered in order for the text to be processed by a different text processing program.

Interactive word processors allow an author to add specific markup to a document as it is being written. This allows an author to generate a visually pleasing document, but the markup is specific to the word processing program being used.

GENERIC MARKUP

Generic or generalized markup is the term that describes the process of assigning generic names to markup. Figure 3 is an example of a generalized markup applied to the same text as Figure 2 .

Macros for typesetting languages were the beginning of generic markup. A macro is a software instruction which executes a series of instructions (a type of shorthand notation). A "chapter" macro can be defined with the contents of the instructions in Figure 2 , to print a chapter heading. This explains the simplicity of the markup in Figure 3 . The macro can also keep track of chapter numbers. If the publisher wants to change the appearance of the chapter heading, only the macro would need to be changed, not each individual heading in the document. The style information on the appearance of a document is kept separate from its structure and content. The assumption behind generic markup is that documents have a structure consisting of logical components.

With word processors that use style sheets, the same level of generic markup can be obtained. A style sheet describes the appearance of a document. Titles can be marked with a "title" style, authors' names can be marked with an "author" style and so on. Figure 4 is an example of WordPerfect document in reveal codes with style codes being used. A publishing group can have multiple style sheets with the same style names, with each style sheet being used for a different appearance of the printed page. Generic markup specifies the logical structure rather than the appearance of a document.

TOWARD A STANDARD

With the availability of computers and capabilities of transferring electronic data to typesetters, large printing customers began looking at ways to reduce costs. First, customers began sending computer files to publishers, thinking this would save them the cost of the typesetter having to rekey the manuscript. They expected big savings, but the publisher offered only a nominal discount. The publisher argued that they had to go through the document character by character and insert typesetting codes for typeface, font sizes, bold, italic, foreign characters, figures, tables, charts, etc.

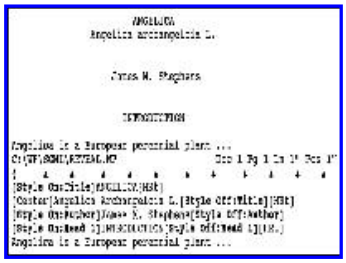


Figure 4. WordPerfect document in reveal codes, illustrating generic markup of title, author, and first level heading with styles (style boldface).

Some customers decided to ask for the typesetters' codes so they could enter these codes at the time the document was being edited. Typically, multiple typesetters were used, and as the typesetters responded, several facts became obvious. Different typesetters used completely different coding schemes. One required a `.bo' command preceding a word or words to be printed in bold, whereas another required a `{fbo##' command. For another typesetter, the command had to be on a line by itself. Typesetters typically reminded the customers that they would still have to charge for proofing each typesetting command. Another limitation was that the only way to save on reprinting costs was to use the same publisher as for the original printing, because of the different coding schemes in use.

Obviously, a better method was needed to encourage the sharing of data among organizations without having to redo the typesetting codes each time. Another need was for a marked document to have a life longer than the software for which it was originally coded.

STANDARDIZATION TIMELINE

The need for longevity of markup and the need for marking documents for electronic database storage led to the development of the international Standard Generalized Markup Language (SGML). Following is a summary of key events which led to SGML.

1967 - William Tunnicliffe spoke on separation of information content of documents from their format, at a meeting at the Canadian Government Printing Office.

Late 60s - Stanley Rice, a New York book designer, proposed a set of parameterized `editorial structure' tags. The Graphic Communications Association (GCA) helped sponsor workshops, seminars, and committees to further develop the concept. From these efforts grew the original GCA GenCode committee. GenCode defined a generalized markup approach based on a document's hierarchy. The markup approach was integrated with a generic coding that emphasized descriptive rather than procedural coding.

1969 - Charles Goldfarb, Edward Mosher, and Raymond Lorie invented the Generalized Markup Language (GML) for IBM. GML was based on the generic coding ideas of Rice and Tunnicliffe.

1970 - Goldfarb proposed a generalized markup language based on the premises that: 1) markup should describe a document's structure rather than its physical characteristics, and 2) markup should be rigorous so that it can be unambiguously understood by a program or a human interpreter.

1978 - An ANSI working group was formed and supported by GenCode and subsequently led by Goldfarb to provide an unambiguous format for text interchange and a markup language rich enough to permit future processing. Their work was based on GML. As the committee considered methods of handling markup, they realized that it needed to be generic. One of the concepts they decided on was to mark a title as <title> rather than <bold> and . By marking a title as <title>, database searches could be limited to titles. This was the beginning of SGML, which represents the structure of a document. Another innovation of the committee was that one could specify the order in which objects could appear in a document (e.g. TO before FROM in a memo). The committee borrowed the concept of header files (or referenced files) from programming languages (such as C) and included tags which the markup procedure would use in a header file and the actual use of the markup in a separate file.

1980 - First draft of SGML, by ANSI committee.

1983 - Sixth working draft of SGML released. Adopted by the Internal Revenue Service and the US Department of Defense.

1984 - SGML working group reorganized under ISO and ANSI concurrently with Goldfarb serving as the technical leader and editor for both groups. The ISO working group was ISO/IEC JTCI/SC18/WG8.

1985 - Draft international standard published.

1986 - SGML approved as ISO international standard 8879.

1991 - Procedure initiated for 5-year review of SGML.

SUMMARY

The process of adding markup to documents has evolved from specific typesetting codes which were often different from one typesetter to another, to a standardized generic markup language. SGML (Standard Generalized Markup Language - ISO 8879) has achieved widespread acceptance and is being used by major government agencies such as the Department of Defense and the Internal Revenue Service. With generic markup as defined by SGML, the process of converting an electronic document from one computer system to another can be automated with computer software. Automated processing of text for inclusion in large information databases is a significant step for providing timely, pertinent information in the information age.

Footnotes

1. This document is Circular 1086, one of a series of the Agricultural and Biological Engineering Department, Florida Cooperative Extension Service, Institute of Food and Agricultural Sciences, University of Florida. Original publication date November, 1992. Reviewed July, 2002. Visit the EDIS Web Site at http://edis.ifas.ufl.edu.

2. Dennis G. Watson, associate professor, Agricultural Engineering Department, Cooperative Extension Service, Institute of Food and Agricultural Sciences, University of Florida, Gainesville FL 32611.

Copyright Information

NOTE-sgml-xml-971215

Comparison of SGML and XML

World Wide Web Consortium Note 15-December-1997

This version:
http://www.w3.org/TR/NOTE-sgml-xml-971215

Author:
James Clark <jjc@jclark.com>

Status of this document

This document is a NOTE made available by the W3 Consortium for discussion only. This indicates no endorsement of its content, nor that the Consortium has, is, or will be allocating any resources to the issues addressed by the NOTE.
Errors or omissions in this document should be reported to the author.

Abstract

This document provides a detailed comparison of SGML (ISO 8879) and XML.

Comparison of SGML and XML

Version 1.0

Table of Contents

---

1. Differences Between XML and SGML

XML allows only documents that use the SGML declaration in this note. This declares all the following SGML features as NO:

- DATATAG
- OMITTAG
- RANK
- LINK (SIMPLE, IMPLICIT and EXPLICIT)
- CONCUR
- SUBDOC
- FORMAL

Note that it differs from the reference concrete syntax in a number of ways:

- It also declares no short reference delimiters; it follows that SHORTREF and USEMAP declarations cannot occur in XML
- The PIC (processing instruction close) delimiter is ?>
- Quantities and capacities are effectively unlimited
- Names are case sensitive (NAMECASE GENERAL is NO)
- Underscore and colon are allowed in names
- Names can use Unicode characters and are not restricted to ASCII

The following constructs which are permitted in SGML when SHORTTAG is YES are not allowed in XML:

- Unclosed start-tags
- Unclosed end-tags
- Empty start-tags
- Empty end-tags
- Attribute values in attribute specifications entered directly rather than as literals
- Attribute specifications that omit the attribute name

NET delimiters can be used only to close an empty element. In SGML without the Web SGML Adaptations Annex, the NET delimiter is declared as />. With this approach, XML is not allowing null end-tags and is allowing net-enabling start-tags only for elements with no end-tag. In SGML with the Web SGML Adaptations Annex, there is a separate NESTC (net-enabling start tag close) delimiter. This allows the XML <e/> syntax to be handled as a combination of a net-enabling start-tag <e/ and a null end-tag >. With this approach, XML is allowing a net-enabling start-tag only when immediately followed by a null end-tag.

XML imposes the following restrictions not in SGML:

- Entity references

    - Entity references must be closed with a REFC delimiter
    - References to external data entities in content are not allowed
    - General entity references in content are required to be synchronous
    - External entity references in attribute values are not allowed
    - Parameter entity references are allowed in the internal subset only within a declaration separator (that is, at a point where a markup declaration could occur)
- Character references
    - Character references must be closed with a REFC delimiter
    - Named character references are not allowed
    - Numeric character references to non-SGML characters are not allowed
- Entity declarations
    - A #DEFAULT entity cannot be declared
    - External SDATA entities are not allowed
    - External CDATA entities are not allowed
    - Internal SDATA entities are not allowed
    - Internal CDATA entities are not allowed
    - PI entities are not allowed
    - Bracketed text entities are not allowed
    - External identifiers must include a system identifier
    - Attributes cannot be specified for an entity
    - The replacement text of general text entities and external parameter entities is required to be well-formed
    - An ampersand in a parameter literal must be followed by a syntactically valid entity reference or numeric character reference
- Attribute definition list declarations
    - Associated element type in attribute definition list declarations cannot be a name group
    - Attributes cannot be declared for a notation
    - CURRENT attributes are not allowed
    - Content reference attributes are not allowed
    - NUTOKEN(S) declared values are not allowed
    - NUMBER(S) declared values are not allowed
    - NAME(S) declared values are not allowed
    - A name token group must use the or connector
    - Attribute values specified as defaults in attribute definition list declarations must be literals (SGML allows them not to be even when SHORTTAG is NO)
- Element type declarations
    - Associated element type in element type declaration cannot be a name group
    - In an element declaration, a generic identifier cannot be specified as a rank stem and rank suffix (SGML allows this even when the RANK feature is NO)
    - Minimization parameters in element declarations are not allowed
    - RCDATA declared content are not allowed
    - CDATA declared content are not allowed
    - Content models cannot use the and connector
    - Content models for mixed content have a restricted form
    - Inclusions are not allowed
    - Exclusions are not allowed
- Comments

    - A parameter separator cannot contain comments; this means that markup declarations (other than comment declarations) cannot contain comments
    - Empty comment declarations (`<!>` in the reference concrete syntax) are not allowed
    - A comment declaration cannot contain more than one comment
    - In a comment declaration, an S separator is not allowed before the final `MDC`
- Processing instructions
    - Processing instructions must start with a name (the PI target)
    - A processing instruction whose PI target is `xml` can only occur at the beginning of a external entity and must be an XML declaration if it occurs in the document entity, and otherwise an text declaration
    - A PI target must not match `[Xx][Mm][Ll]` unless it is `xml`
- Marked sections
    - In marked section declarations, `TEMP` status keyword is not allowed
    - `RCDATA` marked sections are not allowed
    - `INCLUDE`/`IGNORE` marked sections are not allowed in the document instance
    - In a marked section declaration, a status keyword specification that contains no status keywords is not allowed
    - In a marked section declaration, a status keyword specification cannot contain more than one status keyword
    - Marked sections are not allowed in the internal subset
    - Parameter separators are not allowed in status keyword specifications in the document instance; in particular, parameter entity references are not allowed
- Other
    - Names beginning with `[Xx][Mm][Ll]` are reserved
    - The SGML declaration must be implied and cannot be explicitly present in the document entity
    - When `<` and `&` occur as data, they must be entered as `<` and `&`
    - A parameter separator required by the formal syntax must always be present and cannot be omitted when it is adjacent to a delimiter

XML predefines the semantics of the attributes `xml:space` and `xml:lang`. It also reserves all attribute, element type and notation names beginning with `[Xx][Mm][Ll]`.

XML requires that an SGML parser use an entity manager that behaves as follows:

- Lines are terminated by newline (Unicode code #X000A) rather than being delimited by RS and RE as with a typical SGML entity manager
- System identifiers are treated as URLs
- The entity manager must support entities encoded in UTF-16 and UTF-8, and must be able automatically to detect which encoding an entity uses based on the presence of the byte order mark
- The entity manager should be able to recognize the encoding declaration in the XML declaration and encoding PI and use it to determine the encoding of entity

XML imposes requirements on the information that a parser must make available to an application.

XML depends on the following changes to SGML made by Web SGML Adaptations Annex:

- HCRO delimiter (for hex numeric character references); for XML this is &#X
- EMPTYNRM feature that allows elements declared EMPTY to have end-tags
- NESTC delimiter
- Duplicate enumerated attribute tokens are allowed
- Relaxation of rules on use of parameter entity references inside groups
- Multiple ATTLIST declarations for a single element type
- ATTLIST declarations which don't declare any attributes
- KEEPRSRE feature that turns off SGML's rules for ignoring RSs and REs
- Fully-tagged SGML documents; a document that is fully-tagged but not type-valid is a conforming SGML document; this makes all XML documents, including those that are well-formed but not valid, conforming SGML documents
- Predefined data character entities in the SGML declaration (for lt, amp and so on)
- Unlimited capacities and quantities

The Web SGML Adaptations Annex also enables some XML restrictions to be enforced in SGML:

- SHORTTAG is unbundled, so the SGML declaration can allow attribute defaulting and NET without allowing other SHORTTAG constructs
- The SGML declaration can assert that a document is integrally stored, which disallows improperly nested entity references in content

2. Transforming SGML to XML

For most restrictions in XML that go beyond SGML, it is possible to transform an SGML document automatically into a document that meets the restrictions, and is equivalent in the sense that it has the same ESIS. There are a number of restrictions for which this is not the case:

External SDATA entities, external CDATA entities
: These could be transformed into NDATA entities.

Subdocument entities
: These could be converted into NDATA entities with a notation that indicates that they are SGML or XML.

References to external data entities in content
: These could be transformed into an empty element with an attribute whose declared value is ENTITY.

Data attributes
: Since an external data entity can only be used in an ENTITY or ENTITIES attribute on an element, these could be transformed into other attributes on the element.

Internal SDATA entities
: References could be transformed into numeric character references to the appropriate Unicode character; if used in an entity or entities attribute, the entity will have to be made external.

Internal CDATA entities
: If used in an ENTITY or ENTITIES attribute, the entity will have to be made external (references to CDATA entities are not part of ESIS).

`PI` entities

If they contain `?>`, they cannot be converted into an XML PI. It could be an application convention that entity references are replaced in PIs. Also if they do not start with a name, they cannot be converted into a well-formed XML PI.

names

An SGML document can have a concrete syntax which allows characters in names that XML does not allow in names.

3. SGML Declaration for XML

The following SGML declaration takes advantage of the Extended Naming Rules Technical Corrigendum to ISO 8879, but does not make use of the Web SGML Adaptations Annex:

```
<!SGML -- SGML Declaration for XML --
     "ISO 8879:1986 (ENR)"

     CHARSET
         BASESET
             "ISO Registration Number 176//CHARSET
             ISO/IEC 10646-1:1993 UCS-
4 with implementation
             level 3//ESC 2/5 2/15 4/6"
         DESCSET
                 0       9       UNUSED
                 9       2       9
                 11      2       UNUSED
                 13      1       13
                 14      18      UNUSED
                 32      95      32
                 127     1       UNUSED
                 128     32      UNUSED
                 160     55136   160
                 55296   2048    UNUSED  -- surrogates --
                 57344   8190    57344
                 65534   2       UNUSED  -- FFFE and FFFF --
                 65536   1048576 65536
     CAPACITY SGMLREF
         -- Capacities are not restricted in XML --
         TOTALCAP 99999999
         ENTCAP   99999999
         ENTCHCAP 99999999
         ELEMCAP  99999999
```

```
          GRPCAP   99999999
          EXGRPCAP 99999999
          EXNMCAP  99999999
          ATTCAP   99999999
          ATTCHCAP 99999999
          AVGRPCAP 99999999
          NOTCAP   99999999
          NOTCHCAP 99999999
          IDCAP    99999999
          IDREFCAP 99999999
          MAPCAP   99999999
          LKSETCAP 99999999
          LKNMCAP  99999999


     SCOPE DOCUMENT

     SYNTAX
          SHUNCHAR NONE
          BASESET "ISO Registration Number 176//CHARSET
                   ISO/IEC 10646-1:1993 UCS-
4 with implementation
                   level 3//ESC 2/5 2/15 4/6"
          DESCSET
              0 1114112 0
          FUNCTION
              RE    13
              RS    10
              SPACE 32
              TAB   SEPCHAR 9

          NAMING
              LCNMSTRT ""
              UCNMSTRT ""
              NAMESTRT
                  58 95 192-214 216-246 248-305 308-318 321-
328
                  330-382 384-451 461-496 500-501 506-
535 592-680
                  699-705 902 904-906 908 910-929 931-
974 976-982
                  986 988 990 992 994-1011 1025-1036 1038-
```

```
1103
                1105-1116 1118-1153 1168-1220 1223-1224
                1227-1228 1232-1259 1262-1269 1272-1273
                1329-1366 1369 1377-1414 1488-1514 1520-
1522
                1569-1594 1601-1610 1649-1719 1722-1726
                1728-1742 1744-1747 1749 1765-1766 2309-
2361
                2365 2392-2401 2437-2444 2447-2448 2451-
2472
                2474-2480 2482 2486-2489 2524-2525 2527-
2529
                2544-2545 2565-2570 2575-2576 2579-2600
                2602-2608 2610-2611 2613-2614 2616-2617
                2649-2652 2654 2674-2676 2693-2699 2701
                2703-2705 2707-2728 2730-2736 2738-2739
                2741-2745 2749 2784 2821-2828 2831-2832
                2835-2856 2858-2864 2866-2867 2870-
2873 2877
                2908-2909 2911-2913 2949-2954 2958-2960
                2962-2965 2969-2970 2972 2974-2975 2979-
2980
                2984-2986 2990-2997 2999-3001 3077-3084
                3086-3088 3090-3112 3114-3123 3125-3129
                3168-3169 3205-3212 3214-3216 3218-3240
                3242-3251 3253-3257 3294 3296-3297 3333-
3340
                3342-3344 3346-3368 3370-3385 3424-3425
                3585-3630 3632 3634-3635 3648-3653 3713-
3714
                3716 3719-3720 3722 3725 3732-3735 3737-
3743
                3745-3747 3749 3751 3754-3755 3757-
3758 3760
                3762-3763 3773 3776-3780 3904-3911 3913-
3945
                4256-4293 4304-4342 4352 4354-4355 4357-
4359
                4361 4363-4364 4366-
4370 4412 4414 4416 4428
                4430 4432 4436-4437 4441 4447-
```

```
4449 4451 4453
                4455 4457 4461-4462 4466-
4467 4469 4510 4520
                4523 4526-4527 4535-4536 4538 4540-
4546 4587
                4592 4601 7680-7835 7840-7929 7936-7957
                7960-7965 7968-8005 8008-8013 8016-
8023 8025
                8027 8029 8031-8061 8064-8116 8118-
8124 8126
                8130-8132 8134-8140 8144-8147 8150-8155
                8160-8172 8178-8180 8182-8188 8486 8490-
8491
                8494 8576-8578 12295 12321-12329 12353-
12436
                12449-12538 12549-12588 19968-40869 44032-
55203

            LCNMCHAR ""
            UCNMCHAR ""
            NAMECHAR
                45-46 183 720-721 768-837 864-
865 903 1155-1158
                1425-1441 1443-1465 1467-1469 1471 1473-
1474
                1476 1600 1611-1618 1632-1641 1648 1750-
1764
                1767-1768 1770-1773 1776-1785 2305-
2307 2364
                2366-2381 2385-2388 2402-2403 2406-2415
                2433-2435 2492 2494-2500 2503-2504 2507-
2509
                2519 2530-2531 2534-2543 2562 2620 2622-
2626
                2631-2632 2635-2637 2662-2673 2689-
2691 2748
                2750-2757 2759-2761 2763-2765 2790-2799
                2817-2819 2876 2878-2883 2887-2888 2891-
2893
                2902-2903 2918-2927 2946-2947 3006-3010
                3014-3016 3018-3021 3031 3047-3055 3073-
```

```
3075
                3134-3140 3142-3144 3146-3149 3157-3158
                3174-3183 3202-3203 3262-3268 3270-3272
                3274-3277 3285-3286 3302-3311 3330-3331
                3390-3395 3398-3400 3402-3405 3415 3430-
3439
                3633 3636-3642 3654-3662 3664-3673 3761
                3764-3769 3771-3772 3782 3784-3789 3792-
3801
                3864-3865 3872-3881 3893 3895 3897 3902-
3903
                3953-3972 3974-3979 3984-3989 3991 3993-
4013
                4017-4023 4025 8400-8412 8417 12293 12330-
12335
                12337-12341 12441-12442 12445-12446 12540-
12542

            NAMECASE
                GENERAL NO
                ENTITY  NO


          DELIM
              GENERAL SGMLREF
              NET "/>"
              PIC "?>"
              SHORTREF NONE
          NAMES
              SGMLREF


          QUANTITY SGMLREF
              -- Quantities are not restricted in XML --
              ATTCNT      99999999
              ATTSPLEN    99999999
              -- BSEQLEN  not used --
              -- DTAGLEN  not used --
              -- DTEMPLEN not used --
              ENTLVL      99999999
              GRPCNT      99999999
              GRPGTCNT    99999999
```

```
            GRPLVL      99999999
            LITLEN      99999999
            NAMELEN     99999999
            -- no need to change NORMSEP --
            PILEN       99999999
            TAGLEN      99999999
            TAGLVL      99999999


    FEATURES
        MINIMIZE
            DATATAG NO
            OMITTAG NO
            RANK NO
            SHORTTAG YES -- SHORTTAG is needed for NET --
        LINK
            SIMPLE NO
            IMPLICIT NO
            EXPLICIT NO
        OTHER
            CONCUR NO
            SUBDOC NO
            FORMAL NO
    APPINFO NONE
>
```

The following SGML declaration takes advantage of the Web SGML Adaptations Annex to ISO 8879:

```
<!SGML -- SGML Declaration for XML --
     "ISO 8879:1986 (WWW)"

     CHARSET
         BASESET
             "ISO Registration Number 176//CHARSET
             ISO/IEC 10646-1:1993 UCS-
4 with implementation
             level 3//ESC 2/5 2/15 4/6"
         DESCSET
                0       9       UNUSED
                9       2       9
                11      2       UNUSED
```

```
                13      1       13
                14      18      UNUSED
                32      95      32
                127     1       UNUSED
                128     32      UNUSED
                160     55136   160
                55296   2048    UNUSED  -- surrogates --
                57344   8190    57344
                65534   2       UNUSED  -- FFFE and FFFF --
                65536   1048576 65536
    CAPACITY NONE

    SCOPE DOCUMENT

    SYNTAX
        SHUNCHAR NONE
        BASESET "ISO Registration Number 176//CHARSET
                 ISO/IEC 10646-1:1993 UCS-
4 with implementation
                 level 3//ESC 2/5 2/15 4/6"
        DESCSET
            0 1114112 0
        FUNCTION
            RE    13
            RS    10
            SPACE 32
            TAB   SEPCHAR 9

        NAMING
            LCNMSTRT ""
            UCNMSTRT ""
            NAMESTRT
                58 95 192-214 216-246 248-305 308-318 321-
328
                330-382 384-451 461-496 500-501 506-
535 592-680
                699-705 902 904-906 908 910-929 931-
974 976-982
                986 988 990 992 994-1011 1025-1036 1038-
1103
                1105-1116 1118-1153 1168-1220 1223-1224
```

```
                  1227-1228 1232-1259 1262-1269 1272-1273
                  1329-1366 1369 1377-1414 1488-1514 1520-
1522
                  1569-1594 1601-1610 1649-1719 1722-1726
                  1728-1742 1744-1747 1749 1765-1766 2309-
2361
                  2365 2392-2401 2437-2444 2447-2448 2451-
2472
                  2474-2480 2482 2486-2489 2524-2525 2527-
2529
                  2544-2545 2565-2570 2575-2576 2579-2600
                  2602-2608 2610-2611 2613-2614 2616-2617
                  2649-2652 2654 2674-2676 2693-2699 2701
                  2703-2705 2707-2728 2730-2736 2738-2739
                  2741-2745 2749 2784 2821-2828 2831-2832
                  2835-2856 2858-2864 2866-2867 2870-
2873 2877
                  2908-2909 2911-2913 2949-2954 2958-2960
                  2962-2965 2969-2970 2972 2974-2975 2979-
2980
                  2984-2986 2990-2997 2999-3001 3077-3084
                  3086-3088 3090-3112 3114-3123 3125-3129
                  3168-3169 3205-3212 3214-3216 3218-3240
                  3242-3251 3253-3257 3294 3296-3297 3333-
3340
                  3342-3344 3346-3368 3370-3385 3424-3425
                  3585-3630 3632 3634-3635 3648-3653 3713-
3714
                  3716 3719-3720 3722 3725 3732-3735 3737-
3743
                  3745-3747 3749 3751 3754-3755 3757-
3758 3760
                  3762-3763 3773 3776-3780 3904-3911 3913-
3945
                  4256-4293 4304-4342 4352 4354-4355 4357-
4359
                  4361 4363-4364 4366-
4370 4412 4414 4416 4428
                  4430 4432 4436-4437 4441 4447-
4449 4451 4453
                  4455 4457 4461-4462 4466-
```

```
4467 4469 4510 4520
                4523 4526-4527 4535-4536 4538 4540-
4546 4587
                4592 4601 7680-7835 7840-7929 7936-7957
                7960-7965 7968-8005 8008-8013 8016-
8023 8025
                8027 8029 8031-8061 8064-8116 8118-
8124 8126
                8130-8132 8134-8140 8144-8147 8150-8155
                8160-8172 8178-8180 8182-8188 8486 8490-
8491
                8494 8576-8578 12295 12321-12329 12353-
12436
                12449-12538 12549-12588 19968-40869 44032-
55203

            LCNMCHAR ""
            UCNMCHAR ""
            NAMECHAR
                45-46 183 720-721 768-837 864-
865 903 1155-1158
                1425-1441 1443-1465 1467-1469 1471 1473-
1474
                1476 1600 1611-1618 1632-1641 1648 1750-
1764
                1767-1768 1770-1773 1776-1785 2305-
2307 2364
                2366-2381 2385-2388 2402-2403 2406-2415
                2433-2435 2492 2494-2500 2503-2504 2507-
2509
                2519 2530-2531 2534-2543 2562 2620 2622-
2626
                2631-2632 2635-2637 2662-2673 2689-
2691 2748
                2750-2757 2759-2761 2763-2765 2790-2799
                2817-2819 2876 2878-2883 2887-2888 2891-
2893
                2902-2903 2918-2927 2946-2947 3006-3010
                3014-3016 3018-3021 3031 3047-3055 3073-
3075
                3134-3140 3142-3144 3146-3149 3157-3158
```

```
                3174-3183 3202-3203 3262-3268 3270-3272
                3274-3277 3285-3286 3302-3311 3330-3331
                3390-3395 3398-3400 3402-3405 3415 3430-
3439
                3633 3636-3642 3654-3662 3664-3673 3761
                3764-3769 3771-3772 3782 3784-3789 3792-
3801
                3864-3865 3872-3881 3893 3895 3897 3902-
3903
                3953-3972 3974-3979 3984-3989 3991 3993-
4013
                4017-4023 4025 8400-8412 8417 12293 12330-
12335
                12337-12341 12441-12442 12445-12446 12540-
12542

            NAMECASE
                GENERAL NO
                ENTITY  NO

        DELIM
            GENERAL SGMLREF
            HCRO "&#38;
#x" -- 38 is the number for ampersand --
            NESTC "/"
            NET ">"
            PIC "?>"
            SHORTREF NONE

        NAMES
            SGMLREF

        QUANTITY NONE

        ENTITIES
            "amp" 38
            "lt" 60
            "gt" 62
            "quot" 34
            "apos" 39

    FEATURES
```

```
        MINIMIZE
            DATATAG NO
            OMITTAG NO
            RANK NO
            SHORTTAG
                STARTTAG
                    EMPTY NO
                    UNCLOSED NO
                    NETENABL IMMEDNET
                ENDTAG
                    EMPTY NO
                    UNCLOSED NO
                ATTRIB
                    DEFAULT YES
                    OMITNAME NO
                    VALUE NO
            EMPTYNRM YES
            IMPLYDEF
                ATTLIST YES
                DOCTYPE YES
                ELEMENT YES
                ENTITY YES
                NOTATION YES
        LINK
            SIMPLE NO
            IMPLICIT NO
            EXPLICIT NO
        OTHER
            CONCUR NO
            SUBDOC NO
            FORMAL NO
            URN NO
            KEEPRSRE YES
            VALIDITY TAG
            ENTITIES
                REF ANY
                INTEGRAL YES
    APPINFO NONE
    SEEALSO "ISO 8879//
NOTATION Application Requirements for XML//EN"
>
```

Browser Timelines

(Releases important to HTML and CSS development)
= Index DOT Html/Css by Brian Wilson =

Index DOT Html: **Main Index** | **Element Tree** | **Element Index** | **HTML Support History**
Index DOT Css: **Main Index** | **Property Index** | **CSS Support History** | **Browser History**

Expand All

Windows IE

1.0	Final	Aug. 1995
2.0	Final	Nov. 1995
3.0	Final	Aug. 1996
4.0	Final	Oct. 1997
5.0	Final	Mar. 1999
5.5	Final	Jul. 2000
6.0	Final	Oct. 2001

Macintosh IE

2.0	Final	Apr. 1996
2.1	Final	Sep. 1996
3.0	Final	Jan. 1997
4.0	Final	Jan. 1998
4.5	Final	Jan. 1999
5.0	Final	Mar. 2000

Mosaic

1.0	Final	Nov 1993
2.0	Final	Nov. 1995
2.1	Final	Jan. 1996
3.0	Final	Jan. 1997

Mosaic Ends

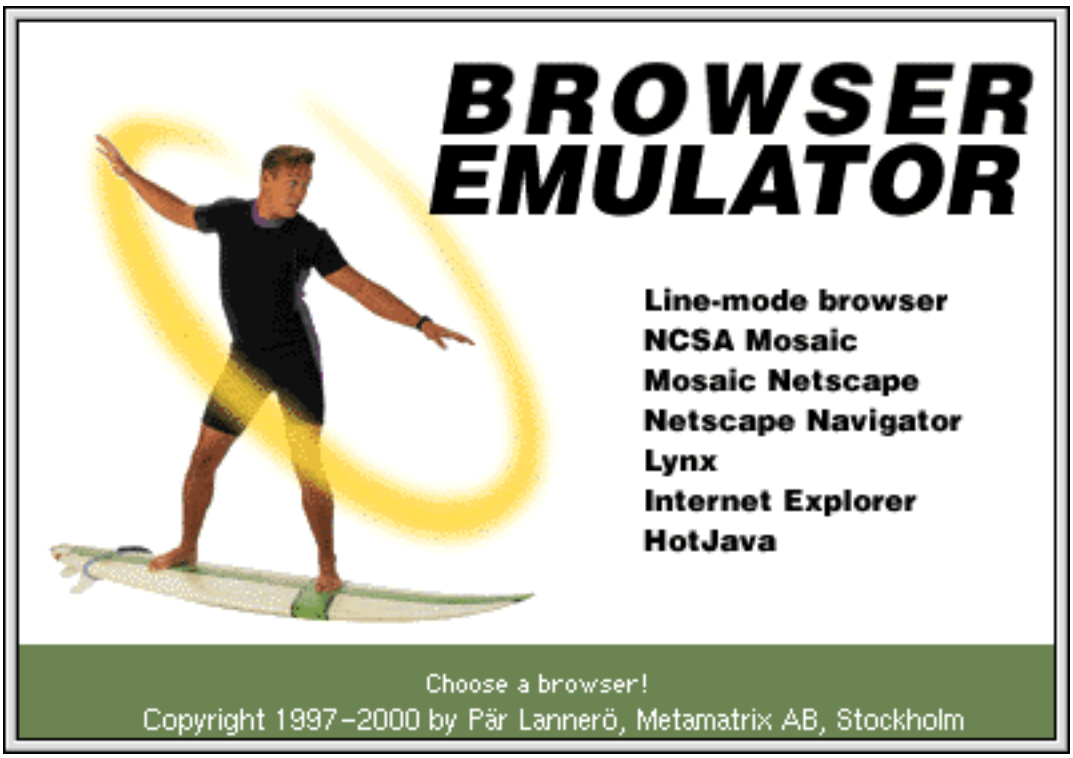
Netscape

1.0	0.9 (Beta 1)	Oct. 1994
	Final	Dec. 1994
1.1	Beta 1	Mar. 1995
	Final	Apr. 1995
1.2	Beta 1	Jun. 1995
	Final	Jul. 1995
2.0	Beta 1	Oct. 1995
	Final	Mar. 1996
3.0	Beta 1	Apr. 1996
	Beta 5	Jul. 1996
	Beta 7	Aug. 1996
	Final	Aug. 1996
4.0	Beta 1	Dec. 1996
	Beta 2	Feb. 1997
	Beta 3	Apr. 1997
	Beta 4/5	May. 1997
	Final	Jun. 1997
	4.01-08 Updates	Jun 97-Nov. 98
4.5	Beta 1	Jul. 1998
	Beta 2	Sep. 1998
	Final	Oct. 1998
	4.51 Update	Mar. 1999
	4.6 Update	May. 1999
	4.7 Update	Sep. 1999
	4.79 Update	Nov. 2001
	4.8 Update	Aug. 2002
6.0	Beta 1	Apr. 2000
	Beta 2	Aug. 2000
	Beta 3	Oct. 2000

Opera

2.1	Final	Dec. 1996
3.0	Final	Dec. 1997
3.5	Final	Nov. 1998
4.0	Final	Jun. 2000
5.0	Final	Dec. 2000
6.0	Final	Nov. 2001
7.0	Final	Jan. 2003

	Final	**Nov. 2000**
	6.01 Update	**Feb. 2001**
	6.1PR1	**Jun. 2001**
	6.1 Update	**Aug. 2001**
	6.2 Update	**Oct. 2001**
	6.2.3 Update	**May. 2002**
7.0	Beta 1	**May. 2002**
	Final	**Aug. 2002**
	7.01 Update	**Dec. 2002**
	7.02 Update	**Feb. 2003**
	7.1 Update	**Jun. 2003**



Sorry, due to heavy load on the server, browsing is quite slow.
On the positive side, it makes the experience even more authentic...

back to dejavu.org

Extensible Markup Language (XML) 1.0 (Third Edition)

W3C Recommendation 04 February 2004

This version:
http://www.w3.org/TR/2004/REC-xml-20040204
Latest version:
http://www.w3.org/TR/REC-xml
Previous version:
http://www.w3.org/TR/2003/PER-xml-20031030
Editors:
Tim Bray, Textuality and Netscape <tbray@textuality.com>
Jean Paoli, Microsoft <jeanpa@microsoft.com>
C. M. Sperberg-McQueen, W3C <cmsmcq@w3.org>
Eve Maler, Sun Microsystems, Inc. <eve.maler@east.sun.com> - Second Edition
François Yergeau <francois@yergeau.com> - Third Edition

Please refer to the **errata** for this document, which may include some normative corrections.

This document is also available in these non-normative formats: XML and XHTML with color-coded revision indicators.

See also **translations**.

Abstract

The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document is a Recommendation of the W3C. It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document specifies a syntax created by subsetting an existing, widely used international text processing standard (Standard Generalized Markup Language, ISO 8879:1986(E) as amended and corrected) for use on the World Wide Web. It is a product of the XML Core Working Group as part of the XML Activity. The English version of this specification is the only normative version. However, for translations of this document, see http://www.w3.org/2003/03/Translations/byTechnology?technology=REC-xml.

This third edition is *not* a new version of XML. As a convenience to readers, it incorporates the changes dictated by the accumulated errata (available at http://www.w3.org/XML/xml-V10-2e-errata) to the Second Edition of XML 1.0, dated 6 October 2000. In addition, markup has been introduced on a significant portion of the prescriptions of the specification, clarifying when prescriptive keywords such as MUST, SHOULD and MAY are used in the formal sense defined in [IETF RFC 2119]. For the convenience of readers, an XHTML version with color-coded revision indicators is also provided; this version highlights each change due to an erratum published in the errata list, together with a link to the particular erratum in that list. Most of the errata in the list provide a rationale for the change.

An implementation report is available at http://www.w3.org/XML/2003/09/xml10-3e-implementation.html.

Documentation of intellectual property possibly relevant to this recommendation may be found at the Working Group's public IPR disclosure page.

Please report errors in this document to xml-editor@w3.org; archives are available. The errata list for this third edition is available at http://www.w3.org/XML/xml-V10-3e-errata.

A Test Suite is maintained to help assessing conformance to this specification.

Table of Contents

Appendices

1 Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

[Definition: A software module called an **XML processor** is used to read XML documents and provide access to their content and structure.] [Definition: It is assumed that an XML processor is doing its work on behalf of another module, called the **application**.] This specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C. The membership of the XML Working Group is given in an appendix. Dan Connolly served as the Working Group's contact with the W3C.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

This specification, together with associated standards (Unicode [Unicode] and ISO/IEC 10646 [ISO/IEC 10646] for characters, Internet RFC 3066 [IETF RFC 3066] for language identification tags, ISO 639 [ISO 639] for language name codes, and ISO 3166 [ISO 3166] for country name codes), provides all the information necessary to understand XML Version 1.0 and construct computer programs to process it.

This version of the XML specification may be distributed freely, as long as all text and legal notices remain intact.

1.2 Terminology

The terminology used to describe XML documents is defined in the body of this specification. The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when EMPHASIZED, are to be interpreted as described in [IETF RFC 2119]. In addition, the terms defined in the following list are used in building those definitions and in describing the actions of an XML processor:

error

: [Definition: A violation of the rules of this specification; results are undefined. Unless otherwise specified, failure to observe a prescription of this specification indicated by one of the keywords MUST, REQUIRED, MUST NOT, SHALL and SHALL NOT is an error. Conforming software MAY detect and report an error and MAY recover from it.]

fatal error

: [Definition: An error which a conforming XML processor MUST detect and report to the application. After encountering a fatal error, the processor MAY continue processing the data to search for further errors and MAY report such errors to the application. In order to support correction of errors, the processor MAY make unprocessed data from the document (with intermingled character data and markup) available to the application. Once a fatal error is detected, however, the processor MUST NOT continue normal processing (i.e., it MUST NOT continue to pass character data and information about the document's logical structure to the application in the normal way).]

at user option

: [Definition: Conforming software MAY or MUST (depending on the modal verb in the sentence) behave as described; if it does, it MUST provide users a means to enable or disable the behavior described.]

validity constraint

: [Definition: A rule which applies to all valid XML documents. Violations of validity constraints are errors; they MUST, at user option, be reported by validating XML processors.]

well-formedness constraint

: [Definition: A rule which applies to all well-formed XML documents. Violations of well-formedness constraints are fatal errors.]

match

[Definition: (Of strings or names:) Two strings or names being compared MUST be identical. Characters with multiple possible representations in ISO/IEC 10646 (e.g. characters with both precomposed and base+diacritic forms) match only if they have the same representation in both strings. No case folding is performed. (Of strings and rules in the grammar:) A string matches a grammatical production if it belongs to the language generated by that production. (Of content and content models:) An element matches its declaration when it conforms in the fashion described in the constraint **[VC: Element Valid]**.]

for compatibility

[Definition: Marks a sentence describing a feature of XML included solely to ensure that XML remains compatible with SGML.]

for interoperability

[Definition: Marks a sentence describing a non-binding recommendation included to increase the chances that XML documents can be processed by the existing installed base of SGML processors which predate the WebSGML Adaptations Annex to ISO 8879.]

2 Documents

[Definition: A data object is an **XML document** if it is well-formed, as defined in this specification. A well-formed XML document MAY in addition be valid if it meets certain further constraints.]

Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity MAY refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The logical and physical structures MUST nest properly, as described in **4.3.2 Well-Formed Parsed Entities**.

2.1 Well-Formed XML Documents

[Definition: A textual object is a **well-formed** XML document if:]

1. Taken as a whole, it matches the production labeled document.
2. It meets all the well-formedness constraints given in this specification.
3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed.

Document

```
[1]  document  ::=  prolog element Misc*
```

Matching the document production implies that:

1. It contains one or more elements.
2. [Definition: There is exactly one element, called the **root**, or document element, no part of which appears in the content of any other element.] For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

[Definition: As a consequence of this, for each non-root element C in the document, there is one other element P in the document such that C is in the content of P, but is not in the content of any other element that is in the content of P. P is referred to as the **parent** of C, and C as a **child** of P.]

2.2 Characters

[Definition: A parsed entity contains **text**, a sequence of characters, which may represent markup or character data.] [Definition: A **character** is an atomic unit of text as specified by ISO/IEC 10646:2000 [ISO/IEC 10646]. Legal characters are tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646. The versions of these standards cited in **A.1 Normative References** were current at the time this document was prepared. New characters may be added to these standards by amendments or new editions. Consequently, XML processors MUST accept any character in the range specified for Char.]

Character Range

```
[2]  Char  ::=  #x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD]  /* any Unicode character, excluding the surrogate blocks,
                | [#x10000-#x10FFFF]                                 FFFE, and FFFF. */
```

The mechanism for encoding character code points into bit patterns MAY vary from entity to entity. All XML processors MUST accept the UTF-8 and UTF-16 encodings of Unicode 3.1 [Unicode3]; the mechanisms for signaling which of the two is in use, or for bringing other encodings into play, are discussed later, in **4.3.3 Character Encoding in Entities**.

Note:

Document authors are encouraged to avoid "compatibility characters", as defined in section 6.8 of [Unicode] (see also D21 in section 3.6 of [Unicode3]). The characters defined in the following ranges are also discouraged. They are either control characters or permanently undefined Unicode characters:

```
[#x7F-#x84], [#x86-#x9F], [#xFDD0-#xFDDF],
[#1FFFE-#x1FFFF], [#2FFFE-#x2FFFF], [#3FFFE-#x3FFFF],
[#4FFFE-#x4FFFF], [#5FFFE-#x5FFFF], [#6FFFE-#x6FFFF],
[#7FFFE-#x7FFFF], [#8FFFE-#x8FFFF], [#9FFFE-#x9FFFF],
```

```
[#AFFFE-#xAFFFF], [#BFFFE-#xBFFFF], [#CFFFE-#xCFFFF],
[#DFFFE-#xDFFFF], [#EFFFE-#xEFFFF], [#FFFFE-#xFFFFF],
[#10FFFE-#x10FFFF].
```

2.3 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.

S (white space) consists of one or more space (#x20) characters, carriage returns, line feeds, or tabs.

White Space

```
[3]  S  ::=  (#x20 | #x9 | #xD | #xA)+
```

Note:

The presence of #xD in the above production is maintained purely for backward compatibility with the First Edition. As explained in **2.11 End-of-Line Handling**, all #xD characters literally present in an XML document are either removed or replaced by #xA characters before any other processing is done. The only way to get a #xD character to match this production is to use a character reference in an entity value literal.

Characters are classified for convenience as letters, digits, or other characters. A letter consists of an alphabetic or syllabic base character or an ideographic character. Full definitions of the specific characters in each class are given in **B Character Classes**.

[Definition: A **Name** is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters.] Names beginning with the string "`xml`", or with any string which would match `(('X'|'x') ('M'|'m') ('L'|'l'))`, are reserved for standardization in this or future versions of this specification.

Note:

The Namespaces in XML Recommendation [XML Names] assigns a meaning to names containing colon characters. Therefore, authors should not use the colon in XML names except for namespace purposes, but XML processors must accept the colon as a name character.

An Nmtoken (name token) is any mixture of name characters.

Names and Tokens

```
[4]  NameChar  ::=  Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
[5]  Name      ::=  (Letter | '_' | ':') (NameChar)*
[6]  Names     ::=  Name (#x20 Name)*
[7]  Nmtoken   ::=  (NameChar)+
[8]  Nmtokens  ::=  Nmtoken (#x20 Nmtoken)*
```

Note:

The Names and Nmtokens productions are used to define the validity of tokenized attribute values after normalization (see **3.3.1 Attribute Types**).

Literal data is any quoted string not containing the quotation mark used as a delimiter for that string. Literals are used for specifying the content of internal entities (EntityValue), the values of attributes (AttValue), and external identifiers (SystemLiteral). Note that a SystemLiteral can be parsed without scanning for markup.

Literals

```
[9]   EntityValue    ::=  '"' ([^%&"] | PEReference | Reference)* '"'
                          |  "'" ([^%&'] | PEReference | Reference)* "'"
[10]  AttValue       ::=  '"' ([^<&"] | Reference)* '"'
                          |  "'" ([^<&'] | Reference)* "'"
[11]  SystemLiteral  ::=  ('"' [^"]* '"') | ("'" [^']* "'")
[12]  PubidLiteral   ::=  '"' PubidChar* '"' | "'" (PubidChar - "'")* "'"
[13]  PubidChar      ::=  #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]
```

Note:

Although the EntityValue production allows the definition of a general entity consisting of a single explicit < in the literal (e.g., `<!ENTITY mylt "<">`), it is strongly advised to avoid this practice since any reference to that entity will cause a well-formedness error.

2.4 Character Data and Markup

Text consists of intermingled character data and markup. [Definition: **Markup** takes the form of start-tags, end-tags, empty-element tags, entity

references, character references, comments, CDATA section delimiters, document type declarations, processing instructions, XML declarations, text declarations, and any white space that is at the top level of the document entity (that is, outside the document element and not inside any other markup).]

[Definition: All text that is not markup constitutes the **character data** of the document.]

The ampersand character (&) and the left angle bracket (<) MUST NOT appear in their literal form, except when used as markup delimiters, or within a comment, a processing instruction, or a CDATA section. If they are needed elsewhere, they MUST be escaped using either numeric character references or the strings "`&`" and "`<`" respectively. The right angle bracket (>) MAY be represented using the string "`>`", and MUST, for compatibility, be escaped using either "`>`" or a character reference when it appears in the string "`]]>`" in content, when that string is not marking the end of a CDATA section.

In the content of elements, character data is any string of characters which does not contain the start-delimiter of any markup and does not include the CDATA-section-close delimiter, "`]]>`". In a CDATA section, character data is any string of characters not including the CDATA-section-close delimiter, "`]]>`".

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') MAY be represented as "`&apos;`", and the double-quote character (") as "`&quot;`".

Character Data

```
[14]  CharData  ::=  [^<&]* - ([^<&]* ']]>' [^<&]*)
```

2.5 Comments

[Definition: **Comments** MAY appear anywhere in a document outside other markup; in addition, they MAY appear within the document type declaration at places allowed by the grammar. They are not part of the document's character data; an XML processor MAY, but need not, make it possible for an application to retrieve the text of comments. For compatibility, the string "`--`" (double-hyphen) MUST NOT occur within comments.] Parameter entity references MUST NOT be recognized within comments.

Comments

```
[15]  Comment  ::=  '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

An example of a comment:

```
<!-- declarations for <head> & <body> -->
```

Note that the grammar does not allow a comment ending in `--->`. The following example is *not* well-formed.

```
<!-- B+, B, or B--->
```

2.6 Processing Instructions

[Definition: **Processing instructions** (PIs) allow documents to contain instructions for applications.]

Processing Instructions

```
[16]  PI        ::=  '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17]  PITarget  ::=  Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

PIs are not part of the document's character data, but MUST be passed through to the application. The PI begins with a target (PITarget) used to identify the application to which the instruction is directed. The target names "`XML`", "`xml`", and so on are reserved for standardization in this or future versions of this specification. The XML Notation mechanism MAY be used for formal declaration of PI targets. Parameter entity references MUST NOT be recognized within processing instructions.

2.7 CDATA Sections

[Definition: **CDATA sections** MAY occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup. CDATA sections begin with the string "`<![CDATA[`" and end with the string "`]]>`":]

CDATA Sections

```
[18]  CDSect   ::=  CDStart CData CDEnd
[19]  CDStart  ::=  '<![CDATA['
[20]  CData    ::=  (Char* - (Char* ']]>'
                    Char*))
[21]  CDEnd    ::=  ']]>'
```

Within a CDATA section, only the CDEnd string is recognized as markup, so that left angle brackets and ampersands may occur in their literal

form; they need not (and cannot) be escaped using "<" and "&". CDATA sections cannot nest.

An example of a CDATA section, in which "<greeting>" and "</greeting>" are recognized as character data, not markup:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

2.8 Prolog and Document Type Declaration

[Definition: XML documents SHOULD begin with an **XML declaration** which specifies the version of XML being used.] For example, the following is a complete XML document, well-formed but not valid:

```
<?xml version="1.0"?>
<greeting>Hello, world!</greeting>
```

and so is this:

```
<greeting>Hello, world!</greeting>
```

The function of the markup in an XML document is to describe its storage and logical structure and to associate attribute name-value pairs with its logical structures. XML provides a mechanism, the document type declaration, to define constraints on the logical structure and to support the use of predefined storage units. [Definition: An XML document is **valid** if it has an associated document type declaration and if the document complies with the constraints expressed in it.]

The document type declaration MUST appear before the first element in the document.

Prolog

```
[22]  prolog       ::=  XMLDecl? Misc* (doctypedecl Misc*)?
[23]  XMLDecl      ::=  '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24]  VersionInfo  ::=  S 'version' Eq ("'" VersionNum "'" | '"' VersionNum '"')
[25]  Eq           ::=  S? '=' S?
[26]  VersionNum   ::=  '1.0'
[27]  Misc         ::=  Comment | PI | S
```

[Definition: The XML **document type declaration** contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or **DTD**. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. The DTD for a document consists of both subsets taken together.]

[Definition: A **markup declaration** is an element type declaration, an attribute-list declaration, an entity declaration, or a notation declaration.] These declarations MAY be contained in whole or in part within parameter entities, as described in the well-formedness and validity constraints below. For further information, see **4 Physical Structures**.

Document Type Definition

```
[28]   doctypedecl  ::=  '<!DOCTYPE' S Name (S ExternalID)? S? ('[' intSubset ']'
                         S?)? '>'                                  [VC: Root Element Type]
                                                                   [WFC: External Subset]
[28a]  DeclSep      ::=  PEReference | S                           [WFC: PE Between Declarations]
[28b]  intSubset    ::=  (markupdecl | DeclSep)*
[29]   markupdecl   ::=  elementdecl | AttlistDecl | EntityDecl | NotationDecl |
                         PI | Comment                             [VC: Proper Declaration/PE Nesting]
                                                                   [WFC: PEs in Internal Subset]
```

Note that it is possible to construct a well-formed document containing a doctypedecl that neither points to an external subset nor contains an internal subset.

The markup declarations MAY be made up in whole or in part of the replacement text of parameter entities. The productions later in this specification for individual nonterminals (elementdecl, AttlistDecl, and so on) describe the declarations *after* all the parameter entities have been included.

Parameter entity references are recognized anywhere in the DTD (internal and external subsets and external parameter entities), except in literals, processing instructions, comments, and the contents of ignored conditional sections (see **3.4 Conditional Sections**). They are also recognized in entity value literals. The use of parameter entities in the internal subset is restricted as described below.

Validity constraint: Root Element Type

The Name in the document type declaration MUST match the element type of the root element.

Validity constraint: Proper Declaration/PE Nesting

Parameter-entity replacement text MUST be properly nested with markup declarations. That is to say, if either the first character or the last character of a markup declaration (markupdecl above) is contained in the replacement text for a parameter-entity reference, both MUST be contained in the same replacement text.

Well-formedness constraint: PEs in Internal Subset

In the internal DTD subset, parameter-entity references MUST NOT occur within markup declarations; they MAY occur where markup declarations can occur. (This does not apply to references that occur in external parameter entities or to the external subset.)

Well-formedness constraint: External Subset

The external subset, if any, MUST match the production for extSubset.

Well-formedness constraint: PE Between Declarations

The replacement text of a parameter entity reference in a DeclSep MUST match the production extSubsetDecl.

Like the internal subset, the external subset and any external parameter entities referenced in a DeclSep MUST consist of a series of complete markup declarations of the types allowed by the non-terminal symbol markupdecl, interspersed with white space or parameter-entity references. However, portions of the contents of the external subset or of these external parameter entities MAY conditionally be ignored by using the conditional section construct; this is not allowed in the internal subset but is allowed in external parameter entities referenced in the internal subset.

External Subset

```
[30]  extSubset      ::=  TextDecl? extSubsetDecl
[31]  extSubsetDecl  ::=  ( markupdecl | conditionalSect | DeclSep)*
```

The external subset and external parameter entities also differ from the internal subset in that in them, parameter-entity references are permitted *within* markup declarations, not only *between* markup declarations.

An example of an XML document with a document type declaration:

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

The system identifier "`hello.dtd`" gives the address (a URI reference) of a DTD for the document.

The declarations can also be given locally, as in this example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

If both the external and internal subsets are used, the internal subset MUST be considered to occur before the external subset. This has the effect that entity and attribute-list declarations in the internal subset take precedence over those in the external subset.

2.9 Standalone Document Declaration

Markup declarations can affect the content of the document, as passed from an XML processor to an application; examples are attribute defaults and entity declarations. The standalone document declaration, which MAY appear as a component of the XML declaration, signals whether or not there are such declarations which appear external to the document entity or in parameter entities. [Definition: An **external markup declaration** is defined as a markup declaration occurring in the external subset or in a parameter entity (external or internal, the latter being included because non-validating processors are not required to read them).]

Standalone Document Declaration

```
[32]  SDDecl  ::=  S 'standalone' Eq (("'" ('yes' | 'no') "'") | ('"' ('yes' |    [VC: Standalone Document Declaration]
                   'no') '"'))
```

In a standalone document declaration, the value "yes" indicates that there are no external markup declarations which affect the information passed from the XML processor to the application. The value "no" indicates that there are or may be such external markup declarations. Note that the standalone document declaration only denotes the presence of external *declarations*; the presence, in a document, of references to external *entities*, when those entities are internally declared, does not change its standalone status.

If there are no external markup declarations, the standalone document declaration has no meaning. If there are external markup declarations but there is no standalone document declaration, the value "no" is assumed.

Any XML document for which `standalone="no"` holds can be converted algorithmically to a standalone document, which may be desirable for some network delivery applications.

Validity constraint: Standalone Document Declaration

The standalone document declaration MUST have the value "no" if any external markup declarations contain declarations of:

- attributes with default values, if elements to which these attributes apply appear in the document without specifications of values for these attributes, or
- entities (other than `amp`, `lt`, `gt`, `apos`, `quot`), if references to those entities appear in the document, or
- attributes with tokenized types, where the attribute appears in the document with a value such that *normalization* will produce a different value from that which would be produced in the absence of the declaration, or
- element types with element content, if white space occurs directly within any instance of those types.

An example XML declaration with a standalone document declaration:

```
<?xml version="1.0" standalone='yes'?>
```

2.10 White Space Handling

In editing XML documents, it is often convenient to use "white space" (spaces, tabs, and blank lines) to set apart the markup for greater readability. Such white space is typically not intended for inclusion in the delivered version of the document. On the other hand, "significant" white space that should be preserved in the delivered version is common, for example in poetry and source code.

An XML processor MUST always pass all characters in a document that are not markup through to the application. A validating XML processor MUST also inform the application which of these characters constitute white space appearing in element content.

A special attribute named `xml:space` MAY be attached to an element to signal an intention that in that element, white space should be preserved by applications. In valid documents, this attribute, like any other, MUST be declared if it is used. When declared, it MUST be given as an enumerated type whose values are one or both of "default" and "preserve". For example:

```
<!ATTLIST poem  xml:space (default|preserve) 'preserve'>

<!ATTLIST pre xml:space (preserve) #FIXED 'preserve'>
```

The value "default" signals that applications' default white-space processing modes are acceptable for this element; the value "preserve" indicates the intent that applications preserve all the white space. This declared intent is considered to apply to all elements within the content of the element where it is specified, unless overridden with another instance of the `xml:space` attribute. This specification does not give meaning to any value of `xml:space` other than "default" and "preserve". It is an error for other values to be specified; the XML processor MAY report the error or MAY recover by ignoring the attribute specification or by reporting the (erroneous) value to the application. Applications may ignore or reject erroneous values.

The root element of any document is considered to have signaled no intentions as regards application space handling, unless it provides a value for this attribute or the attribute is declared with a default value.

2.11 End-of-Line Handling

XML parsed entities are often stored in computer files which, for editing convenience, are organized into lines. These lines are typically separated by some combination of the characters CARRIAGE RETURN (#xD) and LINE FEED (#xA).

To simplify the tasks of applications, the XML processor MUST behave as if it normalized all line breaks in external parsed entities (including the document entity) on input, before parsing, by translating both the two-character sequence #xD #xA and any #xD that is not followed by #xA to a single #xA character.

2.12 Language Identification

In document processing, it is often useful to identify the natural or formal language in which the content is written. A special attribute named `xml:lang` MAY be inserted in documents to specify the language used in the contents and attribute values of any element in an XML document. In valid documents, this attribute, like any other, MUST be declared if it is used. The values of the attribute are language identifiers as defined by [IETF RFC 3066], *Tags for the Identification of Languages*, or its successor; in addition, the empty string MAY be specified.

(Productions 33 through 38 have been removed.)

For example:

```
<p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
  <l>Habe nun, ach! Philosophie,</l>
  <l>Juristerei, und Medizin</l>
  <l>und leider auch Theologie</l>
  <l>durchaus studiert mit heißem Bemüh'n.</l>
```

```
</sp>
```

The intent declared with `xml:lang` is considered to apply to all attributes and content of the element where it is specified, unless overridden with an instance of `xml:lang` on another element within that content. In particular, the empty value of `xml:lang` is used on an element B to override a specification of `xml:lang` on an enclosing element A, without specifying another language. Within B, it is considered that there is no language information available, just as if `xml:lang` had not been specified on B or any of its ancestors.

Note:

Language information may also be provided by external transport protocols (e.g. HTTP or MIME). When available, this information may be used by XML applications, but the more local information provided by `xml:lang` should be considered to override it.

A simple declaration for `xml:lang` might take the form

```
xml:lang CDATA #IMPLIED
```

but specific default values MAY also be given, if appropriate. In a collection of French poems for English students, with glosses and notes in English, the `xml:lang` attribute might be declared this way:

```
<!ATTLIST poem   xml:lang CDATA 'fr'>
<!ATTLIST gloss  xml:lang CDATA 'en'>
<!ATTLIST note   xml:lang CDATA 'en'>
```

3 Logical Structures

[Definition: Each XML document contains one or more **elements**, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, sometimes called its "generic identifier" (GI), and MAY have a set of attribute specifications.] Each attribute specification has a name and a value.

Element

```
[39]  element  ::=  EmptyElemTag
                  | STag content ETag [WFC: Element Type Match]
                                      [VC: Element Valid]
```

This specification does not constrain the semantics, use, or (beyond syntax) names of the element types and attributes, except that names beginning with a match to `(('X'|'x')('M'|'m')('L'|'l'))` are reserved for standardization in this or future versions of this specification.

Well-formedness constraint: Element Type Match

The Name in an element's end-tag MUST match the element type in the start-tag.

Validity constraint: Element Valid

An element is valid if there is a declaration matching elementdecl where the Name matches the element type, and one of the following holds:

1. The declaration matches **EMPTY** and the element has no content (not even entity references, comments, PIs or white space).
2. The declaration matches children and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space, comments and PIs (i.e. markup matching production [27] Misc) between the start-tag and the first child element, between child elements, or between the last child element and the end-tag. Note that a CDATA section containing only white space or a reference to an entity whose replacement text is character references expanding to white space do not match the nonterminal S, and hence cannot appear in these positions; however, a reference to an internal entity with a literal value consisting of character references expanding to white space does match S, since its replacement text is the white space resulting from expansion of the character references.
3. The declaration matches Mixed and the content (after replacing any entity references with their replacement text) consists of character data, comments, PIs and child elements whose types match names in the content model.
4. The declaration matches **ANY**, and the content (after replacing any entity references with their replacement text) consists of character data and child elements whose types have been declared.

3.1 Start-Tags, End-Tags, and Empty-Element Tags

[Definition: The beginning of every non-empty XML element is marked by a **start-tag**.]

Start-tag

```
[40]  STag       ::=  '<' Name (S Attribute)* S? '>' [WFC: Unique Att Spec]
[41]  Attribute  ::=  Name Eq AttValue               [VC: Attribute Value Type]
                                                     [WFC: No External Entity References]
                                                     [WFC: No < in Attribute Values]
```

The Name in the start- and end-tags gives the element's **type**. [Definition: The Name-AttValue pairs are referred to as the **attribute specifications** of the element], [Definition: with the Name in each pair referred to as the **attribute name**] and [Definition: the content of the AttValue (the text between the ' or " delimiters) as the **attribute value**.] Note that the order of attribute specifications in a start-tag or empty-element tag is not significant.

Well-formedness constraint: Unique Att Spec

An attribute name MUST NOT appear more than once in the same start-tag or empty-element tag.

Validity constraint: Attribute Value Type

The attribute MUST have been declared; the value MUST be of the type declared for it. (For attribute types, see **3.3 Attribute-List Declarations**.)

Well-formedness constraint: No External Entity References

Attribute values MUST NOT contain direct or indirect entity references to external entities.

Well-formedness constraint: No < in Attribute Values

The replacement text of any entity referred to directly or indirectly in an attribute value MUST NOT contain a <.

An example of a start-tag:

```
<termdef id="dt-dog" term="dog">
```

[Definition: The end of every element that begins with a start-tag MUST be marked by an **end-tag** containing a name that echoes the element's type as given in the start-tag:]

End-tag

```
[42]  ETag  ::=  '</' Name S? '>'
```

An example of an end-tag:

```
</termdef>
```

[Definition: The text between the start-tag and end-tag is called the element's **content**:]

Content of Elements

```
[43]  content  ::=  CharData? ((element | Reference | CDSect | PI | Comment) CharData?)*
```

[Definition: An element with no content is said to be **empty**.] The representation of an empty element is either a start-tag immediately followed by an end-tag, or an empty-element tag. [Definition: An **empty-element tag** takes a special form:]

Tags for Empty Elements

```
[44]  EmptyElemTag  ::=  '<' Name (S Attribute)* S? '/>' [WFC: Unique Att Spec]
```

Empty-element tags MAY be used for any element which has no content, whether or not it is declared using the keyword **EMPTY**. For interoperability, the empty-element tag SHOULD be used, and SHOULD only be used, for elements which are declared EMPTY.

Examples of empty elements:

```
<IMG align="left"
 src="http://www.w3.org/Icons/WWW/w3c_home" />
<br></br>
<br/>
```

3.2 Element Type Declarations

The element structure of an XML document MAY, for validation purposes, be constrained using element type and attribute-list declarations. An element type declaration constrains the element's content.

Element type declarations often constrain which element types can appear as children of the element. At user option, an XML processor MAY issue a warning when a declaration mentions an element type for which no declaration is provided, but this is not an error.

[Definition: An **element type declaration** takes the form:]

Element Type Declaration

```
[45]  elementdecl  ::=  '<!ELEMENT' S Name S contentspec S? '>' [VC: Unique Element Type Declaration]
[46]  contentspec  ::=  'EMPTY' | 'ANY' | Mixed | children
```

where the Name gives the element type being declared.

Validity constraint: Unique Element Type Declaration

An element type MUST NOT be declared more than once.

Examples of element type declarations:

```
<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA|emph)* >
<!ELEMENT %name.para; %content.para; >
<!ELEMENT container ANY>
```

3.2.1 Element Content

[Definition: An element type has **element content** when elements of that type MUST contain only child elements (no character data), optionally separated by white space (characters matching the nonterminal S).] [Definition: In this case, the constraint includes a **content model**, a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear.] The grammar is built on content particles (cps), which consist of names, choice lists of content particles, or sequence lists of content particles:

Element-content Models

```
[47]  children  ::=  (choice | seq) ('?' | '*' | '+')?
[48]  cp        ::=  (Name | choice | seq) ('?' | '*' | '+')?
[49]  choice    ::=  '(' S? cp ( S? '|' S? cp )+ S? ')'       [VC: Proper Group/PE Nesting]
[50]  seq       ::=  '(' S? cp ( S? ',' S? cp )* S? ')'       [VC: Proper Group/PE Nesting]
```

where each Name is the type of an element which MAY appear as a child. Any content particle in a choice list MAY appear in the element content at the location where the choice list appears in the grammar; content particles occurring in a sequence list MUST each appear in the element content in the order given in the list. The optional character following a name or list governs whether the element or the content particles in the list may occur one or more (+), zero or more (*), or zero or one times (?). The absence of such an operator means that the element or content particle MUST appear exactly once. This syntax and meaning are identical to those used in the productions in this specification.

The content of an element matches a content model if and only if it is possible to trace out a path through the content model, obeying the sequence, choice, and repetition operators and matching each element in the content against an element type in the content model. For compatibility, it is an error if the content model allows an element to match more than one occurrence of an element type in the content model. For more information, see **E Deterministic Content Models**.

Validity constraint: Proper Group/PE Nesting

Parameter-entity replacement text MUST be properly nested with parenthesized groups. That is to say, if either of the opening or closing parentheses in a choice, seq, or Mixed construct is contained in the replacement text for a parameter entity, both MUST be contained in the same replacement text.

For interoperability, if a parameter-entity reference appears in a choice, seq, or Mixed construct, its replacement text SHOULD contain at least one non-blank character, and neither the first nor last non-blank character of the replacement text SHOULD be a connector (| or ,).

Examples of element-content models:

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
<!ELEMENT dictionary-body (%div.mix; | %dict.mix;)*>
```

3.2.2 Mixed Content

[Definition: An element type has **mixed content** when elements of that type MAY contain character data, optionally interspersed with child elements.] In this case, the types of the child elements MAY be constrained, but not their order or their number of occurrences:

Mixed-content Declaration

```
[51]  Mixed  ::=  '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*'
                | '(' S? '#PCDATA' S? ')'                    [VC: Proper Group/PE Nesting]
                                                             [VC: No Duplicate Types]
```

where the Names give the types of elements that may appear as children. The keyword **#PCDATA** derives historically from the term "parsed

character data."

Validity constraint: No Duplicate Types

The same name MUST NOT appear more than once in a single mixed-content declaration.

Examples of mixed content declarations:

```
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >
<!ELEMENT b (#PCDATA)>
```

3.3 Attribute-List Declarations

Attributes are used to associate name-value pairs with elements. Attribute specifications MUST NOT appear outside of start-tags and empty-element tags; thus, the productions used to recognize them appear in **3.1 Start-Tags, End-Tags, and Empty-Element Tags**. Attribute-list declarations MAY be used:

- To define the set of attributes pertaining to a given element type.
- To establish type constraints for these attributes.
- To provide default values for attributes.

[Definition: **Attribute-list declarations** specify the name, data type, and default value (if any) of each attribute associated with a given element type:]

Attribute-list Declaration

```
[52]  AttlistDecl  ::=  '<!ATTLIST' S Name AttDef* S? '>'
[53]  AttDef       ::=  S Name S AttType S DefaultDecl
```

The Name in the AttlistDecl rule is the type of an element. At user option, an XML processor MAY issue a warning if attributes are declared for an element type not itself declared, but this is not an error. The Name in the AttDef rule is the name of the attribute.

When more than one AttlistDecl is provided for a given element type, the contents of all those provided are merged. When more than one definition is provided for the same attribute of a given element type, the first declaration is binding and later declarations are ignored. For interoperability, writers of DTDs MAY choose to provide at most one attribute-list declaration for a given element type, at most one attribute definition for a given attribute name in an attribute-list declaration, and at least one attribute definition in each attribute-list declaration. For interoperability, an XML processor MAY at user option issue a warning when more than one attribute-list declaration is provided for a given element type, or more than one attribute definition is provided for a given attribute, but this is not an error.

3.3.1 Attribute Types

XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The string type may take any literal string as a value; the tokenized types have varying lexical and semantic constraints. The validity constraints noted in the grammar are applied after the attribute value has been normalized as described in **3.3.3 Attribute-Value Normalization**.

Attribute Types

```
[54]  AttType        ::=  StringType | TokenizedType | EnumeratedType
[55]  StringType     ::=  'CDATA'
[56]  TokenizedType  ::=  'ID'                          [VC: ID]
                                                        [VC: One ID per Element Type]
                                                        [VC: ID Attribute Default]
                          | 'IDREF'                     [VC: IDREF]
                          | 'IDREFS'                    [VC: IDREF]
                          | 'ENTITY'                    [VC: Entity Name]
                          | 'ENTITIES'                  [VC: Entity Name]
                          | 'NMTOKEN'                   [VC: Name Token]
                          | 'NMTOKENS'                  [VC: Name Token]
```

Validity constraint: ID

Values of type **ID** MUST match the Name production. A name MUST NOT appear more than once in an XML document as a value of this type; i.e., ID values MUST uniquely identify the elements which bear them.

Validity constraint: One ID per Element Type

An element type MUST NOT have more than one ID attribute specified.

Validity constraint: ID Attribute Default

An ID attribute MUST have a declared default of **#IMPLIED** or **#REQUIRED**.

Validity constraint: IDREF

Values of type **IDREF** MUST match the Name production, and values of type **IDREFS** MUST match Names; each Name MUST match the value of an ID attribute on some element in the XML document; i.e. **IDREF** values MUST match the value of some ID attribute.

Validity constraint: Entity Name

Values of type **ENTITY** MUST match the Name production, values of type **ENTITIES** MUST match Names; each Name MUST match the name of an unparsed entity declared in the DTD.

Validity constraint: Name Token

Values of type **NMTOKEN** MUST match the Nmtoken production; values of type **NMTOKENS** MUST match Nmtokens.

[Definition: **Enumerated attributes** MUST take one of a list of values provided in the declaration]. There are two kinds of enumerated types:

Enumerated Attribute Types

```
[57]  EnumeratedType  ::=  NotationType | Enumeration
[58]  NotationType    ::=  'NOTATION' S '(' S? Name (S? '|' S? Name)* S?
                           ')'                          [VC: Notation Attributes]
                                                        [VC: One Notation Per Element Type]
                                                        [VC: No Notation on Empty Element]
                                                        [VC: No Duplicate Tokens]
[59]  Enumeration     ::=  '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'
                                                        [VC: Enumeration]
                                                        [VC: No Duplicate Tokens]
```

A **NOTATION** attribute identifies a notation, declared in the DTD with associated system and/or public identifiers, to be used in interpreting the element to which the attribute is attached.

Validity constraint: Notation Attributes

Values of this type MUST match one of the *notation* names included in the declaration; all notation names in the declaration MUST be declared.

Validity constraint: One Notation Per Element Type

An element type MUST NOT have more than one **NOTATION** attribute specified.

Validity constraint: No Notation on Empty Element

For compatibility, an attribute of type **NOTATION** MUST NOT be declared on an element declared **EMPTY**.

Validity constraint: No Duplicate Tokens

The notation names in a single NotationType attribute declaration, as well as the NmTokens in a single Enumeration attribute declaration, MUST all be distinct.

Validity constraint: Enumeration

Values of this type MUST match one of the Nmtoken tokens in the declaration.

For interoperability, the same Nmtoken SHOULD NOT occur more than once in the enumerated attribute types of a single element type.

3.3.2 Attribute Defaults

An attribute declaration provides information on whether the attribute's presence is REQUIRED, and if not, how an XML processor is to react if a declared attribute is absent in a document.

Attribute Defaults

```
[60]  DefaultDecl  ::=  '#REQUIRED' | '#IMPLIED'
                        | (('#FIXED' S)? AttValue)  [VC: Required Attribute]
                                                    [VC: Attribute Default Value Syntactically Correct]
                                                    [WFC: No < in Attribute Values]
                                                    [VC: Fixed Attribute Default]
```

In an attribute declaration, **#REQUIRED** means that the attribute MUST always be provided, **#IMPLIED** that no default value is provided. [Definition: If the declaration is neither **#REQUIRED** nor **#IMPLIED**, then the AttValue value contains the declared **default** value; the **#FIXED**

keyword states that the attribute MUST always have the default value. When an XML processor encounters an element without a specification for an attribute for which it has read a default value declaration, it MUST report the attribute with the declared default value to the application.]

Validity constraint: Required Attribute

If the default declaration is the keyword **#REQUIRED**, then the attribute MUST be specified for all elements of the type in the attribute-list declaration.

Validity constraint: Attribute Default Value Syntactically Correct

The declared default value MUST meet the syntactic constraints of the declared attribute type.

Note that only the syntactic constraints of the type are required here; other constraints (e.g. that the value be the name of a declared unparsed entity, for an attribute of type ENTITY) may come into play if the declared default value is actually used (an element without a specification for this attribute occurs).

Validity constraint: Fixed Attribute Default

If an attribute has a default value declared with the **#FIXED** keyword, instances of that attribute MUST match the default value.

Examples of attribute-list declarations:

```
<!ATTLIST termdef
          id      ID      #REQUIRED
          name    CDATA   #IMPLIED>
<!ATTLIST list
          type    (bullets|ordered|glossary)  "ordered">
<!ATTLIST form
          method  CDATA   #FIXED "POST">
```

3.3.3 Attribute-Value Normalization

Before the value of an attribute is passed to the application or checked for validity, the XML processor MUST normalize the attribute value by applying the algorithm below, or by using some other method such that the value passed to the application is the same as that produced by the algorithm.

1. All line breaks MUST have been normalized on input to #xA as described in **2.11 End-of-Line Handling**, so the rest of this algorithm operates on text normalized in this way.
2. Begin with a normalized value consisting of the empty string.
3. For each character, entity reference, or character reference in the unnormalized attribute value, beginning with the first and continuing to the last, do the following:
   - For a character reference, append the referenced character to the normalized value.
   - For an entity reference, recursively apply step 3 of this algorithm to the replacement text of the entity.
   - For a white space character (#x20, #xD, #xA, #x9), append a space character (#x20) to the normalized value.
   - For another character, append the character to the normalized value.

If the attribute type is not CDATA, then the XML processor MUST further process the normalized attribute value by discarding any leading and trailing space (#x20) characters, and by replacing sequences of space (#x20) characters by a single space (#x20) character.

Note that if the unnormalized attribute value contains a character reference to a white space character other than space (#x20), the normalized value contains the referenced character itself (#xD, #xA or #x9). This contrasts with the case where the unnormalized value contains a white space character (not a reference), which is replaced with a space character (#x20) in the normalized value and also contrasts with the case where the unnormalized value contains an entity reference whose replacement text contains a white space character; being recursively processed, the white space character is replaced with a space character (#x20) in the normalized value.

All attributes for which no declaration has been read SHOULD be treated by a non-validating processor as if declared **CDATA**.

It is an error if an attribute value contains a reference to an entity for which no declaration has been read.

Following are examples of attribute normalization. Given the following declarations:

```
<!ENTITY d "&#xD;">
<!ENTITY a "&#xA;">
<!ENTITY da "&#xD;&#xA;">
```

the attribute specifications in the left column below would be normalized to the character sequences of the middle column if the attribute a is declared **NMTOKENS** and to those of the right columns if a is declared **CDATA**.

Attribute specification	a is NMTOKENS	a is CDATA
a=" xyz"	x y z	#x20 #x20 x y z

a="&d;&d;A&a; &a;B&da;"	A #x20 B	#x20 #x20 A #x20 #x20 #x20 B #x20 #x20
a= "

A

B
" | #xD #xD A #xA #xA B #xD #xA | #xD #xD A #xA #xA B #xD #xA |

Note that the last example is invalid (but well-formed) if a is declared to be of type **NMTOKENS**.

3.4 Conditional Sections

[Definition: **Conditional sections** are portions of the document type declaration external subset or of external parameter entities which are included in, or excluded from, the logical structure of the DTD based on the keyword which governs them.]

Conditional Section

[61]	conditionalSect	::=	includeSect | ignoreSect	
[62]	includeSect	::=	'<![' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'	[VC: Proper Conditional Section/PE Nesting]
[63]	ignoreSect	::=	'<![' S? 'IGNORE' S? '[' ignoreSectContents* ']]>'	[VC: Proper Conditional Section/PE Nesting]
[64]	ignoreSectContents	::=	Ignore ('<![' ignoreSectContents ']]>' Ignore)*	
[65]	Ignore	::=	Char* - (Char* ('<![' | ']]>') Char*)	

Validity constraint: Proper Conditional Section/PE Nesting

If any of the "<![", "[", or "]]>" of a conditional section is contained in the replacement text for a parameter-entity reference, all of them MUST be contained in the same replacement text.

Like the internal and external DTD subsets, a conditional section may contain one or more complete declarations, comments, processing instructions, or nested conditional sections, intermingled with white space.

If the keyword of the conditional section is **INCLUDE**, then the contents of the conditional section MUST be considered part of the DTD. If the keyword of the conditional section is **IGNORE**, then the contents of the conditional section MUST be considered as not logically part of the DTD. If a conditional section with a keyword of **INCLUDE** occurs within a larger conditional section with a keyword of **IGNORE**, both the outer and the inner conditional sections MUST be ignored. The contents of an ignored conditional section MUST be parsed by ignoring all characters after the "[" following the keyword, except conditional section starts "<![" and ends "]]>", until the matching conditional section end is found. Parameter entity references MUST NOT be recognized in this process.

If the keyword of the conditional section is a parameter-entity reference, the parameter entity MUST be replaced by its content before the processor decides whether to include or ignore the conditional section.

An example:

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >

<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

4 Physical Structures

[Definition: An XML document may consist of one or many storage units. These are called **entities**; they all have **content** and are all (except for the document entity and the external DTD subset) identified by entity **name**.] Each XML document has one entity called the document entity, which serves as the starting point for the XML processor and may contain the whole document.

Entities may be either parsed or unparsed. [Definition: The contents of a **parsed entity** are referred to as its replacement text; this text is considered an integral part of the document.]

[Definition: An **unparsed entity** is a resource whose contents may or may not be text, and if text, may be other than XML. Each unparsed entity has an associated notation, identified by name. Beyond a requirement that an XML processor make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.]

Parsed entities are invoked by name using entity references; unparsed entities by name, given in the value of **ENTITY** or **ENTITIES** attributes.

[Definition: **General entities** are entities for use within the document content. In this specification, general entities are sometimes referred to with the unqualified term *entity* when this leads to no ambiguity.] [Definition: **Parameter entities** are parsed entities for use within the DTD.] These two types of entities use different forms of reference and are recognized in different contexts. Furthermore, they occupy different namespaces; a parameter entity and a general entity with the same name are two distinct entities.

4.1 Character and Entity References

[Definition: A **character reference** refers to a specific character in the ISO/IEC 10646 character set, for example one not directly accessible from available input devices.]

Character Reference

```
[66]  CharRef  ::=  '&#' [0-9]+ ';'
                  | '&#x' [0-9a-fA-F]+ ';' [WFC: Legal Character]
```

Well-formedness constraint: Legal Character

Characters referred to using character references MUST match the production for Char.

If the character reference begins with "&#x", the digits and letters up to the terminating ; provide a hexadecimal representation of the character's code point in ISO/IEC 10646. If it begins just with "&#", the digits up to the terminating ; provide a decimal representation of the character's code point.

[Definition: An **entity reference** refers to the content of a named entity.] [Definition: References to parsed general entities use ampersand (&) and semicolon (;) as delimiters.] [Definition: **Parameter-entity references** use percent-sign (%) and semicolon (;) as delimiters.]

Entity Reference

```
[67]  Reference     ::=  EntityRef | CharRef
[68]  EntityRef     ::=  '&' Name ';'        [WFC: Entity Declared]
                                             [VC: Entity Declared]
                                             [WFC: Parsed Entity]
                                             [WFC: No Recursion]
[69]  PEReference   ::=  '%' Name ';'        [VC: Entity Declared]
                                             [WFC: No Recursion]
                                             [WFC: In DTD]
```

Well-formedness constraint: Entity Declared

In a document without any DTD, a document with only an internal DTD subset which contains no parameter entity references, or a document with "`standalone='yes'`", for an entity reference that does not occur within the external subset or a parameter entity, the Name given in the entity reference MUST match that in an *entity declaration* that does not occur within the external subset or a parameter entity, except that well-formed documents need not declare any of the following entities: `amp`, `lt`, `gt`, `apos`, `quot`. The declaration of a general entity MUST precede any reference to it which appears in a default value in an attribute-list declaration.

Note that non-validating processors are *not obligated to* to read and process entity declarations occurring in parameter entities or in the external subset; for such documents, the rule that an entity must be declared is a well-formedness constraint only if *standalone='yes'*.

Validity constraint: Entity Declared

In a document with an external subset or external parameter entities with "`standalone='no'`", the Name given in the entity reference MUST match that in an *entity declaration*. For interoperability, valid documents SHOULD declare the entities `amp`, `lt`, `gt`, `apos`, `quot`, in the form specified in **4.6 Predefined Entities**. The declaration of a parameter entity MUST precede any reference to it. Similarly, the declaration of a general entity MUST precede any attribute-list declaration containing a default value with a direct or indirect reference to that general entity.

Well-formedness constraint: Parsed Entity

An entity reference MUST NOT contain the name of an unparsed entity. Unparsed entities may be referred to only in attribute values declared to be of type **ENTITY** or **ENTITIES**.

Well-formedness constraint: No Recursion

A parsed entity MUST NOT contain a recursive reference to itself, either directly or indirectly.

Well-formedness constraint: In DTD

Parameter-entity references MUST NOT appear outside the DTD.

Examples of character and entity references:

```
Type <key>less-than</key> (&#x3C;) to save options.
This document was prepared on &docdate; and
is classified &security-level;.
```

Example of a parameter-entity reference:

```
<!-- declare the parameter entity "ISOLat2"... -->
<!ENTITY % ISOLat2
         SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ... now reference it. -->
%ISOLat2;
```

4.2 Entity Declarations

[Definition: Entities are declared thus:]

Entity Declaration

```
[70]  EntityDecl  ::=  GEDecl | PEDecl
[71]  GEDecl      ::=  '<!ENTITY' S Name S EntityDef S? '>'
[72]  PEDecl      ::=  '<!ENTITY' S '%' S Name S PEDef S? '>'
[73]  EntityDef   ::=  EntityValue| (ExternalID NDataDecl?)
[74]  PEDef       ::=  EntityValue | ExternalID
```

The Name identifies the entity in an entity reference or, in the case of an unparsed entity, in the value of an **ENTITY** or **ENTITIES** attribute. If the same entity is declared more than once, the first declaration encountered is binding; at user option, an XML processor MAY issue a warning if entities are declared multiple times.

4.2.1 Internal Entities

[Definition: If the entity definition is an EntityValue, the defined entity is called an **internal entity**. There is no separate physical storage object, and the content of the entity is given in the declaration.] Note that some processing of entity and character references in the literal entity value may be required to produce the correct replacement text: see **4.5 Construction of Internal Entity Replacement Text**.

An internal entity is a parsed entity.

Example of an internal entity declaration:

```
<!ENTITY Pub-Status "This is a pre-release of the
 specification.">
```

4.2.2 External Entities

[Definition: If the entity is not internal, it is an **external entity**, declared as follows:]

External Entity Declaration

```
[75]  ExternalID  ::=  'SYSTEM' S SystemLiteral
                     | 'PUBLIC' S PubidLiteral S SystemLiteral
[76]  NDataDecl   ::=  S 'NDATA' S Name                        [VC: Notation Declared]
```

If the NDataDecl is present, this is a general unparsed entity; otherwise it is a parsed entity.

Validity constraint: Notation Declared

The Name MUST match the declared name of a notation.

[Definition: The SystemLiteral is called the entity's **system identifier**. It is meant to be converted to a URI reference (as defined in [IETF RFC 2396], updated by [IETF RFC 2732]), as part of the process of dereferencing it to obtain input for the XML processor to construct the entity's replacement text.] It is an error for a fragment identifier (beginning with a # character) to be part of a system identifier. Unless otherwise provided by information outside the scope of this specification (e.g. a special XML element type defined by a particular DTD, or a processing instruction defined by a particular application specification), relative URIs are relative to the location of the resource within which the entity declaration occurs. This is defined to be the external entity containing the '<' which starts the declaration, at the point when it is parsed as a declaration. A URI might thus be relative to the document entity, to the entity containing the external DTD subset, or to some other external parameter entity. Attempts to retrieve the resource identified by a URI MAY be redirected at the parser level (for example, in an entity resolver) or below (at the protocol level, for example, via an HTTP `Location:` header). In the absence of additional information outside the scope of this specification within the resource, the base URI of a resource is always the URI of the actual resource returned. In other words, it is the URI of the resource retrieved after all redirection has occurred.

System identifiers (and other XML strings meant to be used as URI references) MAY contain characters that, according to [IETF RFC 2396] and [IETF RFC 2732], must be escaped before a URI can be used to retrieve the referenced resource. The characters to be escaped are the control characters #x0 to #x1F and #x7F (most of which cannot appear in XML), space #x20, the delimiters '<' #x3C, '>' #x3E and "" #x22, the *unwise* characters '{' #x7B, '}' #x7D, '|' #x7C, '\' #x5C, '^' #x5E and '`' #x60, as well as all characters above #x7F. Since escaping is not always a fully reversible process, it MUST be performed only when absolutely necessary and as late as possible in a processing chain. In particular, neither the process of converting a relative URI to an absolute one nor the process of passing a URI reference to a process or software component responsible for dereferencing it SHOULD trigger escaping. When escaping does occur, it MUST be performed as follows:

1. Each character to be escaped is represented in UTF-8 [Unicode3] as one or more bytes.
2. The resulting bytes are escaped with the URI escaping mechanism (that is, converted to %*HH*, where HH is the hexadecimal notation of the byte value).
3. The original character is replaced by the resulting character sequence.

[Definition: In addition to a system identifier, an external identifier MAY include a **public identifier**.] An XML processor attempting to retrieve the entity's content MAY use any combination of the public and system identifiers as well as additional information outside the scope of this specification to try to generate an alternative URI reference. If the processor is unable to do so, it MUST use the URI reference specified in the system literal. Before a match is attempted, all strings of white space in the public identifier MUST be normalized to single space characters (#x20), and leading and trailing white space MUST be removed.

Examples of external entity declarations:

```
<!ENTITY open-hatch
         SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
         SYSTEM "../grafix/OpenHatch.gif"
         NDATA gif >
```

4.3 Parsed Entities

4.3.1 The Text Declaration

External parsed entities SHOULD each begin with a **text declaration**.

Text Declaration

```
[77]  TextDecl  ::=  '<?xml' VersionInfo? EncodingDecl S? '?>'
```

The text declaration MUST be provided literally, not by reference to a parsed entity. The text declaration MUST NOT appear at any position other than the beginning of an external parsed entity. The text declaration in an external parsed entity is not considered part of its replacement text.

4.3.2 Well-Formed Parsed Entities

The document entity is well-formed if it matches the production labeled document. An external general parsed entity is well-formed if it matches the production labeled extParsedEnt. All external parameter entities are well-formed by definition.

Well-Formed External Parsed Entity

```
[78]  extParsedEnt  ::=  TextDecl? content
```

An internal general parsed entity is well-formed if its replacement text matches the production labeled content. All internal parameter entities are well-formed by definition.

A consequence of well-formedness in general entities is that the logical and physical structures in an XML document are properly nested; no start-tag, end-tag, empty-element tag, element, comment, processing instruction, character reference, or entity reference can begin in one entity and end in another.

4.3.3 Character Encoding in Entities

Each external parsed entity in an XML document MAY use a different encoding for its characters. All XML processors MUST be able to read entities in both the UTF-8 and UTF-16 encodings. The terms "UTF-8" and "UTF-16" in this specification do not apply to character encodings with any other labels, even if the encodings or labels are very similar to UTF-8 or UTF-16.

Entities encoded in UTF-16 MUST and entities encoded in UTF-8 MAY begin with the Byte Order Mark described by Annex H of [ISO/IEC 10646:2000], section 2.4 of [Unicode], and section 2.7 of [Unicode3] (the ZERO WIDTH NO-BREAK SPACE character, #xFEFF). This is an encoding signature, not part of either the markup or the character data of the XML document. XML processors MUST be able to use this character to differentiate between UTF-8 and UTF-16 encoded documents.

Although an XML processor is required to read only entities in the UTF-8 and UTF-16 encodings, it is recognized that other encodings are used around the world, and it may be desired for XML processors to read entities that use them. In the absence of external character encoding information (such as MIME headers), parsed entities which are stored in an encoding other than UTF-8 or UTF-16 MUST begin with a text declaration (see **4.3.1 The Text Declaration**) containing an encoding declaration:

Encoding Declaration

```
[80]  EncodingDecl  ::=  S 'encoding' Eq ('"' EncName '"' | "'" EncName
                         "'" )
[81]  EncName       ::=  [A-Za-z] ([A-Za-z0-9._] | '-')*                  /* Encoding name contains only Latin characters
                                                                          */
```

In the document entity, the encoding declaration is part of the XML declaration. The EncName is the name of the encoding used.

In an encoding declaration, the values "`UTF-8`", "`UTF-16`", "`ISO-10646-UCS-2`", and "`ISO-10646-UCS-4`" SHOULD be used for the various encodings and transformations of Unicode / ISO/IEC 10646, the values "`ISO-8859-1`", "`ISO-8859-2`", ... "`ISO-8859-`*n*" (where *n* is the part number) SHOULD be used for the parts of ISO 8859, and the values "`ISO-2022-JP`", "`Shift_JIS`", and "`EUC-JP`" SHOULD be used for the various encoded forms of JIS X-0208-1997. It is RECOMMENDED that character encodings registered (as *charset*s) with the Internet Assigned Numbers Authority [IANA-CHARSETS], other than those just listed, be referred to using their registered names; other encodings SHOULD use names starting with an "x-" prefix. XML processors SHOULD match character encoding names in a case-insensitive way and SHOULD either interpret an IANA-registered name as the encoding registered at IANA for that name or treat it as unknown (processors are, of course, not required to support all IANA-registered encodings).

In the absence of information provided by an external transport protocol (e.g. HTTP or MIME), it is a fatal error for an entity including an encoding declaration to be presented to the XML processor in an encoding other than that named in the declaration, or for an entity which begins with neither a Byte Order Mark nor an encoding declaration to use an encoding other than UTF-8. Note that since ASCII is a subset of UTF-8, ordinary ASCII entities do not strictly need an encoding declaration.

It is a fatal error for a TextDecl to occur other than at the beginning of an external entity.

It is a fatal error when an XML processor encounters an entity with an encoding that it is unable to process. It is a fatal error if an XML entity is determined (via default, encoding declaration, or higher-level protocol) to be in a certain encoding but contains byte sequences that are not legal in that encoding. Specifically, it is a fatal error if an entity encoded in UTF-8 contains any irregular code unit sequences, as defined in Unicode 3.1 [Unicode3]. Unless an encoding is determined by a higher-level protocol, it is also a fatal error if an XML entity contains no encoding declaration and its content is not legal UTF-8 or UTF-16.

Examples of text declarations containing encoding declarations:

```
<?xml encoding='UTF-8'?>
<?xml encoding='EUC-JP'?>
```

4.4 XML Processor Treatment of Entities and References

The table below summarizes the contexts in which character references, entity references, and invocations of unparsed entities might appear and the REQUIRED behavior of an XML processor in each case. The labels in the leftmost column describe the recognition context:

Reference in Content

as a reference anywhere after the start-tag and before the end-tag of an element; corresponds to the nonterminal content.

Reference in Attribute Value

as a reference within either the value of an attribute in a start-tag, or a default value in an attribute declaration; corresponds to the nonterminal AttValue.

Occurs as Attribute Value

as a Name, not a reference, appearing either as the value of an attribute which has been declared as type **ENTITY**, or as one of the space-separated tokens in the value of an attribute which has been declared as type **ENTITIES**.

Reference in Entity Value

as a reference within a parameter or internal entity's literal entity value in the entity's declaration; corresponds to the nonterminal EntityValue.

Reference in DTD

as a reference within either the internal or external subsets of the DTD, but outside of an EntityValue, AttValue, PI, Comment, SystemLiteral, PubidLiteral, or the contents of an ignored conditional section (see **3.4 Conditional Sections**).

.

	Entity Type				Character
	Parameter	Internal General	External Parsed General	Unparsed	
Reference in Content	*Not recognized*	*Included*	*Included if validating*	*Forbidden*	*Included*
Reference in Attribute Value	*Not recognized*	*Included in literal*	*Forbidden*	*Forbidden*	*Included*
Occurs as Attribute Value	*Not recognized*	*Forbidden*	*Forbidden*	*Notify*	*Not recognized*
Reference in EntityValue	*Included in literal*	*Bypassed*	*Bypassed*	*Error*	*Included*
Reference in DTD	*Included as PE*	*Forbidden*	*Forbidden*	*Forbidden*	*Forbidden*

4.4.1 Not Recognized

Outside the DTD, the % character has no special significance; thus, what would be parameter entity references in the DTD are not recognized as

markup in content. Similarly, the names of unparsed entities are not recognized except when they appear in the value of an appropriately declared attribute.

4.4.2 Included

[Definition: An entity is **included** when its replacement text is retrieved and processed, in place of the reference itself, as though it were part of the document at the location the reference was recognized.] The replacement text MAY contain both character data and (except for parameter entities) markup, which MUST be recognized in the usual way. (The string "`AT&T;`" expands to "`AT&T;`" and the remaining ampersand is not recognized as an entity-reference delimiter.) A character reference is **included** when the indicated character is processed in place of the reference itself.

4.4.3 Included If Validating

When an XML processor recognizes a reference to a parsed entity, in order to validate the document, the processor MUST include its replacement text. If the entity is external, and the processor is not attempting to validate the XML document, the processor MAY, but need not, include the entity's replacement text. If a non-validating processor does not include the replacement text, it MUST inform the application that it recognized, but did not read, the entity.

This rule is based on the recognition that the automatic inclusion provided by the SGML and XML entity mechanism, primarily designed to support modularity in authoring, is not necessarily appropriate for other applications, in particular document browsing. Browsers, for example, when encountering an external parsed entity reference, might choose to provide a visual indication of the entity's presence and retrieve it for display only on demand.

4.4.4 Forbidden

The following are forbidden, and constitute fatal errors:

- the appearance of a reference to an unparsed entity, except in the EntityValue in an entity declaration.
- the appearance of any character or general-entity reference in the DTD except within an EntityValue or AttValue.
- a reference to an external entity in an attribute value.

4.4.5 Included in Literal

When an entity reference appears in an attribute value, or a parameter entity reference appears in a literal entity value, its replacement text MUST be processed in place of the reference itself as though it were part of the document at the location the reference was recognized, except that a single or double quote character in the replacement text MUST always be treated as a normal data character and MUST NOT terminate the literal. For example, this is well-formed:

```
<!ENTITY % YN '"Yes"' >
<!ENTITY WhatHeSaid "He said %YN;" >
```

while this is not:

```
<!ENTITY EndAttr "27'" >
<element attribute='a-&EndAttr;>
```

4.4.6 Notify

When the name of an unparsed entity appears as a token in the value of an attribute of declared type **ENTITY** or **ENTITIES**, a validating processor MUST inform the application of the system and public (if any) identifiers for both the entity and its associated notation.

4.4.7 Bypassed

When a general entity reference appears in the EntityValue in an entity declaration, it MUST be bypassed and left as is.

4.4.8 Included as PE

Just as with external parsed entities, parameter entities need only be *included if validating*. When a parameter-entity reference is recognized in the DTD and included, its replacement text MUST be enlarged by the attachment of one leading and one following space (#x20) character; the intent is to constrain the replacement text of parameter entities to contain an integral number of grammatical tokens in the DTD. This behavior MUST NOT apply to parameter entity references within entity values; these are described in **4.4.5 Included in Literal**.

4.4.9 Error

It is an error for a reference to an unparsed entity to appear in the EntityValue in an entity declaration.

4.5 Construction of Entity Replacement Text

In discussing the treatment of entities, it is useful to distinguish two forms of the entity's value. [Definition: For an internal entity, the **literal entity value** is the quoted string actually present in the entity declaration, corresponding to the non-terminal EntityValue.] [Definition: For an external entity, the **literal entity value** is the exact text contained in the entity.] [Definition: For an internal entity, the **replacement text** is the content of the entity, after replacement of character references and parameter-entity references.] [Definition: For an external entity, the **replacement text** is the content of the entity, after stripping the text declaration (leaving any surrounding whitespace) if there is one but without any replacement of

character references or parameter-entity references.]

The literal entity value as given in an internal entity declaration (EntityValue) MAY contain character, parameter-entity, and general-entity references. Such references MUST be contained entirely within the literal entity value. The actual replacement text that is included (or included in literal) as described above MUST contain the *replacement text* of any parameter entities referred to, and MUST contain the character referred to, in place of any character references in the literal entity value; however, general-entity references MUST be left as-is, unexpanded. For example, given the following declarations:

```
<!ENTITY % pub    "&#xc9;ditions Gallimard" >
<!ENTITY   rights "All rights reserved" >
<!ENTITY   book   "La Peste: Albert Camus,
&#xA9; 1947 %pub;. &rights;" >
```

then the replacement text for the entity "book" is:

```
La Peste: Albert Camus,
© 1947 Éditions Gallimard. &rights;
```

The general-entity reference "&rights;" would be expanded should the reference "&book;" appear in the document's content or an attribute value.

These simple rules may have complex interactions; for a detailed discussion of a difficult example, see **D Expansion of Entity and Character References**.

4.6 Predefined Entities

[Definition: Entity and character references MAY both be used to **escape** the left angle bracket, ampersand, and other delimiters. A set of general entities (amp, lt, gt, apos, quot) is specified for this purpose. Numeric character references MAY also be used; they are expanded immediately when recognized and MUST be treated as character data, so the numeric character references "<" and "&" MAY be used to escape < and & when they occur in character data.]

All XML processors MUST recognize these entities whether they are declared or not. For interoperability, valid XML documents SHOULD declare these entities, like any others, before using them. If the entities lt or amp are declared, they MUST be declared as internal entities whose replacement text is a character reference to the respective character (less-than sign or ampersand) being escaped; the double escaping is REQUIRED for these entities so that references to them produce a well-formed result. If the entities gt, apos, or quot are declared, they MUST be declared as internal entities whose replacement text is the single character being escaped (or a character reference to that character; the double escaping here is OPTIONAL but harmless). For example:

```
<!ENTITY lt     "&#38;#60;">
<!ENTITY gt     "&#62;">
<!ENTITY amp    "&#38;#38;">
<!ENTITY apos   "&#39;">
<!ENTITY quot   "&#34;">
```

4.7 Notation Declarations

[Definition: **Notations** identify by name the format of unparsed entities, the format of elements which bear a notation attribute, or the application to which a processing instruction is addressed.]

[Definition: **Notation declarations** provide a name for the notation, for use in entity and attribute-list declarations and in attribute specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.]

Notation Declarations

```
[82]  NotationDecl  ::=  '<!NOTATION' S Name S (ExternalID | PublicID) S? '>' [VC: Unique Notation Name]
[83]  PublicID      ::=  'PUBLIC' S PubidLiteral
```

Validity constraint: Unique Notation Name

A given Name MUST NOT be declared in more than one notation declaration.

XML processors MUST provide applications with the name and external identifier(s) of any notation declared and referred to in an attribute value, attribute definition, or entity declaration. They MAY additionally resolve the external identifier into the system identifier, file name, or other information needed to allow the application to call a processor for data in the notation described. (It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.)

4.8 Document Entity

[Definition: The **document entity** serves as the root of the entity tree and a starting-point for an XML processor.] This specification does not specify how the document entity is to be located by an XML processor; unlike other entities, the document entity has no name and might well

appear on a processor input stream without any identification at all.

5 Conformance

5.1 Validating and Non-Validating Processors

Conforming XML processors fall into two classes: validating and non-validating.

Validating and non-validating processors alike MUST report violations of this specification's well-formedness constraints in the content of the document entity and any other parsed entities that they read.

[Definition: **Validating processors** MUST, at user option, report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification.] To accomplish this, validating XML processors MUST read and process the entire DTD and all external parsed entities referenced in the document.

Non-validating processors are REQUIRED to check only the document entity, including the entire internal DTD subset, for well-formedness. [Definition: While they are not required to check the document for validity, they are REQUIRED to **process** all the declarations they read in the internal DTD subset and in any parameter entity that they read, up to the first reference to a parameter entity that they do *not* read; that is to say, they MUST use the information in those declarations to *normalize* attribute values, *include* the replacement text of internal entities, and supply *default attribute values*.] Except when `standalone="yes"`, they MUST NOT process entity declarations or attribute-list declarations encountered after a reference to a parameter entity that is not read, since the entity may have contained overriding declarations; when `standalone="yes"`, processors MUST process these declarations.

Note that when processing invalid documents with a non-validating processor the application may not be presented with consistent information. For example, several requirements for uniqueness within the document may not be met, including more than one element with the same id, duplicate declarations of elements or notations with the same name, etc. In these cases the behavior of the parser with respect to reporting such information to the application is undefined.

5.2 Using XML Processors

The behavior of a validating XML processor is highly predictable; it must read every piece of a document and report all well-formedness and validity violations. Less is required of a non-validating processor; it need not read any part of the document other than the document entity. This has two effects that may be important to users of XML processors:

- Certain well-formedness errors, specifically those that require reading external entities, may fail to be detected by a non-validating processor. Examples include the constraints entitled *Entity Declared*, *Parsed Entity*, and *No Recursion*, as well as some of the cases described as *forbidden* in **4.4 XML Processor Treatment of Entities and References**.
- The information passed from the processor to the application may vary, depending on whether the processor reads parameter and external entities. For example, a non-validating processor may fail to *normalize* attribute values, *include* the replacement text of internal entities, or supply *default attribute values*, where doing so depends on having read declarations in external or parameter entities.

For maximum reliability in interoperating between different XML processors, applications which use non-validating processors SHOULD NOT rely on any behaviors not required of such processors. Applications which require DTD facilities not related to validation (such as the declaration of default attributes and internal entities that are or may be specified in external entities) SHOULD use validating XML processors.

6 Notation

The formal grammar of XML is given in this specification using a simple Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one symbol, in the form

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are the start symbol of a regular language, otherwise with an initial lowercase letter. Literal strings are quoted.

Within the expression on the right-hand side of a rule, the following expressions are used to match strings of one or more characters:

`#xN`

where `N` is a hexadecimal integer, the expression matches the character whose number (code point) in ISO/IEC 10646 is `N`. The number of leading zeros in the `#xN` form is insignificant.

`[a-zA-Z], [#xN-#xN]`

matches any Char with a value in the range(s) indicated (inclusive).

`[abc], [#xN#xN#xN]`

matches any Char with a value among the characters enumerated. Enumerations and ranges can be mixed in one set of brackets.

`[^a-z], [^#xN-#xN]`

matches any Char with a value *outside* the range indicated.

`[^abc], [^#xN#xN#xN]`

matches any Char with a value not among the characters given. Enumerations and ranges of forbidden values can be mixed in one set of brackets.

"string"

matches a literal string matching that given inside the double quotes.

'string'

matches a literal string matching that given inside the single quotes.

These symbols may be combined to match more complex patterns as follows, where A and B represent simple expressions:

(expression)

expression is treated as a unit and may be combined as described in this list.

A?

matches A or nothing; optional A.

A B

matches A followed by B. This operator has higher precedence than alternation; thus A B | C D is identical to (A B) | (C D).

A | B

matches A or B.

A - B

matches any string that matches A but does not match B.

A+

matches one or more occurrences of A. Concatenation has higher precedence than alternation; thus A+ | B+ is identical to (A+) | (B+).

A*

matches zero or more occurrences of A. Concatenation has higher precedence than alternation; thus A* | B* is identical to (A*) | (B*).

Other notations used in the productions are:

/* ... */

comment.

[wfc: ...]

well-formedness constraint; this identifies by name a constraint on well-formed documents associated with a production.

[vc: ...]

validity constraint; this identifies by name a constraint on valid documents associated with a production.

A References

A.1 Normative References

IANA-CHARSETS
(Internet Assigned Numbers Authority) *Official Names for Character Sets*, ed. Keld Simonsen et al. (See http://www.iana.org/assignments/character-sets.)

IETF RFC 2119
IETF (Internet Engineering Task Force). *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*. Scott Bradner, 1997. (See http://www.ietf.org/rfc/rfc2119.txt.)

IETF RFC 2396
IETF (Internet Engineering Task Force). *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*. T. Berners-Lee, R. Fielding, L. Masinter. 1998. (See http://www.ietf.org/rfc/rfc2396.txt.)

IETF RFC 2732
IETF (Internet Engineering Task Force). *RFC 2732: Format for Literal IPv6 Addresses in URL's*. R. Hinden, B. Carpenter, L. Masinter. 1999. (See http://www.ietf.org/rfc/rfc2732.txt.)

IETF RFC 3066
IETF (Internet Engineering Task Force). *RFC 3066: Tags for the Identification of Languages*, ed. H. Alvestrand. 2001. (See http://www.ietf.org/rfc/rfc3066.txt.)

ISO/IEC 10646
ISO (International Organization for Standardization). *ISO/IEC 10646-1:2000. Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane* and *ISO/IEC 10646-2:2001. Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 2: Supplementary Planes*, as, from time to time, amended, replaced by a new edition or expanded by the addition of new parts. [Geneva]: International Organization for Standardization. (See http://www.iso.ch for the latest version.)

ISO/IEC 10646:2000
ISO (International Organization for Standardization). *ISO/IEC 10646-1:2000. Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 2000.

Unicode
The Unicode Consortium. *The Unicode Standard, Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press, 1996.

Unicode3

The Unicode Consortium. *The Unicode Standard, Version 3.2*, defined by: *The Unicode Standard, Version 3.0* (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5), as amended by the *Unicode Standard Annex #27: Unicode 3.1* (http://www.unicode.org/reports/tr27) and the *Unicode Standard Annex #28: Unicode 3.2* (http://www.unicode.org/reports/tr28).

A.2 Other References

Aho/Ullman
Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Reading: Addison-Wesley, 1986, rpt. corr. 1988.

Brüggemann-Klein
Brüggemann-Klein, Anne. *Formal Models in Document Processing*. Habilitationsschrift. Faculty of Mathematics at the University of Freiburg, 1993. (See ftp://ftp.informatik.uni-freiburg.de/documents/papers/brueggem/habil.ps.)

Brüggemann-Klein and Wood
Brüggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages*. Universität Freiburg, Institut für Informatik, Bericht 38, Oktober 1991. Extended abstract in A. Finkel, M. Jantzen, Hrsg., STACS 1992, S. 173-184. Springer-Verlag, Berlin 1992. Lecture Notes in Computer Science 577. Full version titled *One-Unambiguous Regular Languages* in Information and Computation 140 (2): 229-253, February 1998.

Clark
James Clark. *Comparison of SGML and XML*. (See http://www.w3.org/TR/NOTE-sgml-xml-971215.)

IANA-LANGCODES
(Internet Assigned Numbers Authority) *Registry of Language Tags*, ed. Keld Simonsen et al. (See http://www.iana.org/assignments/language-tags.)

IETF RFC 2141
IETF (Internet Engineering Task Force). *RFC 2141: URN Syntax*, ed. R. Moats. 1997. (See http://www.ietf.org/rfc/rfc2141.txt.)

IETF RFC 3023
IETF (Internet Engineering Task Force). *RFC 3023: XML Media Types*. eds. M. Murata, S. St.Laurent, D. Kohn. 2001. (See http://www.ietf.org/rfc/rfc3023.txt.)

IETF RFC 2781
IETF (Internet Engineering Task Force). *RFC 2781: UTF-16, an encoding of ISO 10646*, ed. P. Hoffman, F. Yergeau. 2000. (See http://www.ietf.org/rfc/rfc2781.txt.)

ISO 639
(International Organization for Standardization). *ISO 639:1988 (E). Code for the representation of names of languages.* [Geneva]: International Organization for Standardization, 1988.

ISO 3166
(International Organization for Standardization). *ISO 3166-1:1997 (E). Codes for the representation of names of countries and their subdivisions — Part 1: Country codes* [Geneva]: International Organization for Standardization, 1997.

ISO 8879
ISO (International Organization for Standardization). *ISO 8879:1986(E). Information processing — Text and Office Systems — Standard Generalized Markup Language (SGML).* First edition — 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO/IEC 10744
ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology — Hypermedia/Time-based Structuring Language (HyTime).* [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe.* [Geneva]: International Organization for Standardization, 1996.

WEBSGML
ISO (International Organization for Standardization). *ISO 8879:1986 TC2. Information technology — Document Description and Processing Languages*. [Geneva]: International Organization for Standardization, 1998. (See http://www.sgmlsource.com/8879/n0029.htm.)

XML Names
Tim Bray, Dave Hollander, and Andrew Layman, editors. *Namespaces in XML*. Textuality, Hewlett-Packard, and Microsoft. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/REC-xml-names/.)

B Character Classes

Following the characteristics defined in the Unicode standard, characters are classed as base characters (among others, these contain the alphabetic characters of the Latin alphabet), ideographic characters, and combining characters (among others, this class contains most diacritics). Digits and extenders are also distinguished.

Characters

```
[84]  Letter        ::=  BaseChar | Ideographic
[85]  BaseChar      ::=  [#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-
                         #x00FF] | [#x0100-#x0131] | [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E]
                         | [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5] | [#x01FA-#x0217] | [#x0250-
                         #x02A8] | [#x02BB-#x02C1] | #x0386 | [#x0388-#x038A] | #x038C | [#x038E-#x03A1]
                         | [#x03A3-#x03CE] | [#x03D0-#x03D6] | #x03DA | #x03DC | #x03DE | #x03E0 | [#x03E2-
                         #x03F3] | [#x0401-#x040C] | [#x040E-#x044F] | [#x0451-#x045C] | [#x045E-#x0481]
                         | [#x0490-#x04C4] | [#x04C7-#x04C8] | [#x04CB-#x04CC] | [#x04D0-#x04EB] | [#x04EE-
                         #x04F5] | [#x04F8-#x04F9] | [#x0531-#x0556] | #x0559 | [#x0561-#x0586] | [#x05D0-
                         #x05EA] | [#x05F0-#x05F2] | [#x0621-#x063A] | [#x0641-#x064A] | [#x0671-#x06B7]
                         | [#x06BA-#x06BE] | [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5 | [#x06E5-#x06E6]
                         | [#x0905-#x0939] | #x093D | [#x0958-#x0961] | [#x0985-#x098C] | [#x098F-#x0990]
                         | [#x0993-#x09A8] | [#x09AA-#x09B0] | #x09B2 | [#x09B6-#x09B9] | [#x09DC-#x09DD]
                         | [#x09DF-#x09E1] | [#x09F0-#x09F1] | [#x0A05-#x0A0A] | [#x0A0F-#x0A10] | [#x0A13-
                         #x0A28] | [#x0A2A-#x0A30] | [#x0A32-#x0A33] | [#x0A35-#x0A36] | [#x0A38-#x0A39]
                         | [#x0A59-#x0A5C] | #x0A5E | [#x0A72-#x0A74] | [#x0A85-#x0A8B] | #x0A8D | [#x0A8F-
                         #x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0] | [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9]
                         | #x0ABD | #x0AE0 | [#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28] | [#x0B2A-
                         #x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39] | #x0B3D | [#x0B5C-#x0B5D] | [#x0B5F-
```

```
                    #x0B61] | [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-#x0B95] | [#x0B99-#x0B9A]
                    | #x0B9C | [#x0B9E-#x0B9F] | [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA] | [#x0BAE-#x0BB5]
                    | [#x0BB7-#x0BB9] | [#x0C05-#x0C0C] | [#x0C0E-#x0C10] | [#x0C12-#x0C28] | [#x0C2A-
                    #x0C33] | [#x0C35-#x0C39] | [#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90]
                    | [#x0C92-#x0CA8] | [#x0CAA-#x0CB3] | [#x0CB5-#x0CB9] | #x0CDE | [#x0CE0-#x0CE1]
                    | [#x0D05-#x0D0C] | [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39] | [#x0D60-
                    #x0D61] | [#x0E01-#x0E2E] | #x0E30 | [#x0E32-#x0E33] | [#x0E40-#x0E45] | [#x0E81-
                    #x0E82] | #x0E84 | [#x0E87-#x0E88] | #x0E8A | #x0E8D | [#x0E94-#x0E97] | [#x0E99-
                    #x0E9F] | [#x0EA1-#x0EA3] | #x0EA5 | #x0EA7 | [#x0EAA-#x0EAB] | [#x0EAD-#x0EAE]
                    | #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD | [#x0EC0-#x0EC4] | [#x0F40-#x0F47] | [#x0F49-
                    #x0F69] | [#x10A0-#x10C5] | [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103] | [#x1105-
                    #x1107] | #x1109 | [#x110B-#x110C] | [#x110E-#x1112] | #x113C | #x113E | #x1140
                    | #x114C | #x114E | #x1150 | [#x1154-#x1155] | #x1159 | [#x115F-#x1161] | #x1163
                    | #x1165 | #x1167 | #x1169 | [#x116D-#x116E] | [#x1172-#x1173] | #x1175 | #x119E
                    | #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8] | #x11BA | [#x11BC-#x11C2]
                    | #x11EB | #x11F0 | #x11F9 | [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15]
                    | [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D] | [#x1F50-#x1F57] | #x1F59
                    | #x1F5B | #x1F5D | [#x1F5F-#x1F7D] | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC] | #x1FBE
                    | [#x1FC2-#x1FC4] | [#x1FC6-#x1FCC] | [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB] | [#x1FE0-
                    #x1FEC] | [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126 | [#x212A-#x212B] | #x212E
                    | [#x2180-#x2182] | [#x3041-#x3094] | [#x30A1-#x30FA] | [#x3105-#x312C] | [#xAC00-
                    #xD7A3]
[86]  Ideographic    ::=  [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]
[87]  CombiningChar  ::=  [#x0300-#x0345] | [#x0360-#x0361] | [#x0483-#x0486] | [#x0591-#x05A1] | [#x05A3-
                    #x05B9] | [#x05BB-#x05BD] | #x05BF | [#x05C1-#x05C2] | #x05C4 | [#x064B-#x0652]
                    | #x0670 | [#x06D6-#x06DC] | [#x06DD-#x06DF] | [#x06E0-#x06E4] | [#x06E7-#x06E8]
                    | [#x06EA-#x06ED] | [#x0901-#x0903] | #x093C | [#x093E-#x094C] | #x094D | [#x0951-
                    #x0954] | [#x0962-#x0963] | [#x0981-#x0983] | #x09BC | #x09BE | #x09BF | [#x09C0-
                    #x09C4] | [#x09C7-#x09C8] | [#x09CB-#x09CD] | #x09D7 | [#x09E2-#x09E3] | #x0A02
                    | #x0A3C | #x0A3E | #x0A3F | [#x0A40-#x0A42] | [#x0A47-#x0A48] | [#x0A4B-#x0A4D]
                    | [#x0A70-#x0A71] | [#x0A81-#x0A83] | #x0ABC | [#x0ABE-#x0AC5] | [#x0AC7-#x0AC9]
                    | [#x0ACB-#x0ACD] | [#x0B01-#x0B03] | #x0B3C | [#x0B3E-#x0B43] | [#x0B47-#x0B48]
                    | [#x0B4B-#x0B4D] | [#x0B56-#x0B57] | [#x0B82-#x0B83] | [#x0BBE-#x0BC2] | [#x0BC6-
                    #x0BC8] | [#x0BCA-#x0BCD] | #x0BD7 | [#x0C01-#x0C03] | [#x0C3E-#x0C44] | [#x0C46-
                    #x0C48] | [#x0C4A-#x0C4D] | [#x0C55-#x0C56] | [#x0C82-#x0C83] | [#x0CBE-#x0CC4]
                    | [#x0CC6-#x0CC8] | [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6] | [#x0D02-#x0D03] | [#x0D3E-
                    #x0D43] | [#x0D46-#x0D48] | [#x0D4A-#x0D4D] | #x0D57 | #x0E31 | [#x0E34-#x0E3A]
                    | [#x0E47-#x0E4E] | #x0EB1 | [#x0EB4-#x0EB9] | [#x0EBB-#x0EBC] | [#x0EC8-#x0ECD]
                    | [#x0F18-#x0F19] | #x0F35 | #x0F37 | #x0F39 | #x0F3E | #x0F3F | [#x0F71-#x0F84]
                    | [#x0F86-#x0F8B] | [#x0F90-#x0F95] | #x0F97 | [#x0F99-#x0FAD] | [#x0FB1-#x0FB7]
                    | #x0FB9 | [#x20D0-#x20DC] | #x20E1 | [#x302A-#x302F] | #x3099 | #x309A
[88]  Digit          ::=  [#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9] | [#x0966-#x096F] | [#x09E6-
                    #x09EF] | [#x0A66-#x0A6F] | [#x0AE6-#x0AEF] | [#x0B66-#x0B6F] | [#x0BE7-#x0BEF]
                    | [#x0C66-#x0C6F] | [#x0CE6-#x0CEF] | [#x0D66-#x0D6F] | [#x0E50-#x0E59] | [#x0ED0-
                    #x0ED9] | [#x0F20-#x0F29]
[89]  Extender       ::=  #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46 | #x0EC6 | #x3005 | [#x3031-
                    #x3035] | [#x309D-#x309E] | [#x30FC-#x30FE]
```

The character classes defined here can be derived from the Unicode 2.0 character database as follows:

- Name start characters must have one of the categories Ll, Lu, Lo, Lt, Nl.
- Name characters other than Name-start characters must have one of the categories Mc, Me, Mn, Lm, or Nd.
- Characters in the compatibility area (i.e. with character code greater than #xF900 and less than #xFFFE) are not allowed in XML names.
- Characters which have a font or compatibility decomposition (i.e. those with a "compatibility formatting tag" in field 5 of the database -- marked by field 5 beginning with a "<") are not allowed.
- The following characters are treated as name-start characters rather than name characters, because the property file classifies them as Alphabetic: [#x02BB-#x02C1], #x0559, #x06E5, #x06E6.
- Characters #x20DD-#x20E0 are excluded (in accordance with Unicode 2.0, section 5.14).
- Character #x00B7 is classified as an extender, because the property list so identifies it.
- Character #x0387 is added as a name character, because #x00B7 is its canonical equivalent.
- Characters ':' and '_' are allowed as name-start characters.
- Characters '-' and '.' are allowed as name characters.

C XML and SGML (Non-Normative)

XML is designed to be a subset of SGML, in that every XML document should also be a conforming SGML document. For a detailed comparison of the additional restrictions that XML places on documents beyond those of SGML, see [Clark].

D Expansion of Entity and Character References (Non-Normative)

This appendix contains some examples illustrating the sequence of entity- and character-reference recognition and expansion, as specified in **4.4 XML Processor Treatment of Entities and References**.

If the DTD contains the declaration

```
<!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>" >
```

then the XML processor will recognize the character references when it parses the entity declaration, and resolve them before storing the following string as the value of the entity "example":

```
<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>
```

A reference in the document to "&example;" will cause the text to be reparsed, at which time the start- and end-tags of the p element will be recognized and the three references will be recognized and expanded, resulting in a p element with the following content (all data, no delimiters or markup):

```
An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;).
```

A more complex example will illustrate the rules and their effects fully. In the following example, the line numbers are solely for reference.

```
1 <?xml version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60;!ENTITY tricky "error-prone" >' >
6 %xx;
7 ]>
8 <test>This sample shows a &tricky; method.</test>
```

This produces the following:

- in line 4, the reference to character 37 is expanded immediately, and the parameter entity "xx" is stored in the symbol table with the value "%zz;". Since the replacement text is not rescanned, the reference to parameter entity "zz" is not recognized. (And it would be an error if it were, since "zz" is not yet declared.)
- in line 5, the character reference "<" is expanded immediately and the parameter entity "zz" is stored with the replacement text "<!ENTITY tricky "error-prone" >", which is a well-formed entity declaration.
- in line 6, the reference to "xx" is recognized, and the replacement text of "xx" (namely "%zz;") is parsed. The reference to "zz" is recognized in its turn, and its replacement text ("<!ENTITY tricky "error-prone" >") is parsed. The general entity "tricky" has now been declared, with the replacement text "error-prone".
- in line 8, the reference to the general entity "tricky" is recognized, and it is expanded, so the full content of the test element is the self-describing (and ungrammatical) string *This sample shows a error-prone method.*

E Deterministic Content Models (Non-Normative)

As noted in **3.2.1 Element Content**, it is required that content models in element type declarations be deterministic. This requirement is for compatibility with SGML (which calls deterministic content models "unambiguous"); XML processors built using SGML systems may flag non-deterministic content models as errors.

For example, the content model ((b, c) | (b, d)) is non-deterministic, because given an initial b the XML processor cannot know which b in the model is being matched without looking ahead to see which element follows the b. In this case, the two references to b can be collapsed into a single reference, making the model read (b, (c | d)). An initial b now clearly matches only a single name in the content model. The processor doesn't need to look ahead to see what follows; either c or d would be accepted.

More formally: a finite state automaton may be constructed from the content model using the standard algorithms, e.g. algorithm 3.5 in section 3.9 of Aho, Sethi, and Ullman [Aho/Ullman]. In many such algorithms, a follow set is constructed for each position in the regular expression (i.e., each leaf node in the syntax tree for the regular expression); if any position has a follow set in which more than one following position is labeled with the same element type name, then the content model is in error and may be reported as an error.

Algorithms exist which allow many but not all non-deterministic content models to be reduced automatically to equivalent deterministic models; see Brüggemann-Klein 1991 [Brüggemann-Klein].

F Autodetection of Character Encodings (Non-Normative)

The XML encoding declaration functions as an internal label on each entity, indicating which character encoding is in use. Before an XML processor can read the internal label, however, it apparently has to know what character encoding is in use—which is what the internal label is trying to indicate. In the general case, this is a hopeless situation. It is not entirely hopeless in XML, however, because XML limits the general case in two ways: each implementation is assumed to support only a finite set of character encodings, and the XML encoding declaration is restricted in position and content in order to make it feasible to autodetect the character encoding in use in each entity in normal cases. Also, in many cases other sources of information are available in addition to the XML data stream itself. Two cases may be distinguished, depending on whether the XML entity is presented to the processor without, or with, any accompanying (external) information. We consider the first case first.

F.1 Detection Without External Encoding Information

Because each XML entity not accompanied by external encoding information and not in UTF-8 or UTF-16 encoding must begin with an XML encoding declaration, in which the first characters must be '<?xml', any conforming processor can detect, after two to four octets of input, which of the following cases apply. In reading this list, it may help to know that in UCS-4, '<' is "#x0000003C" and '?' is "#x0000003F", and the Byte Order Mark required of UTF-16 data streams is "#xFEFF". The notation ## is used to denote any byte value except that two consecutive ##s cannot be both 00.

With a Byte Order Mark:

00 00 FE FF	UCS-4, big-endian machine (1234 order)
FF FE 00 00	UCS-4, little-endian machine (4321 order)
00 00 FF FE	UCS-4, unusual octet order (2143)
FE FF 00 00	UCS-4, unusual octet order (3412)
FE FF ## ##	UTF-16, big-endian
FF FE ## ##	UTF-16, little-endian
EF BB BF	UTF-8

Without a Byte Order Mark:

00 00 00 3C 3C 00 00 00 00 00 3C 00 00 3C 00 00	UCS-4 or other encoding with a 32-bit code unit and ASCII characters encoded as ASCII values, in respectively big-endian (1234), little-endian (4321) and two unusual byte orders (2143 and 3412). The encoding declaration must be read to determine which of UCS-4 or other supported 32-bit encodings applies.
00 3C 00 3F	UTF-16BE or big-endian ISO-10646-UCS-2 or other encoding with a 16-bit code unit in big-endian order and ASCII characters encoded as ASCII values (the encoding declaration must be read to determine which)
3C 00 3F 00	UTF-16LE or little-endian ISO-10646-UCS-2 or other encoding with a 16-bit code unit in little-endian order and ASCII characters encoded as ASCII values (the encoding declaration must be read to determine which)
3C 3F 78 6D	UTF-8, ISO 646, ASCII, some part of ISO 8859, Shift-JIS, EUC, or any other 7-bit, 8-bit, or mixed-width encoding which ensures that the characters of ASCII have their normal positions, width, and values; the actual encoding declaration must be read to detect which of these applies, but since all of these encodings use the same bit patterns for the relevant ASCII characters, the encoding declaration itself may be read reliably
4C 6F A7 94	EBCDIC (in some flavor; the full encoding declaration must be read to tell which code page is in use)
Other	UTF-8 without an encoding declaration, or else the data stream is mislabeled (lacking a required encoding declaration), corrupt, fragmentary, or enclosed in a wrapper of some kind

Note:

In cases above which do not require reading the encoding declaration to determine the encoding, section 4.3.3 still requires that the encoding declaration, if present, be read and that the encoding name be checked to match the actual encoding of the entity. Also, it is possible that new character encodings will be invented that will make it necessary to use the encoding declaration to determine the encoding, in cases where this is not required at present.

This level of autodetection is enough to read the XML encoding declaration and parse the character-encoding identifier, which is still necessary to distinguish the individual members of each family of encodings (e.g. to tell UTF-8 from 8859, and the parts of 8859 from each other, or to distinguish the specific EBCDIC code page in use, and so on).

Because the contents of the encoding declaration are restricted to characters from the ASCII repertoire (however encoded), a processor can reliably read the entire encoding declaration as soon as it has detected which family of encodings is in use. Since in practice, all widely used character encodings fall into one of the categories above, the XML encoding declaration allows reasonably reliable in-band labeling of character encodings, even when external sources of information at the operating-system or transport-protocol level are unreliable. Character encodings such as UTF-7 that make overloaded usage of ASCII-valued bytes may fail to be reliably detected.

Once the processor has detected the character encoding in use, it can act appropriately, whether by invoking a separate input routine for each case, or by calling the proper conversion function on each character of input.

Like any self-labeling system, the XML encoding declaration will not work if any software changes the entity's character set or encoding without updating the encoding declaration. Implementors of character-encoding routines should be careful to ensure the accuracy of the internal and external information used to label the entity.

F.2 Priorities in the Presence of External Encoding Information

The second possible case occurs when the XML entity is accompanied by encoding information, as in some file systems and some network protocols. When multiple sources of information are available, their relative priority and the preferred method of handling conflict should be specified as part of the higher-level protocol used to deliver XML. In particular, please refer to [IETF RFC 3023] or its successor, which defines the `text/xml` and `application/xml` MIME types and provides some useful guidance. In the interests of interoperability, however, the following rule is recommended.

- If an XML entity is in a file, the Byte-Order Mark and encoding declaration are used (if present) to determine the character encoding.

G W3C XML Working Group (Non-Normative)

This specification was prepared and approved for publication by the W3C XML Working Group (WG). WG approval of this specification does not necessarily imply that all WG members voted for its approval. The current and former participants of the XML WG are:

- Jon Bosak, Sun (*Chair*)
- James Clark (*Technical Lead*)
- Tim Bray, Textuality and Netscape (*XML Co-editor*)
- Jean Paoli, Microsoft (*XML Co-editor*)
- C. M. Sperberg-McQueen, U. of Ill. (*XML Co-editor*)
- Dan Connolly, W3C (*W3C Liaison*)
- Paula Angerstein, Texcel
- Steve DeRose, INSO
- Dave Hollander, HP
- Eliot Kimber, ISOGEN
- Eve Maler, ArborText
- Tom Magliery, NCSA
- Murray Maloney, SoftQuad, Grif SA, Muzmo and Veo Systems
- MURATA Makoto (FAMILY Given), Fuji Xerox Information Systems
- Joel Nava, Adobe
- Conleth O'Connell, Vignette
- Peter Sharpe, SoftQuad
- John Tigue, DataChannel

H W3C XML Core Working Group (Non-Normative)

The third edition of this specification was prepared by the W3C XML Core Working Group (WG). The participants in the WG at the time of publication of this edition were:

- Leonid Arbouzov, Sun Microsystems
- Mary Brady
- John Cowan
- John Evdemon, Microsoft
- Andrew Fang, Arbortext
- Paul Grosso, Arbortext (*Co-Chair*)
- Arnaud Le Hors, IBM
- Dmitry Lenkov, Oracle
- Anjana Manian, Oracle
- Glenn Marcy, IBM
- Jonathan Marsh, Microsoft
- Sandra Martinez, NIST
- Liam Quin, W3C (*Staff Contact*)
- Lew Shannon
- Richard Tobin, University of Edinburgh
- Daniel Veillard
- Norman Walsh, Sun Microsystems (*Co-Chair*)
- François Yergeau (*Third Edition Editor*)

I Production Notes (Non-Normative)

This Third Edition was encoded in a slightly modified version of the XMLspec DTD, v2.5. The XHTML versions were produced with a combination of the xmlspec.xsl, diffspec.xsl, and REC-xml-3e.xsl XSLT stylesheets.

XML Information Set (Second Edition)

W3C Recommendation 4 February 2004

This version:
http://www.w3.org/TR/2004/REC-xml-infoset-20040204
Latest version:
http://www.w3.org/TR/xml-infoset
Previous version:
http://www.w3.org/TR/2003/PER-xml-infoset-20031210
Editors:
John Cowan, jcowan@reutershealth.com
Richard Tobin, richard@cogsci.ed.ac.uk

Please refer to the **errata** for this document, which may include some normative corrections.

See also **translations**.

Abstract

This specification provides a set of definitions for use in other specifications that need to refer to the information in an XML document.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found

in the W3C technical reports index at http://www.w3.org/TR/.

This document is a Recommendation of the W3C. It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document updates the Infoset to cover XML 1.1 and Namespaces 1.1, clarifies the consequences of certain kinds of invalidity, and corrects some typographical errors. It is a product of the W3C XML Activity. The English version of this specification is the only normative version. However, for translations of this document, see http://www.w3.org/2003/03/Translations/byTechnology?technology=xml-infoset.

Documentation of intellectual property possibly relevant to this recommendation may be found at the Working Group's public IPR disclosure page.

Please report errors in this document to www-xml-infoset-comments@w3.org (public archives are available). The errata list for this Recommendation is available at http://www.w3.org/2001/10/02/xml-infoset-errata.html.

Contents

---

1. Introduction

This specification defines an abstract data set called the ***XML Information Set*** (***Infoset***). Its purpose is to provide a consistent set of definitions for use in other specifications that need to refer to the information in a well-formed XML document [XML].

It does not attempt to be exhaustive; the primary criterion for inclusion of an information item or property has been that of expected usefulness in future specifications. Nor does it constitute a minimum set of information that must be returned by an XML processor.

An XML document has an information set if it is well-formed and satisfies the namespace constraints described below. There is no requirement for an XML document to be valid in order to have an information set.

Information sets may be created by methods (not described in this specification) other than parsing an XML document. See Synthetic Infosets below.

An XML document's information set consists of a number of ***information items***; the information set for any well-formed XML document will contain at least a document information item and several others. An information item is an abstract description of some part of an XML document: each information item has a set of associated named ***properties***. In this specification, the property names are shown in square brackets, **[thus]**. The types of information item are listed in section 2.

The XML Information Set does not require or favor a specific interface or class of interfaces. This specification presents the information set as a modified tree for the sake of clarity and simplicity, but there is no requirement that the XML Information Set be made available through a tree structure; other types of interfaces, including (but not limited to) event-based and query-based interfaces, are also capable of providing information conforming to the XML Information Set.

The terms "information set" and "information item" are similar in meaning to the generic terms "tree" and "node", as they are used in computing. However, the former terms are used in this specification to reduce possible confusion with other specific data models. Information items do *not* map one-to-one with the nodes of the DOM or the "tree" and "nodes" of the XPath data model.

In this specification, the words "must", "should", and "may" assume the meanings specified in [RFC2119], except that the words do not appear in uppercase.

XML Versions

Different versions of the XML specification may specify different parsing rules. The information set of an XML document is defined to be the one obtained by parsing it according to the rules of the specification whose version corresponds to that of the document. A document which does not specify a version number is considered to have version 1.0. If an XML processor accepts a document with a version number that it does not understand, it will not necessarily be able to produce the correct information set.

Namespaces

XML documents that do not conform to [Namespaces], though technically well-formed, are not considered to have meaningful information sets. That is, this specification does not define an information set for documents that have element or attribute names containing colons that are used in other ways than as prescribed by [Namespaces].

Furthermore, this specification does not define an information set for documents which use relative URI references in namespace declarations. This is in accordance with the decision of the W3C XML Plenary Interest Group described in [Relative Namespace URI References].

The value of a [namespace name] property is the normalized value of the corresponding namespace attribute; no additional URI escaping is applied to it by the processor.

Entities

An information set describes its XML document with entity references already expanded, that is, represented by the information items corresponding to their replacement text. However, there are various circumstances in which a processor may not perform this expansion. An entity may not be declared, or may not be retrievable. A non-validating processor may choose not to read all

declarations, and even if it does, may not expand all external entities. In these cases an unexpanded entity reference information item is used to represent the entity reference.

End-of-Line Handling

The values of all properties in the Infoset take account of the end-of-line normalization described in [XML], 2.11 "End-of-Line Handling".

Base URIs

Several information items have a [base URI] or [declaration base URI] property. These are computed according to [XML Base]. Note that retrieval of a resource may involve redirection at the parser level (for example, in an entity resolver) or below; in this case the base URI is the final URI used to retrieve the resource after all redirection.

The value of these properties does not reflect any URI escaping that may be required for retrieval of the resource, but it may include escaped characters if these were specified in the document, or returned by a server in the case of redirection.

In some cases (such as a document read from a string or a pipe) the rules in [XML Base] may result in a base URI being application dependent. In these cases this specification does not define the value of the [base URI] or [declaration base URI] property.

When resolving relative URIs the [base URI] property should be used in preference to the values of xml:base attributes; they may be inconsistent in the case of Synthetic Infosets.

``Unknown'' and ``No Value''

Some properties may sometimes have the value ***unknown*** or ***no value***, and it is said that a property value is unknown or that a property has no value respectively. These values are distinct from each other and from all other values. In particular they are distinct from the empty string, the empty set, and the empty list, each of which simply has no members. This specification does not use the term **null** since in some communities it has particular connotations which may not match those intended here.

Inconsistencies Resulting from Invalidity

As noted above, an XML document need not be valid to have an information set. However, certain kinds of invalidity affect the values assigned to some properties. Entities, notations, elements and attributes may be undeclared. Notations and elements may be multiply declared (multiple declarations are valid for entities and attributes). An ID may be undefined or multiply defined. Such cases are noted where relevant in the Information Item definitions below.

Synthetic Infosets

This specification describes the information set resulting from parsing an XML document. Information sets may be constructed by other means, for example by use of an API such as the DOM or by transforming an existing information set.

An information set corresponding to a real document will necessarily be consistent in various ways; for example the [in-scope namespaces] property of an element will be consistent with the [namespace attributes] properties of the element and its ancestors. This may not be true of an information set constructed by other means; in such a case there will be no XML document corresponding to the information set, and to serialize it will require resolution of the inconsistencies (for example, by outputting namespace declarations that correspond to the namespaces in scope).

2. Information Items

An information set can contain up to eleven different types of information item, as explained in the following sections. Every information item has properties. For ease of reference, each property is given a name, indicated **[thus]**. Links to a definition and/or syntax in the XML 1.0 Recommendation [XML] are given for each information item.

2.1. The Document Information Item

XML Definition: *document (Section 2, Documents)*

XML Syntax: *[1] Document (Section 2.1, Well-Formed XML Documents)*

There is exactly one ***document information item*** in the information set, and all other information items are accessible from the properties of the document information item, either directly or indirectly through the properties of other information items.

The document information item has the following properties:

1. **[children]** An ordered list of child information items, in document order. The list contains exactly one element information item. The list also contains one processing instruction information item for each processing instruction outside the document element, and one comment information item for each comment outside the document element. Processing instructions and comments within the DTD are excluded. If there is a document type declaration, the list also contains a document type declaration information item.
2. **[document element]** The element information item corresponding to the document element.
3. **[notations]** An unordered set of notation information items, one for each notation declared in the DTD. If any notation is multiply declared, this property has no value.
4. **[unparsed entities]** An unordered set of unparsed entity information items, one for each unparsed entity declared in the DTD.
5. **[base URI]** The base URI of the document entity.
6. **[character encoding scheme]** The name of the character encoding scheme in which the document entity is expressed.
7. **[standalone]** An indication of the standalone status of the document, either yes or no. This property is derived from the optional standalone document declaration in the XML declaration at the beginning of the document entity, and has no value if there is no standalone document declaration.
8. **[version]** A string representing the XML version of the document. This property is derived from the XML declaration optionally present at the beginning of the document entity, and has no value if there is no XML declaration.
9. **[all declarations processed]** This property is not strictly speaking part of the infoset of the document. Rather it is an indication of whether the processor has read the complete DTD. Its value is a boolean. If it is false, then certain properties (indicated in their descriptions below) may be unknown. If it is true, those properties are never unknown.

2.2. Element Information Items

XML Definition: *element (Section 3, Logical Structures)*

XML Syntax: *[39] Element (Section 3, Logical Structures)*

There is an ***element information item*** for each element appearing in the XML document. One of the element information items is the value of the [document element] property of the document information item, corresponding to the root of the element tree, and all other element information items are

accessible by recursively following its [children] property.

An element information item has the following properties:

1. **[namespace name]** The namespace name, if any, of the element type. If the element does not belong to a namespace, this property has no value.
2. **[local name]** The local part of the element-type name. This does not include any namespace prefix or following colon.
3. **[prefix]** The namespace prefix part of the element-type name. If the name is unprefixed, this property has no value. Note that namespace-aware applications should use the namespace name rather than the prefix to identify elements.
4. **[children]** An ordered list of child information items, in document order. This list contains element, processing instruction, unexpanded entity reference, character, and comment information items, one for each element, processing instruction, reference to an unprocessed external entity, data character, and comment appearing immediately within the current element. If the element is empty, this list has no members.
5. **[attributes]** An unordered set of attribute information items, one for each of the attributes (specified or defaulted from the DTD) of this element. Namespace declarations do not appear in this set. If the element has no attributes, this set has no members.
6. **[namespace attributes]** An unordered set of attribute information items, one for each of the namespace declarations (specified or defaulted from the DTD) of this element. Declarations of the form xmlns="" and xmlns:name="", which undeclare the default namespace and prefixes respectively, count as namespace declarations. Prefix undeclaration was added in Namespaces in XML 1.1. By definition, all namespace attributes (including those named `xmlns`, whose [prefix] property has no value) have a namespace URI of `http://www.w3.org/2000/xmlns/`. If the element has no namespace declarations, this set has no members.
7. **[in-scope namespaces]** An unordered set of namespace information items, one for each of the namespaces in effect for this element. This set always contains an item with the prefix `xml` which is implicitly bound to the namespace name `http://www.w3.org/XML/1998/namespace`. It does not contain an item with the prefix `xmlns` (used for declaring namespaces), since an application can never encounter an element or attribute with that prefix. The set will include namespace items corresponding to all of the members of [namespace attributes], except for any representing declarations of the form xmlns="" or xmlns:name="", which do not declare a namespace but rather undeclare the default namespace and prefixes. When resolving the prefixes of qualified names this property should be used in preference to the

[namespace attributes] property; they may be inconsistent in the case of Synthetic Infosets.

8. **[base URI]** The base URI of the element.
9. **[parent]** The document or element information item which contains this information item in its [children] property.

2.3. Attribute Information Items

XML Definition: *attribute (Section 3.1, Start-Tags, End-Tags, and Empty-Element Tags)*

XML Syntax: *[41] Attribute (Section 3.1, Start-Tags, End-Tags, and Empty-Element Tags)*

There is an ***attribute information item*** for each attribute (specified or defaulted) of each element in the document, including those which are namespace declarations. The latter however appear as members of an element's [namespace attributes] property rather than its [attributes] property.

Attributes declared in the DTD with no default value and not specified in the element's start tag are not represented by attribute information items.

An attribute information item has the following properties:

1. **[namespace name]** The namespace name, if any, of the attribute. Otherwise, this property has no value.
2. **[local name]** The local part of the attribute name. This does not include any namespace prefix or following colon.
3. **[prefix]** The namespace prefix part of the attribute name. If the name is unprefixed, this property has no value. Note that namespace-aware applications should use the namespace name rather than the prefix to identify attributes.
4. **[normalized value]** The normalized attribute value (see 3.3.3 Attribute-Value Normalization [XML]).
5. **[specified]** A flag indicating whether this attribute was actually specified in the start-tag of its element, or was defaulted from the DTD.
6. **[attribute type]** An indication of the type declared for this attribute in the DTD. Legitimate values are ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS, NOTATION, CDATA, and ENUMERATION. If there is no declaration for the attribute, this property has no value. If no declaration has been read, but the [all declarations processed] property of the document information item is false (so there may be an unread declaration), then the value of this property is unknown. Applications should treat no value and unknown as

equivalent to a value of CDATA. The value of this property is not affected by the validity of the attribute value.

7. **[references]** If the attribute type is ID, NMTOKEN, NMTOKENS, CDATA, or ENUMERATION, this property has no value. If the attribute type is unknown, the value of this property is unknown. Otherwise (that is, if the attribute type is IDREF, IDREFS, ENTITY, ENTITIES, or NOTATION), the value of this property is an ordered list of the element, unparsed entity, or notation information items referred to in the attribute value, in the order that they appear there. In this case, if the attribute value is syntactically invalid, this property has no value. If the type is IDREF or IDREFS and any of the IDs does not appear as the value of an ID attribute in the document, or if the type is ENTITY, ENTITIES or NOTATION and no declaration has been read for any of the entities or the notation, then this property has no value or is unknown, depending on whether the [all declarations processed] property of the document information item is true or false. If the type is IDREF or IDREFS and any of the IDs appears as the value of more than one ID attribute in the document, or if the type is NOTATION and there are multiple declarations for the notation, then this property has no value.
8. **[owner element]** The element information item which contains this information item in its [attributes] property.

2.4. Processing Instruction Information Items

XML Definition: *processing instruction (Section 2.6, Processing Instructions)*

XML Syntax: *[16] PI (Section 2.6, Processing Instructions)*

There is a ***processing instruction information item*** for each processing instruction in the document. The XML declaration and text declarations for external parsed entities are not considered processing instructions.

A processing instruction information item has the following properties:

1. **[target]** A string representing the target part of the processing instruction (an XML name).
2. **[content]** A string representing the content of the processing instruction, excluding the target and any white space immediately following it. If there is no such content, the value of this property will be an empty string.
3. **[base URI]** The base URI of the PI. Note that if an infoset is serialized as an XML document, it will not be possible to preserve the base URI of any PI that originally appeared at the top level of an external entity, since there is no syntax for PIs corresponding to the `xml:base`

attribute on elements.

4. **[notation]** The notation information item named by the target. If there is no declaration for a notation with that name, or there are multiple declarations, this property has no value. If no declaration has been read, but the [all declarations processed] property of the document information item is false (so there may be an unread declaration), then the value of this property is unknown.
5. **[parent]** The document, element, or document type declaration information item which contains this information item in its [children] property.

2.5. Unexpanded Entity Reference Information Items

XML Definition: *Section 4.4.3, Included If Validating*

A ***unexpanded entity reference information item*** serves as a placeholder by which an XML processor can indicate that it has not expanded an external parsed entity. There is such an information item for each unexpanded reference to an external general entity within the content of an element. A validating XML processor, or a non-validating processor that reads all external general entities, will never generate unexpanded entity reference information items for a valid document.

An unexpanded entity reference information item has the following properties:

1. **[name]** The name of the entity referenced.
2. **[system identifier]** The system identifier of the entity, as it appears in the declaration of the entity, without any additional URI escaping applied by the processor. If there is no declaration for the entity, this property has no value. If no declaration has been read, but the [all declarations processed] property of the document information item is false (so there may be an unread declaration), then the value of this property is unknown.
3. **[public identifier]** The public identifier of the entity, normalized as described in 4.2.2 External Entities [XML]. If there is no declaration for the entity, or the declaration does not include a public identifier, this property has no value. If no declaration has been read, but the [all declarations processed] property of the document information item is false (so there may be an unread declaration), then the value of this property is unknown.
4. **[declaration base URI]** The base URI relative to which the system identifier should be resolved (i.e. the base URI of the resource within which the entity declaration occurs). This is unknown or has no value in the same circumstances as the [system identifier] property.
5. **[parent]** The element information item which contains this information

item in its [children] property.

2.6. Character Information Items

XML Syntax: *[2] Char (Section 2.2, Characters)*

There is a ***character information item*** for each data character that appears in the document, whether literally, as a character reference, or within a CDATA section.

Each character is a logically separate information item, but XML applications are free to chunk characters into larger groups as necessary or desirable.

A character information item has the following properties:

1. **[character code]** The ISO 10646 character code (in the range 0 to #x10FFFF, though not every value in this range is a legal XML character code) of the character.
2. **[element content whitespace]** A boolean indicating whether the character is white space appearing within element content (see [XML], 2.10 "White Space Handling"). Note that validating XML processors are *required* to provide this information. If there is no declaration for the containing element, or there are multiple declarations, this property has no value for white space characters. If no declaration has been read, but the [all declarations processed] property of the document information item is false (so there may be an unread declaration), then the value of this property is unknown for white space characters. It is always false for characters that are not white space.
3. **[parent]** The element information item which contains this information item in its [children] property.

2.7. Comment Information Items

XML Definition: *comment (Section 2.5, Comments)*

XML Syntax: *[15] Comment (Section 2.5, Comments)*

There is a ***comment information item*** for each XML comment in the original document, except for those appearing in the DTD (which are not represented).

A comment information item has the following properties:

1. **[content]** A string representing the content of the comment.
2. **[parent]** The document or element information item which contains this

information item in its [children] property.

2.8. The Document Type Declaration Information Item

XML Definition: *document type declaration (section 2.8, Prolog and Document Type Declaration)*

XML Syntax: *[28] doctypedecl (section 2.8, Prolog and Document Type Declaration)*

If the XML document has a document type declaration, then the information set contains a single ***document type declaration information item***. Note that entities and notations are provided as properties of the document information item, not the document type declaration information item.

A document type declaration information item has the following properties:

1. **[system identifier]** The system identifier of the external subset, as it appears in the DOCTYPE declaration, without any additional URI escaping applied by the processor. If there is no external subset this property has no value.
2. **[public identifier]** The public identifier of the external subset, normalized as described in 4.2.2 External Entities [XML]. If there is no external subset or if it has no public identifier, this property has no value.
3. **[children]** An ordered list of processing instruction information items representing processing instructions appearing in the DTD, in the original document order. Items from the internal DTD subset appear before those in the external subset.
4. **[parent]** The document information item.

2.9. Unparsed Entity Information Items

XML Definition: *entity (section 4, Physical Structures)*

XML Syntax: *[71] GEDecl (section 4.2, Entities)*

There is an ***unparsed entity information item*** for each unparsed general entity declared in the DTD.

An unparsed entity information item has the following properties:

1. **[name]** The name of the entity.

2. **[system identifier]** The system identifier of the entity, as it appears in the declaration of the entity, without any additional URI escaping applied by the processor.
3. **[public identifier]** The public identifier of the entity, normalized as described in 4.2.2 External Entities [XML]. If the entity has no public identifier, this property has no value.
4. **[declaration base URI]** The base URI relative to which the system identifier should be resolved (i.e. the base URI of the resource within which the entity declaration occurs).
5. **[notation name]** The notation name associated with the entity.
6. **[notation]** The notation information item named by the notation name. If there is no declaration for a notation with that name, or there are multiple declarations, this property has no value. If no declaration has been read, but the [all declarations processed] property of the document information item is false (so there may be an unread declaration), then the value of this property is unknown.

2.10. Notation Information Items

XML Definition: *notation (section 4.7, Notations)*

XML Syntax: *[82] NotationDecl (section 4.7, Notations)*

There is a ***notation information item*** for each notation declared in the DTD.

A notation information item has the following properties:

1. **[name]** The name of the notation.
2. **[system identifier]** The system identifier of the notation, as it appears in the declaration of the notation, without any additional URI escaping applied by the processor. If no system identifier was specified, this property has no value.
3. **[public identifier]** The public identifier of the notation, normalized as described in 4.2.2 External Entities [XML]. If the notation has no public identifier, this property has no value.
4. **[declaration base URI]** The base URI relative to which the system identifier should be resolved (i.e. the base URI of the resource within which the notation declaration occurs).

2.11. Namespace Information Items

Each element in the document has a ***namespace information item*** for each namespace that is in scope for that element.

A namespace information item has the following properties:

1. **[prefix]** The prefix whose binding this item describes. Syntactically, this is the part of the attribute name following the `xmlns:` prefix. If the attribute name is simply `xmlns`, so that the declaration is of the default namespace, this property has no value.
2. **[namespace name]** The namespace name to which the prefix is bound.

3. Conformance

Since the purpose of the Information Set is to provide a set of definitions, conformance is a property of specifications that use those definitions, rather than of implementations.

Specifications referring to the Infoset must:

- Indicate the information items and properties that are needed to implement the specification. (This indirectly imposes conformance requirements on processors used to implement the specification.)
- Specify how other information items and properties are treated (for example, they might be passed through unchanged).
- Note any information required from an XML document that is not defined by the Infoset.
- Note any difference in the use of terms defined by the Infoset (this should be avoided).

If a specification allows the construction of an infoset that has inconsistencies as described above under Synthetic Infosets it may describe how those inconsistencies are to be resolved, and should do so if it provides for serialization of the infoset.

Appendix A. References

Normative References

ISO/IEC 10646

ISO (International Organization for Standardization). *ISO/IEC 10646-1:2000. Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane* and *ISO/IEC 10646-2:2001.Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 2: Supplementary Planes*, as, from time to time, amended, replaced by a new edition or expanded by the addition of new parts. [Geneva]: International Organization for Standardization. (See http://www.iso.ch for the latest

version.)

Namespaces

Namespaces in XML, W3C, eds. Tim Bray, Dave Hollander, Andrew Layman. 14 January 1999. Available at http://www.w3.org/TR/REC-xml-names.

Namespaces 1.1

Namespaces in XML 1.1, W3C, eds. Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin. 4 February 2004. Available at http://www.w3.org/TR/xml-names11.

RFC2119

Key words for use in RFCs to Indicate Requirement Levels, ed. S. Bradner. March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt.

XML

Extensible Markup Language (XML) 1.0 (Third Edition), W3C, eds. Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, François Yergeau. 4 February 2004. Available at http://www.w3.org/TR/REC-xml.

XML 1.1

Extensible Markup Language (XML) 1.1, W3C, eds. Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, John Cowan, François Yergeau. 4 February 2004. Available at http://www.w3.org/TR/xml11.

XML Base

XML Base, W3C, ed. Jonathan Marsh. February 2000. Available at http://www.w3.org/TR/xmlbase.

Informative References

DOM

Document Object Model (DOM) Level 1 Specification, W3C, eds. Vidur Apparao, Steve Byrne, Mike Champion, et al. 1 October 1998. Available at http://www.w3.org/TR/REC-DOM-Level-1.

XPointer-Liaison

XPointer-Information Set Liaison Statement, W3C, ed. Steven J. DeRose. 24 February 1999. Available at http://www.w3.org/TR/NOTE-xptr-infoset-liaison.

Relative Namespace URI References

Results of W3C XML Plenary Ballot on relative URI References in namespace declarations, 3-17 July 2000, W3C, eds. Dave Hollander, C. M. Sperberg-McQueen. 6 September 2000. Available at http://www.w3.org/2000/09/xppa.

RDF Schema for the XML Information Set

RDF Schema for the XML Information Set, W3C, ed. Richard Tobin. 6

April 2001. Available at http://www.w3.org/TR/xml-infoset-rdfs.

Appendix B: XML Reporting Requirements (informative)

Although the XML Recommendation [XML] is primarily concerned with XML syntax, it also includes some specific reporting requirements for XML processors.

The reporting requirements include errors, which are outside the scope of this specification, and document information. All of the XML requirements for document information reporting have been integrated into the XML Information Set; numbers in parentheses refer to sections of the XML Recommendation:

1. An XML processor must always provide all characters in a document that are not part of markup to the application (2.10).
2. A validating XML processor must inform the application which of the character data in a document is white space appearing within element content (2.10).
3. An XML processor must normalize line-ends to LF before passing them to the application (2.11).
4. An XML processor must normalize the value of attributes according to the rules in clause 3.3.3 before passing them to the application.
5. An XML processor must pass the names and external identifiers (system identifiers, public identifiers or both) of declared notations to the application (4.7).
6. When the name of an unparsed entity appears as the explicit or default value of an ENTITY or ENTITIES attribute, an XML processor must provide the names, system identifiers, and (if present) public identifiers of both the entity and its notation to the application (4.6, 4.7).
7. An XML processor must pass processing instructions to the application (2.6).
8. An XML processor (necessarily a non-validating one) that does not include the replacement text of an external parsed entity in place of an entity reference must notify the application that it recognized but did not read the entity (4.4.3).
9. A validating XML processor must include the replacement text of an entity in place of an entity reference (5.2).
10. An XML processor must supply the default value of attributes declared in the DTD for a given element type but not appearing in the element's start tag (3.3.2).

Appendix C: Example (informative)

Consider the following example XML document:

```
<?xml version="1.0"?>

<msg:message doc:date="19990421"
                 xmlns:doc="http://doc.example.org/
namespaces/doc"
                 xmlns:msg="http://message.example.
org/"
>Phone home!</msg:message>
```

The information set for this XML document contains the following information items:

- A document information item.
- An element information item with namespace name "`http://message.example.org/`", local part "`message`", and prefix "`msg`".
- An attribute information item with the namespace name "`http://doc.example.org/namespaces/doc`", local part "`date`", prefix "`doc`", and normalized value "`19990421`".
- Three namespace information items for the `http://www.w3.org/XML/1998/namespace`, `http://doc.example.org/namespaces/doc`, and `http://message.example.org/` namespaces.
- Two attribute information items for the namespace attributes.
- Eleven character information items for the character data.

Appendix D: What is not in the Information Set

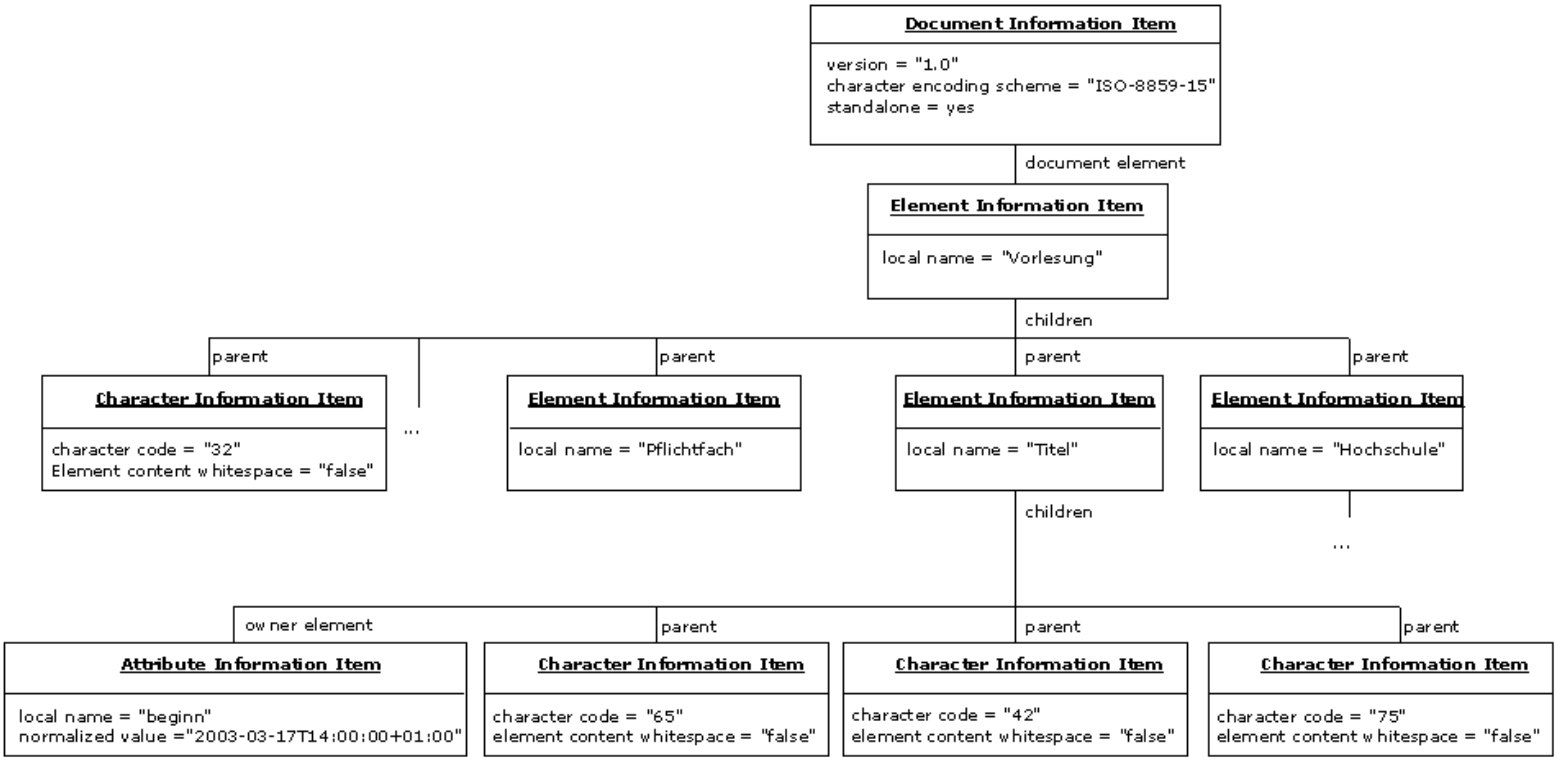
The following information is not represented in the current version of the XML Information Set (this list is not intended to be exhaustive):

1. The content models of elements, from ELEMENT declarations in the DTD.
2. The grouping and ordering of attribute declarations in ATTLIST declarations.
3. The document type name.
4. White space outside the document element.
5. White space immediately following the target name of a PI.
6. Whether characters are represented by character references.
7. The difference between the two forms of an empty element: `<foo/>` and `<foo></foo>`.
8. White space within start-tags (other than significant white space in

attribute values) and end-tags.
9. The difference between CR, CR-LF, and LF line termination.
10. The order of attributes within a start-tag.
11. The order of declarations within the DTD.
12. The boundaries of conditional sections in the DTD.
13. The boundaries of parameter entities in the DTD.
14. Comments in the DTD.
15. The location of declarations (whether in internal or external subset or parameter entities).
16. Any ignored declarations, including those within an IGNORE conditional section, as well as entity and attribute declarations ignored because previous declarations override them.
17. The kind of quotation marks (single or double) used to quote attribute values.
18. The boundaries of general parsed entities.
19. The boundaries of CDATA marked sections.
20. The default value of attributes declared in the DTD.

Appendix E: RDF Schema (informative)

See RDF Schema for the XML Information Set for a formal characterization of the Infoset.


Document Information Item
version = "1.0"
character encoding scheme = "ISO-8859-15"
standalone = yes
document element
Element Information Item
local name = "Vorlesung"
children
parent
parent
parent
parent
Character Information Item
character code = "32"
Element content whitespace = "false"
...
Element Information Item
local name = "Pflichtfach"
Element Information Item
local name = "Titel"
Element Information Item
local name = "Hochschule"
children
...
owner element
parent
parent
parent
Attribute Information Item
local name = "beginn"
normalized value ="2003-03-17T14:00:00+01:00"
Character Information Item
character code = "65"
element content whitespace = "false"
Character Information Item
character code = "42"
element content whitespace = "false"
Character Information Item
character code = "75"
element content whitespace = "false"

Network Working Group
Request for Comments: 2396
Updates: 1808, 1738
Category: Standards Track

T. Berners-Lee
MIT/LCS
R. Fielding
U.C. Irvine
L. Masinter
Xerox Corporation

August 1998

Uniform Resource Identifiers (URI): Generic Syntax

Status of this Memo

Copyright Notice

IESG Note

This paper describes a "superset" of operations that can be applied to URI. It consists of both a grammar and a description of basic functionality for URI. To understand what is a valid URI, both the grammar and the associated description have to be studied. Some of the functionality described is not applicable to all URI schemes, and some operations are only possible when certain media types are retrieved using the URI, regardless of the scheme used.

Abstract

A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource. This document defines the generic syntax of URI, including both absolute and relative forms, and guidelines for their use; it revises and replaces the generic definitions in RFC 1738 and RFC 1808.

This document defines a grammar that is a superset of all valid URI, such that an implementation can parse the common components of a URI reference without knowing the scheme-specific requirements of every possible identifier type. This document does not define a generative grammar for URI; that task will be performed by the individual specifications of each URI scheme.

1. Introduction

Uniform Resource Identifiers (URI) provide a simple and extensible means for identifying a resource. This specification of URI syntax and semantics is derived from concepts introduced by the World Wide Web global information initiative, whose use of such objects dates from 1990 and is described in "Universal Resource Identifiers in WWW" [RFC1630]. The specification of URI is designed to meet the recommendations laid out in "Functional Recommendations for Internet Resource Locators" [RFC1736] and "Functional Requirements for Uniform Resource Names" [RFC1737].

This document updates and merges "Uniform Resource Locators" [RFC1738] and "Relative Uniform Resource Locators" [RFC1808] in order to define a single, generic syntax for all URI. It excludes those portions of RFC 1738 that defined the specific syntax of individual URL schemes; those portions will be updated as separate documents, as will the process for registration of new URI schemes. This document does not discuss the issues and recommendation for dealing with characters outside of the US-ASCII character set [ASCII]; those recommendations are discussed in a separate document.

All significant changes from the prior RFCs are noted in Appendix G.

1.1 Overview of URI

URI are characterized by the following definitions:

Uniform
Uniformity provides several benefits: it allows different types

of resource identifiers to be used in the same context, even when the mechanisms used to access those resources may differ; it allows uniform semantic interpretation of common syntactic conventions across different types of resource identifiers; it allows introduction of new types of resource identifiers without interfering with the way that existing identifiers are used; and, it allows the identifiers to be reused in many different contexts, thus permitting new applications or protocols to leverage a pre-existing, large, and widely-used set of resource identifiers.

Resource
A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources.

The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content---the entities to which it currently corresponds---changes over time, provided that the conceptual mapping is not changed in the process.

Identifier
An identifier is an object that can act as a reference to something that has identity. In the case of URI, the object is a sequence of characters with a restricted syntax.

Having identified a resource, a system may perform a variety of operations on the resource, as might be characterized by such words as `access', `update', `replace', or `find attributes'.

1.2. URI, URL, and URN

A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URI

that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource. The term "Uniform Resource Name" (URN) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

The URI scheme (Section 3.1) defines the namespace of the URI, and thus may further restrict the syntax and semantics of identifiers using that scheme. This specification defines those elements of the URI syntax that are either required of all URI schemes or are common to many URI schemes. It thus defines the syntax and semantics that are needed to implement a scheme-independent parsing mechanism for URI references, such that the scheme-dependent handling of a URI can be postponed until the scheme-dependent semantics are needed. We use the term URL below when describing syntax or semantics that only apply to locators.

Although many URL schemes are named after protocols, this does not imply that the only way to access the URL's resource is via the named protocol. Gateways, proxies, caches, and name resolution services might be used to access some resources, independent of the protocol of their origin, and the resolution of some URL may require the use of more than one protocol (e.g., both DNS and HTTP are typically used to access an "http" URL's resource when it can't be found in a local cache).

A URN differs from a URL in that it's primary purpose is persistent labeling of a resource with an identifier. That identifier is drawn from one of a set of defined namespaces, each of which has its own set name structure and assignment procedures. The "urn" scheme has been reserved to establish the requirements for a standardized URN namespace, as defined in "URN Syntax" [RFC2141] and its related specifications.

Most of the examples in this specification demonstrate URL, since they allow the most varied use of the syntax and often have a

hierarchical namespace. A parser of the URI syntax is capable of parsing both URL and URN references as a generic URI; once the scheme is determined, the scheme-specific parsing can be performed on the generic URI components. In other words, the URI syntax is a superset of the syntax of all URI schemes.

1.3. Example URI

The following examples illustrate URI that are in common use.

ftp://ftp.is.co.za/rfc/rfc1808.txt
-- ftp scheme for File Transfer Protocol services

gopher://spinaltap.micro.umn.edu/00/Weather/California/Los%20Angeles
-- gopher scheme for Gopher and Gopher+ Protocol services

http://www.math.uio.no/faq/compression-faq/part1.html
-- http scheme for Hypertext Transfer Protocol services

mailto:mduerst@ifi.unizh.ch
-- mailto scheme for electronic mail addresses

news:comp.infosystems.www.servers.unix
-- news scheme for USENET news groups and articles

telnet://melvyl.ucop.edu/
-- telnet scheme for interactive services via the TELNET Protocol

1.4. Hierarchical URI and Relative Forms

An absolute identifier refers to a resource independent of the context in which the identifier is used. In contrast, a relative identifier refers to a resource by describing the difference within a hierarchical namespace between the current context and an absolute identifier of the resource.

Some URI schemes support a hierarchical naming system, where the hierarchy of the name is denoted by a "/" delimiter separating the components in the scheme. This document defines a scheme-independent `relative' form of URI reference that can be used in conjunction with a `base' URI (of a hierarchical scheme) to produce another URI. The syntax of hierarchical URI is described in Section 3; the relative URI calculation is described in Section 5.

1.5. URI Transcribability

The URI syntax was designed with global transcribability as one of its main concerns. A URI is a sequence of characters from a very limited set, i.e. the letters of the basic Latin alphabet, digits, and a few special characters. A URI may be represented in a variety of ways: e.g., ink on paper, pixels on a screen, or a sequence of octets in a coded character set. The interpretation of a URI depends only on the characters used and not how those characters are represented in a network protocol.

The goal of transcribability can be described by a simple scenario. Imagine two colleagues, Sam and Kim, sitting in a pub at an international conference and exchanging research ideas. Sam asks Kim for a location to get more information, so Kim writes the URI for the research site on a napkin. Upon returning home, Sam takes out the napkin and types the URI into a computer, which then retrieves the information to which Kim referred.

There are several design concerns revealed by the scenario:

- A URI is a sequence of characters, which is not always represented as a sequence of octets.

- A URI may be transcribed from a non-network source, and thus should consist of characters that are most likely to be able to be typed into a computer, within the constraints imposed by keyboards (and related input devices) across languages and locales.

- A URI often needs to be remembered by people, and it is easier for people to remember a URI when it consists of meaningful components.

These design concerns are not always in alignment. For example, it is often the case that the most meaningful name for a URI component would require characters that cannot be typed into some systems. The ability to transcribe the resource identifier from one medium to

another was considered more important than having its URI consist of the most meaningful of components. In local and regional contexts

and with improving technology, users might benefit from being able to use a wider range of characters; such use is not defined in this document.

1.6. Syntax Notation and Common Elements

This document uses two conventions to describe and define the syntax for URI. The first, called the layout form, is a general description of the order of components and component separators, as in

<first>/<second>;<third>?<fourth>

The component names are enclosed in angle-brackets and any characters outside angle-brackets are literal separators. Whitespace should be ignored. These descriptions are used informally and do not define the syntax requirements.

The second convention is a BNF-like grammar, used to define the formal URI syntax. The grammar is that of [RFC822], except that "|" is used to designate alternatives. Briefly, rules are separated from definitions by an equal "=", indentation is used to continue a rule definition over more than one line, literals are quoted with "", parentheses "(" and ")" are used to group elements, optional elements are enclosed in "[" and "]" brackets, and elements may be preceded with <n>* to designate n or more repetitions of the following element; n defaults to 0.

Unlike many specifications that use a BNF-like grammar to define the bytes (octets) allowed by a protocol, the URI grammar is defined in terms of characters. Each literal in the grammar corresponds to the character it represents, rather than to the octet encoding of that character in any particular coded character set. How a URI is represented in terms of bits and bytes on the wire is dependent upon the character encoding of the protocol used to transport it, or the charset of the document which contains it.

The following definitions are common to many elements:

```
alpha    = lowalpha | upalpha

lowalpha = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
           "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
           "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

upalpha  = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
           "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
           "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
```



```
digit    = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
           "8" | "9"

alphanum = alpha | digit
```

The complete URI syntax is collected in Appendix A.

2. URI Characters and Escape Sequences

URI consist of a restricted set of characters, primarily chosen to aid transcribability and usability both in computer systems and in non-computer communications. Characters used conventionally as delimiters around URI were excluded. The restricted set of characters consists of digits, letters, and a few graphic symbols were chosen from those common to most of the character encodings and input facilities available to Internet users.

```
uric         = reserved | unreserved | escaped
```

Within a URI, characters are either used as delimiters, or to represent strings of data (octets) within the delimited portions. Octets are either represented directly by a character (using the US-ASCII character for that octet [ASCII]) or by an escape encoding. This representation is elaborated below.

2.1 URI and non-ASCII characters

The relationship between URI and characters has been a source of confusion for characters that are not part of US-ASCII. To describe the relationship, it is useful to distinguish between a "character" (as a distinguishable semantic entity) and an "octet" (an 8-bit byte). There are two mappings, one from URI characters to octets, and a second from octets to original characters:

URI character sequence->octet sequence->original character sequence

A URI is represented as a sequence of characters, not as a sequence of octets. That is because URI might be "transported" by means that are not through a computer network, e.g., printed on paper, read over the radio, etc.

A URI scheme may define a mapping from URI characters to octets; whether this is done depends on the scheme. Commonly, within a delimited component of a URI, a sequence of characters may be used to represent a sequence of octets. For example, the character "a" represents the octet 97 (decimal), while the character sequence "%", "0", "a" represents the octet 10 (decimal).

There is a second translation for some resources: the sequence of octets defined by a component of the URI is subsequently used to represent a sequence of characters. A 'charset' defines this mapping. There are many charsets in use in Internet protocols. For example, UTF-8 [UTF-8] defines a mapping from sequences of octets to sequences of characters in the repertoire of ISO 10646.

In the simplest case, the original character sequence contains only characters that are defined in US-ASCII, and the two levels of mapping are simple and easily invertible: each 'original character' is represented as the octet for the US-ASCII code for it, which is, in turn, represented as either the US-ASCII character, or else the "%" escape sequence for that octet.

For original character sequences that contain non-ASCII characters, however, the situation is more difficult. Internet protocols that

transmit octet sequences intended to represent character sequences are expected to provide some way of identifying the charset used, if there might be more than one [RFC2277]. However, there is currently no provision within the generic URI syntax to accomplish this identification. An individual URI scheme may require a single charset, define a default charset, or provide a way to indicate the charset used.

It is expected that a systematic treatment of character encoding within URI will be developed as a future modification of this specification.

2.2. Reserved Characters

Many URI include components consisting of or delimited by, certain special characters. These characters are called "reserved", since their usage within the URI component is limited to their reserved purpose. If the data for a URI component would conflict with the reserved purpose, then the conflicting data must be escaped before forming the URI.

```
reserved    = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
              "$" | ","
```

The "reserved" syntax class above refers to those characters that are allowed within a URI, but which may not be allowed within a particular component of the generic URI syntax; they are used as delimiters of the components described in Section 3.

Characters in the "reserved" set are not reserved in all contexts. The set of characters actually reserved within any given URI component is defined by that component. In general, a character is reserved if the semantics of the URI changes if the character is replaced with its escaped US-ASCII encoding.

2.3. Unreserved Characters

Data characters that are allowed in a URI but do not have a reserved purpose are called unreserved. These include upper and lower case letters, decimal digits, and a limited set of punctuation marks and symbols.

```
unreserved  = alphanum | mark

mark        = "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
```

Unreserved characters can be escaped without changing the semantics of the URI, but this should not be done unless the URI is being used in a context that does not allow the unescaped character to appear.

2.4. Escape Sequences

Data must be escaped if it does not have a representation using an unreserved character; this includes data that does not correspond to a printable character of the US-ASCII coded character set, or that corresponds to any US-ASCII character that is disallowed, as explained below.

2.4.1. Escaped Encoding

An escaped octet is encoded as a character triplet, consisting of the percent character "%" followed by the two hexadecimal digits representing the octet code. For example, "%20" is the escaped encoding for the US-ASCII space character.

```
escaped     = "%" hex hex
hex         = digit | "A" | "B" | "C" | "D" | "E" | "F" |
                  "a" | "b" | "c" | "d" | "e" | "f"
```

2.4.2. When to Escape and Unescape

A URI is always in an "escaped" form, since escaping or unescaping a completed URI might change its semantics. Normally, the only time escape encodings can safely be made is when the URI is being created from its component parts; each component may have its own set of characters that are reserved, so only the mechanism responsible for generating or interpreting that component can determine whether or

not escaping a character will change its semantics. Likewise, a URI must be separated into its components before the escaped characters within those components can be safely decoded.

In some cases, data that could be represented by an unreserved character may appear escaped; for example, some of the unreserved "mark" characters are automatically escaped by some systems. If the given URI scheme defines a canonicalization algorithm, then unreserved characters may be unescaped according to that algorithm. For example, "%7e" is sometimes used instead of "~" in an http URL path, but the two are equivalent for an http URL.

Because the percent "%" character always has the reserved purpose of being the escape indicator, it must be escaped as "%25" in order to be used as data within a URI. Implementers should be careful not to escape or unescape the same string more than once, since unescaping an already unescaped string might lead to misinterpreting a percent data character as another escaped character, or vice versa in the case of escaping an already escaped string.

2.4.3. Excluded US-ASCII Characters

Although they are disallowed within the URI syntax, we include here a description of those US-ASCII characters that have been excluded and the reasons for their exclusion.

The control characters in the US-ASCII coded character set are not used within a URI, both because they are non-printable and because they are likely to be misinterpreted by some control mechanisms.

control = <US-ASCII coded characters 00-1F and 7F hexadecimal>

The space character is excluded because significant spaces may disappear and insignificant spaces may be introduced when URI are transcribed or typeset or subjected to the treatment of word-processing programs. Whitespace is also used to delimit URI in many contexts.

space = <US-ASCII coded character 20 hexadecimal>

The angle-bracket "<" and ">" and double-quote (") characters are excluded because they are often used as the delimiters around URI in

text documents and protocol fields. The character "#" is excluded because it is used to delimit a URI from a fragment identifier in URI references (Section 4). The percent character "%" is excluded because it is used for the encoding of escaped characters.

delims = "<" | ">" | "#" | "%" | <">

Other characters are excluded because gateways and other transport agents are known to sometimes modify such characters, or they are used as delimiters.

unwise = "{" | "}" | "|" | "\" | "^" | "[" | "]" | "`"

Data corresponding to excluded characters must be escaped in order to be properly represented within a URI.

3. URI Syntactic Components

The URI syntax is dependent upon the scheme. In general, absolute URI are written as follows:

<scheme>:<scheme-specific-part>

An absolute URI contains the name of the scheme being used (<scheme>) followed by a colon (":") and then a string (the <scheme-specific-part>) whose interpretation depends on the scheme.

The URI syntax does not require that the scheme-specific-part have any general structure or set of semantics which is common among all URI. However, a subset of URI do share a common syntax for representing hierarchical relationships within the namespace. This "generic URI" syntax consists of a sequence of four main components:

<scheme>://<authority><path>?<query>

each of which, except <scheme>, may be absent from a particular URI. For example, some URI schemes do not allow an <authority> component, and others do not use a <query> component.

absoluteURI = scheme ":" (hier_part | opaque_part)

URI that are hierarchical in nature use the slash "/" character for separating hierarchical components. For some file systems, a "/" character (used to denote the hierarchical structure of a URI) is the delimiter used to construct a file name hierarchy, and thus the URI path will look similar to a file pathname. This does NOT imply that the resource is a file or that the URI maps to an actual filesystem pathname.

hier_part = (net_path | abs_path) ["?" query]

net_path = "//" authority [abs_path]

abs_path = "/" path_segments

URI that do not make use of the slash "/" character for separating hierarchical components are considered opaque by the generic URI parser.

opaque_part = uric_no_slash *uric

uric_no_slash = unreserved | escaped | ";" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","

We use the term <path> to refer to both the <abs_path> and <opaque_part> constructs, since they are mutually exclusive for any given URI and can be parsed as a single component.

3.1. Scheme Component

Just as there are many different methods of access to resources, there are a variety of schemes for identifying such resources. The URI syntax consists of a sequence of components separated by reserved characters, with the first component defining the semantics for the remainder of the URI string.

Scheme names consist of a sequence of characters beginning with a

lower case letter and followed by any combination of lower case letters, digits, plus ("+"), period ("."), or hyphen ("-"). For resiliency, programs interpreting URI should treat upper case letters as equivalent to lower case in scheme names (e.g., allow "HTTP" as well as "http").

```
scheme      = alpha *( alpha | digit | "+" | "-" | "." )
```

Relative URI references are distinguished from absolute URI in that they do not begin with a scheme name. Instead, the scheme is inherited from the base URI, as described in Section 5.2.

3.2. Authority Component

Many URI schemes include a top hierarchical element for a naming authority, such that the namespace defined by the remainder of the URI is governed by that authority. This authority component is typically defined by an Internet-based server or a scheme-specific registry of naming authorities.

```
authority    = server | reg_name
```

The authority component is preceded by a double slash "//" and is terminated by the next slash "/", question-mark "?", or by the end of the URI. Within the authority component, the characters ";", ":", "@", "?", and "/" are reserved.

An authority component is not required for a URI scheme to make use of relative references. A base URI without an authority component implies that any relative reference will also be without an authority component.

3.2.1. Registry-based Naming Authority

The structure of a registry-based naming authority is specific to the URI scheme, but constrained to the allowed characters for an authority component.

```
reg_name    = 1*( unreserved | escaped | "$" | "," |
```

";" | ":" | "@" | "&" | "=" | "+")

3.2.2. Server-based Naming Authority

URL schemes that involve the direct use of an IP-based protocol to a specified server on the Internet use a common syntax for the server component of the URI's scheme-specific data:

<userinfo>@<host>:<port>

where <userinfo> may consist of a user name and, optionally, scheme-specific information about how to gain authorization to access the server. The parts "<userinfo>@" and ":<port>" may be omitted.

server = [[userinfo "@"] hostport]

The user information, if present, is followed by a commercial at-sign "@".

userinfo = *(unreserved | escaped |
";" | ":" | "&" | "=" | "+" | "$" | ",")

Some URL schemes use the format "user:password" in the userinfo field. This practice is NOT RECOMMENDED, because the passing of authentication information in clear text (such as URI) has proven to be a security risk in almost every case where it has been used.

The host is a domain name of a network host, or its IPv4 address as a set of four decimal digit groups separated by ".". Literal IPv6 addresses are not supported.

hostport = host [":" port]
host = hostname | IPv4address
hostname = *(domainlabel ".") toplabel ["."]
domainlabel = alphanum | alphanum *(alphanum | "-") alphanum
toplabel = alpha | alpha *(alphanum | "-") alphanum

IPv4address = 1*digit "." 1*digit "." 1*digit "." 1*digit
port = *digit

Hostnames take the form described in Section 3 of [RFC1034] and Section 2.1 of [RFC1123]: a sequence of domain labels separated by ".", each domain label starting and ending with an alphanumeric character and possibly also containing "-" characters. The rightmost domain label of a fully qualified domain name will never start with a digit, thus syntactically distinguishing domain names from IPv4 addresses, and may be followed by a single "." if it is necessary to distinguish between the complete domain name and any local domain. To actually be "Uniform" as a resource locator, a URL hostname should be a fully qualified domain name. In practice, however, the host component may be a local domain literal.

> Note: A suitable representation for including a literal IPv6 address as the host part of a URL is desired, but has not yet been determined or implemented in practice.

The port is the network port number for the server. Most schemes designate protocols that have a default port number. Another port number may optionally be supplied, in decimal, separated from the host by a colon. If the port is omitted, the default port number is assumed.

3.3. Path Component

The path component contains data, specific to the authority (or the scheme if there is no authority component), identifying the resource within the scope of that scheme and authority.

```
path         = [ abs_path | opaque_part ]

path_segments = segment *( "/" segment )
segment       = *pchar *( ";" param )
param         = *pchar

pchar         = unreserved | escaped |
                ":" | "@" | "&" | "=" | "+" | "$" | ","
```

The path may consist of a sequence of path segments separated by a single slash "/" character. Within a path segment, the characters "/", ";", "=", and "?" are reserved. Each path segment may include a sequence of parameters, indicated by the semicolon ";" character. The parameters are not significant to the parsing of relative references.

3.4. Query Component

The query component is a string of information to be interpreted by the resource.

```
query       = *uric
```

Within a query component, the characters ";", "/", "?", ":", "@", "&", "=", "+", ",", and "$" are reserved.

4. URI References

The term "URI-reference" is used here to denote the common usage of a resource identifier. A URI reference may be absolute or relative, and may have additional information attached in the form of a fragment identifier. However, "the URI" that results from such a reference includes only the absolute URI after the fragment identifier (if any) is removed and after any relative URI is resolved to its absolute form. Although it is possible to limit the discussion of URI syntax and semantics to that of the absolute result, most usage of URI is within general URI references, and it is impossible to obtain the URI from such a reference without also parsing the fragment and resolving the relative form.

```
URI-reference = [ absoluteURI | relativeURI ] [ "#" fragment ]
```

The syntax for relative URI is a shortened form of that for absolute URI, where some prefix of the URI is missing and certain path components ("." and "..") have a special meaning when, and only when, interpreting a relative path. The relative URI syntax is defined in Section 5.

4.1. Fragment Identifier

When a URI reference is used to perform a retrieval action on the identified resource, the optional fragment identifier, separated from the URI by a crosshatch ("#") character, consists of additional reference information to be interpreted by the user agent after the

retrieval action has been successfully completed. As such, it is not part of a URI, but is often used in conjunction with a URI.

fragment = *uric

The semantics of a fragment identifier is a property of the data resulting from a retrieval action, regardless of the type of URI used in the reference. Therefore, the format and interpretation of fragment identifiers is dependent on the media type [RFC2046] of the retrieval result. The character restrictions described in Section 2

for URI also apply to the fragment in a URI-reference. Individual media types may define additional restrictions or structure within the fragment for specifying different types of "partial views" that can be identified within that media type.

A fragment identifier is only meaningful when a URI reference is intended for retrieval and the result of that retrieval is a document for which the identified fragment is consistently defined.

4.2. Same-document References

A URI reference that does not contain a URI is a reference to the current document. In other words, an empty URI reference within a document is interpreted as a reference to the start of that document, and a reference containing only a fragment identifier is a reference to the identified fragment of that document. Traversal of such a reference should not result in an additional retrieval action. However, if the URI reference occurs in a context that is always intended to result in a new request, as in the case of HTML's FORM element, then an empty URI reference represents the base URI of the current document and should be replaced by that URI when transformed into a request.

4.3. Parsing a URI Reference

A URI reference is typically parsed according to the four main components and fragment identifier in order to determine what components are present and whether the reference is relative or

absolute. The individual components are then parsed for their subparts and, if not opaque, to verify their validity.

Although the BNF defines what is allowed in each component, it is ambiguous in terms of differentiating between an authority component and a path component that begins with two slash characters. The greedy algorithm is used for disambiguation: the left-most matching rule soaks up as much of the URI reference string as it is capable of matching. In other words, the authority component wins.

Readers familiar with regular expressions should see Appendix B for a concrete parsing example and test oracle.

5. Relative URI References

It is often the case that a group or "tree" of documents has been constructed to serve a common purpose; the vast majority of URI in these documents point to resources within the tree rather than

outside of it. Similarly, documents located at a particular site are much more likely to refer to other resources at that site than to resources at remote sites.

Relative addressing of URI allows document trees to be partially independent of their location and access scheme. For instance, it is possible for a single set of hypertext documents to be simultaneously accessible and traversable via each of the "file", "http", and "ftp" schemes if the documents refer to each other using relative URI. Furthermore, such document trees can be moved, as a whole, without changing any of the relative references. Experience within the WWW has demonstrated that the ability to perform relative referencing is necessary for the long-term usability of embedded URI.

The syntax for relative URI takes advantage of the <hier_part> syntax of <absoluteURI> (Section 3) in order to express a reference that is relative to the namespace of another hierarchical URI.

relativeURI = (net_path | abs_path | rel_path) ["?" query]

A relative reference beginning with two slash characters is termed a network-path reference, as defined by <net_path> in Section 3. Such references are rarely used.

A relative reference beginning with a single slash character is termed an absolute-path reference, as defined by <abs_path> in Section 3.

A relative reference that does not begin with a scheme name or a slash character is termed a relative-path reference.

rel_path = rel_segment [abs_path]

rel_segment = 1*(unreserved | escaped |
";" | "@" | "&" | "=" | "+" | "$" | ",")

Within a relative-path reference, the complete path segments "." and ".." have special meanings: "the current hierarchy level" and "the level above this hierarchy level", respectively. Although this is very similar to their use within Unix-based filesystems to indicate directory levels, these path components are only considered special when resolving a relative-path reference to its absolute form (Section 5.2).

Authors should be aware that a path segment which contains a colon character cannot be used as the first segment of a relative URI path (e.g., "this:that"), because it would be mistaken for a scheme name.

It is therefore necessary to precede such segments with other segments (e.g., "./this:that") in order for them to be referenced as a relative path.

It is not necessary for all URI within a given scheme to be restricted to the <hier_part> syntax, since the hierarchical properties of that syntax are only necessary when relative URI are used within a particular document. Documents can only make use of

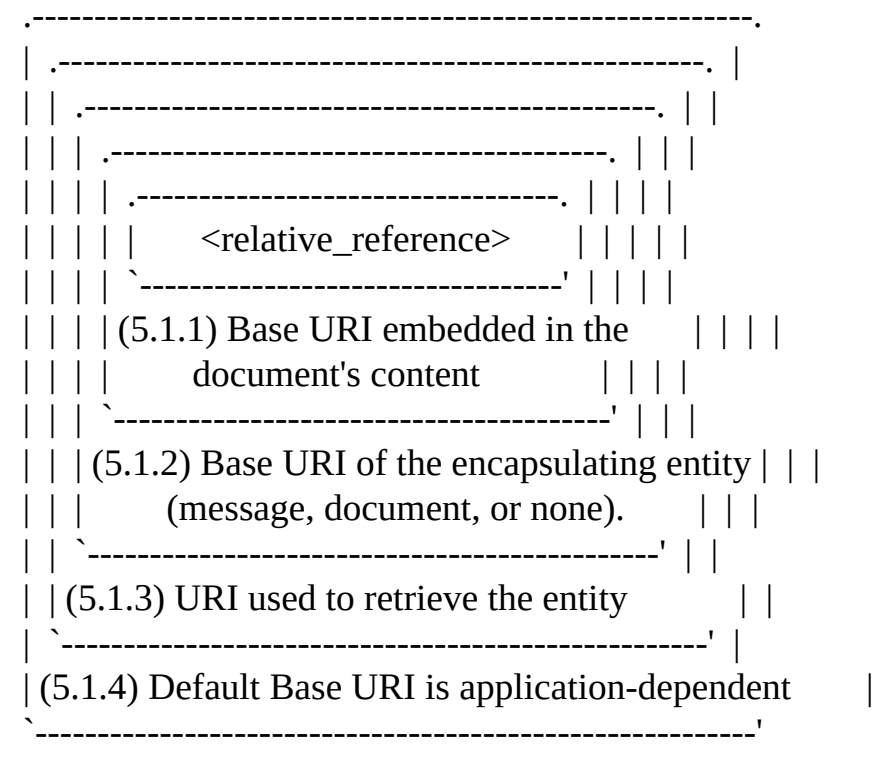
relative URI when their base URI fits within the <hier_part> syntax. It is assumed that any document which contains a relative reference will also have a base URI that obeys the syntax. In other words, relative URI cannot be used within a document that has an unsuitable base URI.

Some URI schemes do not allow a hierarchical syntax matching the <hier_part> syntax, and thus cannot use relative references.

5.1. Establishing a Base URI

The term "relative URI" implies that there exists some absolute "base URI" against which the relative reference is applied. Indeed, the base URI is necessary to define the semantics of any relative URI reference; without it, a relative reference is meaningless. In order for relative URI to be usable within a document, the base URI of that document must be known to the parser.

The base URI of a document can be established in one of four ways, listed below in order of precedence. The order of precedence can be thought of in terms of layers, where the innermost defined base URI has the highest precedence. This can be visualized graphically as:



5.1.1. Base URI within Document Content

Within certain document media types, the base URI of the document can be embedded within the content itself such that it can be readily obtained by a parser. This can be useful for descriptive documents, such as tables of content, which may be transmitted to others through protocols other than their usual retrieval context (e.g., E-Mail or USENET news).

It is beyond the scope of this document to specify how, for each media type, the base URI can be embedded. It is assumed that user agents manipulating such media types will be able to obtain the appropriate syntax from that media type's specification. An example of how the base URI can be embedded in the Hypertext Markup Language (HTML) [RFC1866] is provided in Appendix D.

A mechanism for embedding the base URI within MIME container types (e.g., the message and multipart types) is defined by MHTML [RFC2110]. Protocols that do not use the MIME message header syntax, but which do allow some form of tagged metainformation to be included within messages, may define their own syntax for defining the base URI as part of a message.

5.1.2. Base URI from the Encapsulating Entity

If no base URI is embedded, the base URI of a document is defined by the document's retrieval context. For a document that is enclosed within another entity (such as a message or another document), the retrieval context is that entity; thus, the default base URI of the document is the base URI of the entity in which the document is encapsulated.

5.1.3. Base URI from the Retrieval URI

If no base URI is embedded and the document is not encapsulated within some other entity (e.g., the top level of a composite entity), then, if a URI was used to retrieve the base document, that URI shall be considered the base URI. Note that if the retrieval was the result of a redirected request, the last URI used (i.e., that which resulted in the actual retrieval of the document) is the base URI.

5.1.4. Default Base URI

If none of the conditions described in Sections 5.1.1--5.1.3 apply,

then the base URI is defined by the context of the application. Since this definition is necessarily application-dependent, failing

to define the base URI using one of the other methods may result in the same content being interpreted differently by different types of application.

It is the responsibility of the distributor(s) of a document containing relative URI to ensure that the base URI for that document can be established. It must be emphasized that relative URI cannot be used reliably in situations where the document's base URI is not well-defined.

5.2. Resolving Relative References to Absolute Form

This section describes an example algorithm for resolving URI references that might be relative to a given base URI.

The base URI is established according to the rules of Section 5.1 and parsed into the four main components as described in Section 3. Note that only the scheme component is required to be present in the base URI; the other components may be empty or undefined. A component is undefined if its preceding separator does not appear in the URI reference; the path component is never undefined, though it may be empty. The base URI's query component is not used by the resolution algorithm and may be discarded.

For each URI reference, the following steps are performed in order:

1) The URI reference is parsed into the potential four components and fragment identifier, as described in Section 4.3.

2) If the path component is empty and the scheme, authority, and query components are undefined, then it is a reference to the current document and we are done. Otherwise, the reference URI's query and fragment components are defined as found (or not found) within the URI reference and not inherited from the base URI.

3) If the scheme component is defined, indicating that the reference starts with a scheme name, then the reference is interpreted as an absolute URI and we are done. Otherwise, the reference URI's scheme is inherited from the base URI's scheme component.

Due to a loophole in prior specifications [RFC1630], some parsers allow the scheme name to be present in a relative URI if it is the same as the base URI scheme. Unfortunately, this can conflict with the correct parsing of non-hierarchical URI. For backwards compatibility, an implementation may work around such references by removing the scheme if it matches that of the base URI and the scheme is known to always use the <hier_part> syntax. The parser

can then continue with the steps below for the remainder of the reference components. Validating parsers should mark such a misformed relative reference as an error.

4) If the authority component is defined, then the reference is a network-path and we skip to step 7. Otherwise, the reference URI's authority is inherited from the base URI's authority component, which will also be undefined if the URI scheme does not use an authority component.

5) If the path component begins with a slash character ("/"), then the reference is an absolute-path and we skip to step 7.

6) If this step is reached, then we are resolving a relative-path reference. The relative path needs to be merged with the base URI's path. Although there are many ways to do this, we will describe a simple method using a separate string buffer.

a) All but the last segment of the base URI's path component is copied to the buffer. In other words, any characters after the last (right-most) slash character, if any, are excluded.

b) The reference's path component is appended to the buffer string.

c) All occurrences of "./", where "." is a complete path segment, are removed from the buffer string.

d) If the buffer string ends with "." as a complete path segment, that "." is removed.

e) All occurrences of "/../", where is a complete path segment not equal to "..", are removed from the buffer string. Removal of these path segments is performed iteratively, removing the leftmost matching pattern on each iteration, until no matching pattern remains.

f) If the buffer string ends with "/..", where is a complete path segment not equal to "..", that "/.." is removed.

g) If the resulting buffer string still begins with one or more complete path segments of "..", then the reference is considered to be in error. Implementations may handle this error by retaining these components in the resolved path (i.e., treating them as part of the final URI), by removing them from the resolved path (i.e., discarding relative levels above the root), or by avoiding traversal of the reference.

h) The remaining buffer string is the reference URI's new path component.

7) The resulting URI components, including any inherited from the base URI, are recombined to give the absolute form of the URI reference. Using pseudocode, this would be

```
result = ""

if scheme is defined then
   append scheme to result
   append ":" to result

if authority is defined then
```

```
      append "//" to result
      append authority to result

   append path to result

   if query is defined then
      append "?" to result
      append query to result

   if fragment is defined then
      append "#" to result
      append fragment to result

   return result
```

Note that we must be careful to preserve the distinction between a component that is undefined, meaning that its separator was not present in the reference, and a component that is empty, meaning that the separator was present and was immediately followed by the next component separator or the end of the reference.

The above algorithm is intended to provide an example by which the output of implementations can be tested -- implementation of the algorithm itself is not required. For example, some systems may find it more efficient to implement step 6 as a pair of segment stacks being merged, rather than as a series of string pattern replacements.

Note: Some WWW client applications will fail to separate the reference's query component from its path component before merging the base and reference paths in step 6 above. This may result in a loss of information if the query component contains the strings "/../" or "/./".

Resolution examples are provided in Appendix C.

6. URI Normalization and Equivalence

In many cases, different URI strings may actually identify the identical resource. For example, the host names used in URL are

actually case insensitive, and the URL <http://www.XEROX.com> is equivalent to <http://www.xerox.com>. In general, the rules for equivalence and definition of a normal form, if any, are scheme dependent. When a scheme uses elements of the common syntax, it will also use the common syntax equivalence rules, namely that the scheme and hostname are case insensitive and a URL with an explicit ":port", where the port is the default for the scheme, is equivalent to one where the port is elided.

7. Security Considerations

A URI does not in itself pose a security threat. Users should beware that there is no general guarantee that a URL, which at one time located a given resource, will continue to do so. Nor is there any guarantee that a URL will not locate a different resource at some later point in time, due to the lack of any constraint on how a given authority apportions its namespace. Such a guarantee can only be obtained from the person(s) controlling that namespace and the resource in question. A specific URI scheme may include additional semantics, such as name persistence, if those semantics are required of all naming authorities for that scheme.

It is sometimes possible to construct a URL such that an attempt to perform a seemingly harmless, idempotent operation, such as the retrieval of an entity associated with the resource, will in fact cause a possibly damaging remote operation to occur. The unsafe URL is typically constructed by specifying a port number other than that reserved for the network protocol in question. The client unwittingly contacts a site that is in fact running a different protocol. The content of the URL contains instructions that, when interpreted according to this other protocol, cause an unexpected operation. An example has been the use of a gopher URL to cause an unintended or impersonating message to be sent via a SMTP server.

Caution should be used when using any URL that specifies a port number other than the default for the protocol, especially when it is a number within the reserved space.

Care should be taken when a URL contains escaped delimiters for a given protocol (for example, CR and LF characters for telnet protocols) that these are not unescaped before transmission. This might violate the protocol, but avoids the potential for such

characters to be used to simulate an extra operation or parameter in that protocol, which might lead to an unexpected and possibly harmful remote operation to be performed.

It is clearly unwise to use a URL that contains a password which is intended to be secret. In particular, the use of a password within the 'userinfo' component of a URL is strongly disrecommended except in those rare cases where the 'password' parameter is intended to be public.

8. Acknowledgements

This document was derived from RFC 1738 [RFC1738] and RFC 1808 [RFC1808]; the acknowledgements in those specifications still apply. In addition, contributions by Gisle Aas, Martin Beet, Martin Duerst, Jim Gettys, Martijn Koster, Dave Kristol, Daniel LaLiberte, Foteos Macrides, James Marshall, Ryan Moats, Keith Moore, and Lauren Wood are gratefully acknowledged.

9. References

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[RFC1630] Berners-Lee, T., "Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", RFC 1630, June 1994.

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, Editors, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[RFC1866] Berners-Lee T., and D. Connolly, "HyperText Markup Language Specification -- 2.0", RFC 1866, November 1995.

[RFC1123] Braden, R., Editor, "Requirements for Internet Hosts -- Application and Support", STD 3, RFC 1123, October 1989.

[RFC822] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, August 1982.

[RFC1808] Fielding, R., "Relative Uniform Resource Locators", RFC 1808, June 1995.

[RFC2046] Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC1736] Kunze, J., "Functional Recommendations for Internet Resource Locators", RFC 1736, February 1995.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.

[RFC1034] Mockapetris, P., "Domain Names - Concepts and Facilities", STD 13, RFC 1034, November 1987.

[RFC2110] Palme, J., and A. Hopmann, "MIME E-mail Encapsulation of Aggregate Documents, such as HTML (MHTML)", RFC 2110, March 1997.

[RFC1737] Sollins, K., and L. Masinter, "Functional Requirements for Uniform Resource Names", RFC 1737, December 1994.

[ASCII] US-ASCII. "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4-1986.

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

10. Authors' Addresses

Tim Berners-Lee
World Wide Web Consortium
MIT Laboratory for Computer Science, NE43-356
545 Technology Square
Cambridge, MA 02139

Fax: +1(617)258-8682
EMail: timbl@w3.org

Roy T. Fielding
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425

Fax: +1(949)824-1715
EMail: fielding@ics.uci.edu

Larry Masinter
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94034

Fax: +1(415)812-4333
EMail: masinter@parc.xerox.com

A. Collected BNF for URI

```
URI-reference = [ absoluteURI | relativeURI ] [ "#" fragment ]
absoluteURI   = scheme ":" ( hier_part | opaque_part )
relativeURI   = ( net_path | abs_path | rel_path ) [ "?" query ]

hier_part     = ( net_path | abs_path ) [ "?" query ]
opaque_part   = uric_no_slash *uric

uric_no_slash = unreserved | escaped | ";" | "?" | ":" | "@" |
```

```
                "&" | "=" | "+" | "$" | ","

    net_path      = "//" authority [ abs_path ]
    abs_path      = "/"  path_segments
    rel_path      = rel_segment [ abs_path ]

    rel_segment   = 1*( unreserved | escaped |
                    ";" | "@" | "&" | "=" | "+" | "$" | "," )

    scheme        = alpha *( alpha | digit | "+" | "-" | "." )

    authority     = server | reg_name

    reg_name      = 1*( unreserved | escaped | "$" | "," |
                    ";" | ":" | "@" | "&" | "=" | "+" )

    server        = [ [ userinfo "@" ] hostport ]
    userinfo      = *( unreserved | escaped |
                    ";" | ":" | "&" | "=" | "+" | "$" | "," )

    hostport      = host [ ":" port ]
    host          = hostname | IPv4address
    hostname      = *( domainlabel "." ) toplabel [ "." ]
    domainlabel   = alphanum | alphanum *( alphanum | "-" ) alphanum
    toplabel      = alpha | alpha *( alphanum | "-" ) alphanum
    IPv4address   = 1*digit "." 1*digit "." 1*digit "." 1*digit
    port          = *digit

    path          = [ abs_path | opaque_part ]
    path_segments = segment *( "/" segment )
    segment       = *pchar *( ";" param )
    param         = *pchar
    pchar         = unreserved | escaped |
                    ":" | "@" | "&" | "=" | "+" | "$" | ","

    query         = *uric

    fragment      = *uric
```

```
uric          = reserved | unreserved | escaped
reserved      = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
                "$" | ","
unreserved    = alphanum | mark
mark          = "-" | "_" | "." | "!" | "~" | "*" | "'" |
                "(" | ")"

escaped       = "%" hex hex
hex           = digit | "A" | "B" | "C" | "D" | "E" | "F" |
                        "a" | "b" | "c" | "d" | "e" | "f"

alphanum      = alpha | digit
alpha         = lowalpha | upalpha

lowalpha = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
           "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
           "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
upalpha  = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
           "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
           "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
digit    = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
           "8" | "9"
```

B. Parsing a URI Reference with a Regular Expression

As described in Section 4.3, the generic URI syntax is not sufficient to disambiguate the components of some forms of URI. Since the "greedy algorithm" described in that section is identical to the disambiguation method used by POSIX regular expressions, it is natural and commonplace to use a regular expression for parsing the potential four components and fragment identifier of a URI reference.

The following line is the regular expression for breaking-down a URI reference into its components.

```
^(([^:/?#]+):)?(//([^/?#]*))?([^?#]*)(\?([^#]*))?(#(.*))?
 12            3  4          5       6  7        8 9
```

The numbers in the second line above are only to assist readability; they indicate the reference points for each subexpression (i.e., each paired parenthesis). We refer to the value matched for subexpression <n> as $<n>. For example, matching the above expression to

```
http://www.ics.uci.edu/pub/ietf/uri/#Related
```

results in the following subexpression matches:

```
$1 = http:
$2 = http
$3 = //www.ics.uci.edu
$4 = www.ics.uci.edu
$5 = /pub/ietf/uri/
$6 = <undefined>
$7 = <undefined>
$8 = #Related
$9 = Related
```

where <undefined> indicates that the component is not present, as is the case for the query component in the above example. Therefore, we

can determine the value of the four components and fragment as

```
scheme    = $2
authority = $4
path      = $5
query     = $7
fragment  = $9
```

and, going in the opposite direction, we can recreate a URI reference from its components using the algorithm in step 7 of Section 5.2.

C. Examples of Resolving Relative URI References

Within an object with a well-defined base URI of

```
http://a/b/c/d;p?q
```

the relative URI would be resolved as follows:

C.1. Normal Examples

```
g:h           =  g:h
g             =  http://a/b/c/g
./g           =  http://a/b/c/g
g/            =  http://a/b/c/g/
/g            =  http://a/g
//g           =  http://g
?y            =  http://a/b/c/?y
g?y           =  http://a/b/c/g?y
#s            =  (current document)#s
g#s           =  http://a/b/c/g#s
g?y#s         =  http://a/b/c/g?y#s
;x            =  http://a/b/c/;x
g;x           =  http://a/b/c/g;x
g;x?y#s       =  http://a/b/c/g;x?y#s
.             =  http://a/b/c/
./            =  http://a/b/c/
```

```
..          =  http://a/b/
../         =  http://a/b/
../g        =  http://a/b/g
../..       =  http://a/
../../      =  http://a/
../../g     =  http://a/g
```

C.2. Abnormal Examples

Although the following abnormal examples are unlikely to occur in normal practice, all URI parsers should be capable of resolving them consistently. Each example uses the same base as above.

An empty reference refers to the start of the current document.

```
<>          =  (current document)
```

Parsers must be careful in handling the case where there are more relative path ".." segments than there are hierarchical levels in the base URI's path. Note that the ".." syntax cannot be used to change the authority component of a URI.


```
../../../g    =  http://a/../g
../../../../g =  http://a/../../g
```

In practice, some implementations strip leading relative symbolic elements (".", "..") after applying a relative URI calculation, based on the theory that compensating for obvious author errors is better than allowing the request to fail. Thus, the above two references will be interpreted as "http://a/g" by some implementations.

Similarly, parsers must avoid treating "." and ".." as special when they are not complete components of a relative path.

```
/./g        =  http://a/./g
/../g       =  http://a/../g
g.          =  http://a/b/c/g.
.g          =  http://a/b/c/.g
```

```
g..         =  http://a/b/c/g..
..g         =  http://a/b/c/..g
```

Less likely are cases where the relative URI uses unnecessary or nonsensical forms of the "." and ".." complete path segments.

```
./../g      =  http://a/b/g
./g/.       =  http://a/b/c/g/
g/./h       =  http://a/b/c/g/h
g/../h      =  http://a/b/c/h
g;x=1/./y   =  http://a/b/c/g;x=1/y
g;x=1/../y  =  http://a/b/c/y
```

All client applications remove the query component from the base URI before resolving relative URI. However, some applications fail to separate the reference's query and/or fragment components from a relative path before merging it with the base path. This error is rarely noticed, since typical usage of a fragment never includes the hierarchy ("/") character, and the query component is not normally used within relative references.

```
g?y/./x     =  http://a/b/c/g?y/./x
g?y/../x    =  http://a/b/c/g?y/../x
g#s/./x     =  http://a/b/c/g#s/./x
g#s/../x    =  http://a/b/c/g#s/../x
```


Some parsers allow the scheme name to be present in a relative URI if it is the same as the base URI scheme. This is considered to be a loophole in prior specifications of partial URI [RFC1630]. Its use should be avoided.

```
http:g      =  http:g        ; for validating parsers
```

```
    |  http://a/b/c/g   ; for backwards compatibility
```

D. Embedding the Base URI in HTML documents

It is useful to consider an example of how the base URI of a document can be embedded within the document's content. In this appendix, we describe how documents written in the Hypertext Markup Language (HTML) [RFC1866] can include an embedded base URI. This appendix does not form a part of the URI specification and should not be considered as anything more than a descriptive example.

HTML defines a special element "BASE" which, when present in the "HEAD" portion of a document, signals that the parser should use the BASE element's "HREF" attribute as the base URI for resolving any relative URI. The "HREF" attribute must be an absolute URI. Note that, in HTML, element and attribute names are case-insensitive. For example:

```
<!doctype html public "-//IETF//DTD HTML//EN">
<HTML><HEAD>
<TITLE>An example HTML document</TITLE>
<BASE href="http://www.ics.uci.edu/Test/a/b/c">
</HEAD><BODY>
... <A href="../x">a hypertext anchor</A> ...
</BODY></HTML>
```

A parser reading the example document should interpret the given relative URI "../x" as representing the absolute URI

```
<http://www.ics.uci.edu/Test/a/x>
```

regardless of the context in which the example document was obtained.

E. Recommendations for Delimiting URI in Context

URI are often transmitted through formats that do not provide a clear context for their interpretation. For example, there are many occasions when URI are included in plain text; examples include text sent in electronic mail, USENET news messages, and, most importantly, printed on paper. In such cases, it is important to be able to delimit the URI from the rest of the text, and in particular from punctuation marks that might be mistaken for part of the URI.

In practice, URI are delimited in a variety of ways, but usually within double-quotes "http://test.com/", angle brackets <http://test.com/>, or just using whitespace

```
http://test.com/
```

These wrappers do not form part of the URI.

In the case where a fragment identifier is associated with a URI reference, the fragment would be placed within the brackets as well (separated from the URI with a "#" character).

In some cases, extra whitespace (spaces, linebreaks, tabs, etc.) may need to be added to break long URI across lines. The whitespace should be ignored when extracting the URI.

No whitespace should be introduced after a hyphen ("-") character. Because some typesetters and printers may (erroneously) introduce a hyphen at the end of line when breaking a line, the interpreter of a URI containing a line break immediately after a hyphen should ignore all unescaped whitespace around the line break, and should be aware that the hyphen may or may not actually be part of the URI.

Using <> angle brackets around each URI is especially recommended as a delimiting style for URI that contain whitespace.

The prefix "URL:" (with or without a trailing space) was recommended as a way to used to help distinguish a URL from other bracketed designators, although this is not common in practice.

For robustness, software that accepts user-typed URI should attempt to recognize and strip both delimiters and embedded whitespace.

For example, the text:

> Yes, Jim, I found it under "http://www.w3.org/Addressing/", but you can probably pick it up from <ftp://ds.internic.net/rfc/>. Note the warning in <http://www.ics.uci.edu/pub/ietf/uri/historical.html#WARNING>.

contains the URI references

> http://www.w3.org/Addressing/
> ftp://ds.internic.net/rfc/
> http://www.ics.uci.edu/pub/ietf/uri/historical.html#WARNING

F. Abbreviated URLs

The URL syntax was designed for unambiguous reference to network resources and extensibility via the URL scheme. However, as URL identification and usage have become commonplace, traditional media (television, radio, newspapers, billboards, etc.) have increasingly used abbreviated URL references. That is, a reference consisting of only the authority and path portions of the identified resource, such as

www.w3.org/Addressing/

or simply the DNS hostname on its own. Such references are primarily intended for human interpretation rather than machine, with the assumption that context-based heuristics are sufficient to complete the URL (e.g., most hostnames beginning with "www" are likely to have a URL prefix of "http://"). Although there is no standard set of heuristics for disambiguating abbreviated URL references, many client implementations allow them to be entered by the user and heuristically resolved. It should be noted that such heuristics may change over time, particularly when new URL schemes are introduced.

Since an abbreviated URL has the same syntax as a relative URL path, abbreviated URL references cannot be used in contexts where relative URLs are expected. This limits the use of abbreviated URLs to places where there is no defined base URL, such as dialog boxes and off-line advertisements.

G. Summary of Non-editorial Changes

G.1. Additions

Section 4 (URI References) was added to stem the confusion regarding "what is a URI" and how to describe fragment identifiers given that they are not part of the URI, but are part of the URI syntax and parsing concerns. In addition, it provides a reference definition for use by other IETF specifications (HTML, HTTP, etc.) that have previously attempted to redefine the URI syntax in order to account for the presence of fragment identifiers in URI references.

Section 2.4 was rewritten to clarify a number of misinterpretations and to leave room for fully internationalized URI.

Appendix F on abbreviated URLs was added to describe the shortened references often seen on television and magazine advertisements and explain why they are not used in other contexts.

G.2. Modifications from both RFC 1738 and RFC 1808

Changed to URI syntax instead of just URL.

Confusion regarding the terms "character encoding", the URI "character set", and the escaping of characters with %<hex><hex> equivalents has (hopefully) been reduced. Many of the BNF rule names regarding the character sets have been changed to more accurately describe their purpose and to encompass all "characters" rather than just US-ASCII octets. Unless otherwise noted here, these modifications do not affect the URI syntax.

Both RFC 1738 and RFC 1808 refer to the "reserved" set of characters as if URI-interpreting software were limited to a single set of characters with a reserved purpose (i.e., as meaning something other than the data to which the characters correspond), and that this set was fixed by the URI scheme. However, this has not been true in practice; any character that is interpreted differently when it is escaped is, in effect, reserved. Furthermore, the interpreting engine on a HTTP server is often dependent on the resource, not just the URI scheme. The description of reserved characters has been changed accordingly.

The plus "+", dollar "$", and comma "," characters have been added to those in the "reserved" set, since they are treated as reserved within the query component.

The tilde "~" character was added to those in the "unreserved" set, since it is extensively used on the Internet in spite of the difficulty to transcribe it with some keyboards.

The syntax for URI scheme has been changed to require that all schemes begin with an alpha character.

The "user:password" form in the previous BNF was changed to a "userinfo" token, and the possibility that it might be "user:password" made scheme specific. In particular, the use of passwords in the clear is not even suggested by the syntax.

The question-mark "?" character was removed from the set of allowed characters for the userinfo in the authority component, since testing showed that many applications treat it as reserved for separating the query component from the rest of the URI.

The semicolon ";" character was added to those stated as being reserved within the authority component, since several new schemes are using it as a separator within userinfo to indicate the type of user authentication.

RFC 1738 specified that the path was separated from the authority portion of a URI by a slash. RFC 1808 followed suit, but with a fudge of carrying around the separator as a "prefix" in order to describe the parsing algorithm. RFC 1630 never had this problem, since it considered the slash to be part of the path. In writing this specification, it was found to be impossible to accurately describe and retain the difference between the two URI

<foo:/bar> and <foo:bar>

without either considering the slash to be part of the path (as corresponds to actual practice) or creating a separate component just to hold that slash. We chose the former.

G.3. Modifications from RFC 1738

The definition of specific URL schemes and their scheme-specific syntax and semantics has been moved to separate documents.

The URL host was defined as a fully-qualified domain name. However, many URLs are used without fully-qualified domain names (in contexts for which the full qualification is not necessary), without any host (as in some file URLs), or with a host of "localhost".

The URL port is now *digit instead of 1*digit, since systems are expected to handle the case where the ":" separator between host and port is supplied without a port.

The recommendations for delimiting URI in context (Appendix E) have been adjusted to reflect current practice.

G.4. Modifications from RFC 1808

RFC 1808 (Section 4) defined an empty URL reference (a reference containing nothing aside from the fragment identifier) as being a reference to the base URL. Unfortunately, that definition could be interpreted, upon selection of such a reference, as a new retrieval action on that resource. Since the normal intent of such references is for the user agent to change its view of the current document to the beginning of the specified fragment within that document, not to make an additional request of the resource, a description of how to correctly interpret an empty reference has been added in Section 4.

The description of the mythical Base header field has been replaced with a reference to the Content-Location header field defined by MHTML [RFC2110].

RFC 1808 described various schemes as either having or not having the properties of the generic URI syntax. However, the only requirement is that the particular document containing the relative references have a base URI that abides by the generic URI syntax, regardless of the URI scheme, so the associated description has been updated to reflect that.

The BNF term <net_loc> has been replaced with <authority>, since the latter more accurately describes its use and purpose. Likewise, the

authority is no longer restricted to the IP server syntax.

Extensive testing of current client applications demonstrated that the majority of deployed systems do not use the ";" character to indicate trailing parameter information, and that the presence of a semicolon in a path segment does not affect the relative parsing of that segment. Therefore, parameters have been removed as a separate component and may now appear in any path segment. Their influence has been removed from the algorithm for resolving a relative URI reference. The resolution examples in Appendix C have been modified to reflect this change.

Implementations are now allowed to work around misformed relative references that are prefixed by the same scheme as the base URI, but only for schemes known to use the <hier_part> syntax.

H. Full Copyright Statement

==========

CHARACTER SETS

(last updated 2004-02-06)

These are the official names for character sets that may be used in the Internet and may be referred to in Internet documentation. These names are expressed in ANSI_X3.4-1968 which is commonly called US-ASCII or simply ASCII. The character set most commonly use in the Internet and used especially in protocol standards is US-ASCII, this is strongly encouraged. The use of the name US-ASCII is also encouraged.

The character set names may be up to 40 characters taken from the printable characters of US-ASCII. However, no distinction is made between use of upper and lower case letters.

The MIBenum value is a unique value for use in MIBs to identify coded character sets.

The value space for MIBenum values has been divided into three regions. The first region (3-999) consists of coded character sets that have been standardized by some standard setting organization. This region is intended for standards that do not have subset implementations. The second region (1000-1999) is for the Unicode and ISO/IEC 10646 coded character sets together with a specification of a (set of) sub-repertoires that may occur. The third region (>1999) is intended for vendor specific coded character sets.

Assigned MIB enum Numbers	
0-2	Reserved
3-999	Set By Standards Organizations
1000-1999	Unicode / 10646
2000-2999	Vendor

The aliases that start with "cs" have been added for use with the Printer MIB (see RFC 1759) and contain the standard numbers along with suggestive names in order to facilitate applications that want to display the names in user interfaces. The "cs" stands for character set and is provided for applications that need a lower case first letter but want to use mixed case thereafter that cannot contain any special characters, such as underbar ("_") and dash ("-"). If the character set is from an ISO standard, its cs alias is the ISO standard number or name. If the character set is not from an ISO

standard, but is registered with ISO (IPSJ/ITSCJ is the current ISO Registration Authority), the ISO Registry number is specified as ISOnnn followed by letters suggestive of the name or standards number of the code set. When a national or international standard is revised, the year of revision is added to the cs alias of the new character set entry in the IANA Registry in order to distinguish the revised character set from the original character set.

Character Set	Reference

Name: ANSI_X3.4-1968 [RFC1345,KXS2]
MIBenum: 3
Source: ECMA registry
Alias: iso-ir-6
Alias: ANSI_X3.4-1986
Alias: ISO_646.irv:1991
Alias: ASCII
Alias: ISO646-US
Alias: US-ASCII (preferred MIME name)
Alias: us
Alias: IBM367
Alias: cp367
Alias: csASCII

Name: ISO-10646-UTF-1
MIBenum: 27
Source: Universal Transfer Format (1), this is the multibyte encoding, that subsets ASCII-7. It does not have byte ordering issues.
Alias: csISO10646UTF1

Name: ISO_646.basic:1983 [RFC1345,KXS2]
MIBenum: 28
Source: ECMA registry
Alias: ref
Alias: csISO646basic1983

Name: INVARIANT [RFC1345,KXS2]
MIBenum: 29
Alias: csINVARIANT

Name: ISO_646.irv:1983 [RFC1345,KXS2]
MIBenum: 30
Source: ECMA registry

Alias: iso-ir-2
Alias: irv
Alias: csISO2IntlRefVersion

Name: BS_4730 [RFC1345,KXS2]
MIBenum: 20
Source: ECMA registry
Alias: iso-ir-4
Alias: ISO646-GB
Alias: gb
Alias: uk
Alias: csISO4UnitedKingdom

Name: NATS-SEFI [RFC1345,KXS2]
MIBenum: 31
Source: ECMA registry
Alias: iso-ir-8-1
Alias: csNATSSEFI

Name: NATS-SEFI-ADD [RFC1345,KXS2]
MIBenum: 32
Source: ECMA registry
Alias: iso-ir-8-2
Alias: csNATSSEFIADD

Name: NATS-DANO [RFC1345,KXS2]
MIBenum: 33
Source: ECMA registry
Alias: iso-ir-9-1
Alias: csNATSDANO

Name: NATS-DANO-ADD [RFC1345,KXS2]
MIBenum: 34
Source: ECMA registry
Alias: iso-ir-9-2
Alias: csNATSDANOADD

Name: SEN_850200_B [RFC1345,KXS2]
MIBenum: 35
Source: ECMA registry
Alias: iso-ir-10
Alias: FI
Alias: ISO646-FI
Alias: ISO646-SE
Alias: se
Alias: csISO10Swedish

Name: SEN_850200_C [RFC1345,KXS2]
MIBenum: 21
Source: ECMA registry
Alias: iso-ir-11
Alias: ISO646-SE2
Alias: se2
Alias: csISO11SwedishForNames

Name: KS_C_5601-1987 [RFC1345,KXS2]
MIBenum: 36
Source: ECMA registry
Alias: iso-ir-149
Alias: KS_C_5601-1989
Alias: KSC_5601
Alias: korean
Alias: csKSC56011987

Name: ISO-2022-KR (preferred MIME name) [RFC1557,Choi]
MIBenum: 37
Source: RFC-1557 (see also KS_C_5601-1987)
Alias: csISO2022KR

Name: EUC-KR (preferred MIME name) [RFC1557,Choi]
MIBenum: 38
Source: RFC-1557 (see also KS_C_5861-1992)
Alias: csEUCKR

Name: ISO-2022-JP (preferred MIME name) [RFC1468,Murai]
MIBenum: 39
Source: RFC-1468 (see also RFC-2237)
Alias: csISO2022JP

Name: ISO-2022-JP-2 (preferred MIME name) [RFC1554,Ohta]
MIBenum: 40
Source: RFC-1554
Alias: csISO2022JP2

Name: ISO-2022-CN [RFC1922]
MIBenum: 104
Source: RFC-1922

Name: ISO-2022-CN-EXT [RFC1922]
MIBenum: 105
Source: RFC-1922

Name: JIS_C6220-1969-jp [RFC1345,KXS2]
MIBenum: 41
Source: ECMA registry
Alias: JIS_C6220-1969
Alias: iso-ir-13
Alias: katakana
Alias: x0201-7
Alias: csISO13JISC6220jp

Name: JIS_C6220-1969-ro [RFC1345,KXS2]
MIBenum: 42
Source: ECMA registry
Alias: iso-ir-14
Alias: jp
Alias: ISO646-JP
Alias: csISO14JISC6220ro

Name: IT [RFC1345,KXS2]
MIBenum: 22
Source: ECMA registry
Alias: iso-ir-15
Alias: ISO646-IT
Alias: csISO15Italian

Name: PT [RFC1345,KXS2]
MIBenum: 43
Source: ECMA registry
Alias: iso-ir-16
Alias: ISO646-PT
Alias: csISO16Portuguese

Name: ES [RFC1345,KXS2]
MIBenum: 23
Source: ECMA registry
Alias: iso-ir-17
Alias: ISO646-ES
Alias: csISO17Spanish

Name: greek7-old [RFC1345,KXS2]
MIBenum: 44
Source: ECMA registry
Alias: iso-ir-18
Alias: csISO18Greek7Old

Name: latin-greek [RFC1345,KXS2]
MIBenum: 45

Source: ECMA registry
Alias: iso-ir-19
Alias: csISO19LatinGreek

Name: DIN_66003 [RFC1345,KXS2]
MIBenum: 24
Source: ECMA registry
Alias: iso-ir-21
Alias: de
Alias: ISO646-DE
Alias: csISO21German

Name: NF_Z_62-010_(1973) [RFC1345,KXS2]
MIBenum: 46
Source: ECMA registry
Alias: iso-ir-25
Alias: ISO646-FR1
Alias: csISO25French

Name: Latin-greek-1 [RFC1345,KXS2]
MIBenum: 47
Source: ECMA registry
Alias: iso-ir-27
Alias: csISO27LatinGreek1

Name: ISO_5427 [RFC1345,KXS2]
MIBenum: 48
Source: ECMA registry
Alias: iso-ir-37
Alias: csISO5427Cyrillic

Name: JIS_C6226-1978 [RFC1345,KXS2]
MIBenum: 49
Source: ECMA registry
Alias: iso-ir-42
Alias: csISO42JISC62261978

Name: BS_viewdata [RFC1345,KXS2]
MIBenum: 50
Source: ECMA registry
Alias: iso-ir-47
Alias: csISO47BSViewdata

Name: INIS [RFC1345,KXS2]
MIBenum: 51
Source: ECMA registry

Alias: iso-ir-49
Alias: csISO49INIS

Name: INIS-8 [RFC1345,KXS2]
MIBenum: 52
Source: ECMA registry
Alias: iso-ir-50
Alias: csISO50INIS8

Name: INIS-cyrillic [RFC1345,KXS2]
MIBenum: 53
Source: ECMA registry
Alias: iso-ir-51
Alias: csISO51INISCyrillic

Name: ISO_5427:1981 [RFC1345,KXS2]
MIBenum: 54
Source: ECMA registry
Alias: iso-ir-54
Alias: ISO5427Cyrillic1981

Name: ISO_5428:1980 [RFC1345,KXS2]
MIBenum: 55
Source: ECMA registry
Alias: iso-ir-55
Alias: csISO5428Greek

Name: GB_1988-80 [RFC1345,KXS2]
MIBenum: 56
Source: ECMA registry
Alias: iso-ir-57
Alias: cn
Alias: ISO646-CN
Alias: csISO57GB1988

Name: GB_2312-80 [RFC1345,KXS2]
MIBenum: 57
Source: ECMA registry
Alias: iso-ir-58
Alias: chinese
Alias: csISO58GB231280

Name: NS_4551-1 [RFC1345,KXS2]
MIBenum: 25
Source: ECMA registry
Alias: iso-ir-60

Alias: ISO646-NO
Alias: no
Alias: csISO60DanishNorwegian
Alias: csISO60Norwegian1

Name: NS_4551-2 [RFC1345,KXS2]
MIBenum: 58
Source: ECMA registry
Alias: ISO646-NO2
Alias: iso-ir-61
Alias: no2
Alias: csISO61Norwegian2

Name: NF_Z_62-010 [RFC1345,KXS2]
MIBenum: 26
Source: ECMA registry
Alias: iso-ir-69
Alias: ISO646-FR
Alias: fr
Alias: csISO69French

Name: videotex-suppl [RFC1345,KXS2]
MIBenum: 59
Source: ECMA registry
Alias: iso-ir-70
Alias: csISO70VideotexSupp1

Name: PT2 [RFC1345,KXS2]
MIBenum: 60
Source: ECMA registry
Alias: iso-ir-84
Alias: ISO646-PT2
Alias: csISO84Portuguese2

Name: ES2 [RFC1345,KXS2]
MIBenum: 61
Source: ECMA registry
Alias: iso-ir-85
Alias: ISO646-ES2
Alias: csISO85Spanish2

Name: MSZ_7795.3 [RFC1345,KXS2]
MIBenum: 62
Source: ECMA registry
Alias: iso-ir-86
Alias: ISO646-HU

Alias: hu
Alias: csISO86Hungarian

Name: JIS_C6226-1983 [RFC1345,KXS2]
MIBenum: 63
Source: ECMA registry
Alias: iso-ir-87
Alias: x0208
Alias: JIS_X0208-1983
Alias: csISO87JISX0208

Name: greek7 [RFC1345,KXS2]
MIBenum: 64
Source: ECMA registry
Alias: iso-ir-88
Alias: csISO88Greek7

Name: ASMO_449 [RFC1345,KXS2]
MIBenum: 65
Source: ECMA registry
Alias: ISO_9036
Alias: arabic7
Alias: iso-ir-89
Alias: csISO89ASMO449

Name: iso-ir-90 [RFC1345,KXS2]
MIBenum: 66
Source: ECMA registry
Alias: csISO90

Name: JIS_C6229-1984-a [RFC1345,KXS2]
MIBenum: 67
Source: ECMA registry
Alias: iso-ir-91
Alias: jp-ocr-a
Alias: csISO91JISC62291984a

Name: JIS_C6229-1984-b [RFC1345,KXS2]
MIBenum: 68
Source: ECMA registry
Alias: iso-ir-92
Alias: ISO646-JP-OCR-B
Alias: jp-ocr-b
Alias: csISO92JISC62991984b

Name: JIS_C6229-1984-b-add [RFC1345,KXS2]

MIBenum: 69
Source: ECMA registry
Alias: iso-ir-93
Alias: jp-ocr-b-add
Alias: csISO93JIS62291984badd

Name: JIS_C6229-1984-hand [RFC1345,KXS2]
MIBenum: 70
Source: ECMA registry
Alias: iso-ir-94
Alias: jp-ocr-hand
Alias: csISO94JIS62291984hand

Name: JIS_C6229-1984-hand-add [RFC1345,KXS2]
MIBenum: 71
Source: ECMA registry
Alias: iso-ir-95
Alias: jp-ocr-hand-add
Alias: csISO95JIS62291984handadd

Name: JIS_C6229-1984-kana [RFC1345,KXS2]
MIBenum: 72
Source: ECMA registry
Alias: iso-ir-96
Alias: csISO96JISC62291984kana

Name: ISO_2033-1983 [RFC1345,KXS2]
MIBenum: 73
Source: ECMA registry
Alias: iso-ir-98
Alias: e13b
Alias: csISO2033

Name: ANSI_X3.110-1983 [RFC1345,KXS2]
MIBenum: 74
Source: ECMA registry
Alias: iso-ir-99
Alias: CSA_T500-1983
Alias: NAPLPS
Alias: csISO99NAPLPS

Name: ISO_8859-1:1987 [RFC1345,KXS2]
MIBenum: 4
Source: ECMA registry
Alias: iso-ir-100
Alias: ISO_8859-1

Alias: ISO-8859-1 (preferred MIME name)
Alias: latin1
Alias: l1
Alias: IBM819
Alias: CP819
Alias: csISOLatin1

Name: ISO_8859-2:1987 [RFC1345,KXS2]
MIBenum: 5
Source: ECMA registry
Alias: iso-ir-101
Alias: ISO_8859-2
Alias: ISO-8859-2 (preferred MIME name)
Alias: latin2
Alias: l2
Alias: csISOLatin2

Name: T.61-7bit [RFC1345,KXS2]
MIBenum: 75
Source: ECMA registry
Alias: iso-ir-102
Alias: csISO102T617bit

Name: T.61-8bit [RFC1345,KXS2]
MIBenum: 76
Alias: T.61
Source: ECMA registry
Alias: iso-ir-103
Alias: csISO103T618bit

Name: ISO_8859-3:1988 [RFC1345,KXS2]
MIBenum: 6
Source: ECMA registry
Alias: iso-ir-109
Alias: ISO_8859-3
Alias: ISO-8859-3 (preferred MIME name)
Alias: latin3
Alias: l3
Alias: csISOLatin3

Name: ISO_8859-4:1988 [RFC1345,KXS2]
MIBenum: 7
Source: ECMA registry
Alias: iso-ir-110
Alias: ISO_8859-4
Alias: ISO-8859-4 (preferred MIME name)

Alias: latin4
Alias: l4
Alias: csISOLatin4

Name: ECMA-cyrillic
MIBenum: 77
Source: ISO registry (formerly ECMA registry)
http://www.itscj.ipsj.jp/ISO-IR/111.pdf
Alias: iso-ir-111
Alias: KOI8-E
Alias: csISO111ECMACyrillic

Name: CSA_Z243.4-1985-1 [RFC1345,KXS2]
MIBenum: 78
Source: ECMA registry
Alias: iso-ir-121
Alias: ISO646-CA
Alias: csa7-1
Alias: ca
Alias: csISO121Canadian1

Name: CSA_Z243.4-1985-2 [RFC1345,KXS2]
MIBenum: 79
Source: ECMA registry
Alias: iso-ir-122
Alias: ISO646-CA2
Alias: csa7-2
Alias: csISO122Canadian2

Name: CSA_Z243.4-1985-gr [RFC1345,KXS2]
MIBenum: 80
Source: ECMA registry
Alias: iso-ir-123
Alias: csISO123CSAZ24341985gr

Name: ISO_8859-6:1987 [RFC1345,KXS2]
MIBenum: 9
Source: ECMA registry
Alias: iso-ir-127
Alias: ISO_8859-6
Alias: ISO-8859-6 (preferred MIME name)
Alias: ECMA-114
Alias: ASMO-708
Alias: arabic
Alias: csISOLatinArabic

Name: ISO_8859-6-E [RFC1556,IANA]
MIBenum: 81
Source: RFC1556
Alias: csISO88596E
Alias: ISO-8859-6-E (preferred MIME name)

Name: ISO_8859-6-I [RFC1556,IANA]
MIBenum: 82
Source: RFC1556
Alias: csISO88596I
Alias: ISO-8859-6-I (preferred MIME name)

Name: ISO_8859-7:1987 [RFC1947,RFC1345,KXS2]
MIBenum: 10
Source: ECMA registry
Alias: iso-ir-126
Alias: ISO_8859-7
Alias: ISO-8859-7 (preferred MIME name)
Alias: ELOT_928
Alias: ECMA-118
Alias: greek
Alias: greek8
Alias: csISOLatinGreek

Name: T.101-G2 [RFC1345,KXS2]
MIBenum: 83
Source: ECMA registry
Alias: iso-ir-128
Alias: csISO128T101G2

Name: ISO_8859-8:1988 [RFC1345,KXS2]
MIBenum: 11
Source: ECMA registry
Alias: iso-ir-138
Alias: ISO_8859-8
Alias: ISO-8859-8 (preferred MIME name)
Alias: hebrew
Alias: csISOLatinHebrew

Name: ISO_8859-8-E [RFC1556,Nussbacher]
MIBenum: 84
Source: RFC1556
Alias: csISO88598E
Alias: ISO-8859-8-E (preferred MIME name)

Name: ISO_8859-8-I [RFC1556,Nussbacher]

MIBenum: 85
Source: RFC1556
Alias: csISO88598I
Alias: ISO-8859-8-I (preferred MIME name)

Name: CSN_369103 [RFC1345,KXS2]
MIBenum: 86
Source: ECMA registry
Alias: iso-ir-139
Alias: csISO139CSN369103

Name: JUS_I.B1.002 [RFC1345,KXS2]
MIBenum: 87
Source: ECMA registry
Alias: iso-ir-141
Alias: ISO646-YU
Alias: js
Alias: yu
Alias: csISO141JUSIB1002

Name: ISO_6937-2-add [RFC1345,KXS2]
MIBenum: 14
Source: ECMA registry and ISO 6937-2:1983
Alias: iso-ir-142
Alias: csISOTextComm

Name: IEC_P27-1 [RFC1345,KXS2]
MIBenum: 88
Source: ECMA registry
Alias: iso-ir-143
Alias: csISO143IECP271

Name: ISO_8859-5:1988 [RFC1345,KXS2]
MIBenum: 8
Source: ECMA registry
Alias: iso-ir-144
Alias: ISO_8859-5
Alias: ISO-8859-5 (preferred MIME name)
Alias: cyrillic
Alias: csISOLatinCyrillic

Name: JUS_I.B1.003-serb [RFC1345,KXS2]
MIBenum: 89
Source: ECMA registry
Alias: iso-ir-146
Alias: serbian

Alias: csISO146Serbian

Name: JUS_I.B1.003-mac [RFC1345,KXS2]
MIBenum: 90
Source: ECMA registry
Alias: macedonian
Alias: iso-ir-147
Alias: csISO147Macedonian

Name: ISO_8859-9:1989 [RFC1345,KXS2]
MIBenum: 12
Source: ECMA registry
Alias: iso-ir-148
Alias: ISO_8859-9
Alias: ISO-8859-9 (preferred MIME name)
Alias: latin5
Alias: l5
Alias: csISOLatin5

Name: greek-ccitt [RFC1345,KXS2]
MIBenum: 91
Source: ECMA registry
Alias: iso-ir-150
Alias: csISO150
Alias: csISO150GreekCCITT

Name: NC_NC00-10:81 [RFC1345,KXS2]
MIBenum: 92
Source: ECMA registry
Alias: cuba
Alias: iso-ir-151
Alias: ISO646-CU
Alias: csISO151Cuba

Name: ISO_6937-2-25 [RFC1345,KXS2]
MIBenum: 93
Source: ECMA registry
Alias: iso-ir-152
Alias: csISO6937Add

Name: GOST_19768-74 [RFC1345,KXS2]
MIBenum: 94
Source: ECMA registry
Alias: ST_SEV_358-88
Alias: iso-ir-153
Alias: csISO153GOST1976874

Name: ISO_8859-supp [RFC1345,KXS2]
MIBenum: 95
Source: ECMA registry
Alias: iso-ir-154
Alias: latin1-2-5
Alias: csISO8859Supp

Name: ISO_10367-box [RFC1345,KXS2]
MIBenum: 96
Source: ECMA registry
Alias: iso-ir-155
Alias: csISO10367Box

Name: ISO-8859-10 (preferred MIME name) [RFC1345,KXS2]
MIBenum: 13
Source: ECMA registry
Alias: iso-ir-157
Alias: l6
Alias: ISO_8859-10:1992
Alias: csISOLatin6
Alias: latin6

Name: latin-lap [RFC1345,KXS2]
MIBenum: 97
Source: ECMA registry
Alias: lap
Alias: iso-ir-158
Alias: csISO158Lap

Name: JIS_X0212-1990 [RFC1345,KXS2]
MIBenum: 98
Source: ECMA registry
Alias: x0212
Alias: iso-ir-159
Alias: csISO159JISX02121990

Name: DS_2089 [RFC1345,KXS2]
MIBenum: 99
Source: Danish Standard, DS 2089, February 1974
Alias: DS2089
Alias: ISO646-DK
Alias: dk
Alias: csISO646Danish

Name: us-dk [RFC1345,KXS2]

MIBenum: 100
Alias: csUSDK

Name: dk-us [RFC1345,KXS2]
MIBenum: 101
Alias: csDKUS

Name: JIS_X0201 [RFC1345,KXS2]
MIBenum: 15
Source: JIS X 0201-1976. One byte only, this is equivalent to
JIS/Roman (similar to ASCII) plus eight-bit half-width
Katakana
Alias: X0201
Alias: csHalfWidthKatakana

Name: KSC5636 [RFC1345,KXS2]
MIBenum: 102
Alias: ISO646-KR
Alias: csKSC5636

Name: ISO-10646-UCS-2
MIBenum: 1000
Source: the 2-octet Basic Multilingual Plane, aka Unicode
this needs to specify network byte order: the standard
does not specify (it is a 16-bit integer space)
Alias: csUnicode

Name: ISO-10646-UCS-4
MIBenum: 1001
Source: the full code space. (same comment about byte order,
these are 31-bit numbers.
Alias: csUCS4

Name: DEC-MCS [RFC1345,KXS2]
MIBenum: 2008
Source: VAX/VMS User's Manual,
Order Number: AI-Y517A-TE, April 1986.
Alias: dec
Alias: csDECMCS

Name: hp-roman8 [HP-PCL5,RFC1345,KXS2]
MIBenum: 2004
Source: LaserJet IIP Printer User's Manual,
HP part no 33471-90901, Hewlet-Packard, June 1989.
Alias: roman8
Alias: r8

Alias: csHPRoman8

Name: macintosh [RFC1345,KXS2]
MIBenum: 2027
Source: The Unicode Standard ver1.0, ISBN 0-201-56788-1, Oct 1991
Alias: mac
Alias: csMacintosh

Name: IBM037 [RFC1345,KXS2]
MIBenum: 2028
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp037
Alias: ebcdic-cp-us
Alias: ebcdic-cp-ca
Alias: ebcdic-cp-wt
Alias: ebcdic-cp-nl
Alias: csIBM037

Name: IBM038 [RFC1345,KXS2]
MIBenum: 2029
Source: IBM 3174 Character Set Ref, GA27-3831-02, March 1990
Alias: EBCDIC-INT
Alias: cp038
Alias: csIBM038

Name: IBM273 [RFC1345,KXS2]
MIBenum: 2030
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP273
Alias: csIBM273

Name: IBM274 [RFC1345,KXS2]
MIBenum: 2031
Source: IBM 3174 Character Set Ref, GA27-3831-02, March 1990
Alias: EBCDIC-BE
Alias: CP274
Alias: csIBM274

Name: IBM275 [RFC1345,KXS2]
MIBenum: 2032
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: EBCDIC-BR
Alias: cp275
Alias: csIBM275

Name: IBM277 [RFC1345,KXS2]

MIBenum: 2033
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: EBCDIC-CP-DK
Alias: EBCDIC-CP-NO
Alias: csIBM277

Name: IBM278 [RFC1345,KXS2]
MIBenum: 2034
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP278
Alias: ebcdic-cp-fi
Alias: ebcdic-cp-se
Alias: csIBM278

Name: IBM280 [RFC1345,KXS2]
MIBenum: 2035
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP280
Alias: ebcdic-cp-it
Alias: csIBM280

Name: IBM281 [RFC1345,KXS2]
MIBenum: 2036
Source: IBM 3174 Character Set Ref, GA27-3831-02, March 1990
Alias: EBCDIC-JP-E
Alias: cp281
Alias: csIBM281

Name: IBM284 [RFC1345,KXS2]
MIBenum: 2037
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP284
Alias: ebcdic-cp-es
Alias: csIBM284

Name: IBM285 [RFC1345,KXS2]
MIBenum: 2038
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP285
Alias: ebcdic-cp-gb
Alias: csIBM285

Name: IBM290 [RFC1345,KXS2]
MIBenum: 2039
Source: IBM 3174 Character Set Ref, GA27-3831-02, March 1990
Alias: cp290

Alias: EBCDIC-JP-kana
Alias: csIBM290

Name: IBM297 [RFC1345,KXS2]
MIBenum: 2040
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp297
Alias: ebcdic-cp-fr
Alias: csIBM297

Name: IBM420 [RFC1345,KXS2]
MIBenum: 2041
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990,
IBM NLS RM p 11-11
Alias: cp420
Alias: ebcdic-cp-ar1
Alias: csIBM420

Name: IBM423 [RFC1345,KXS2]
MIBenum: 2042
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp423
Alias: ebcdic-cp-gr
Alias: csIBM423

Name: IBM424 [RFC1345,KXS2]
MIBenum: 2043
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp424
Alias: ebcdic-cp-he
Alias: csIBM424

Name: IBM437 [RFC1345,KXS2]
MIBenum: 2011
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp437
Alias: 437
Alias: csPC8CodePage437

Name: IBM500 [RFC1345,KXS2]
MIBenum: 2044
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP500
Alias: ebcdic-cp-be
Alias: ebcdic-cp-ch
Alias: csIBM500

Name: IBM775 [HP-PCL5]
MIBenum: 2087
Source: HP PCL 5 Comparison Guide (P/N 5021-0329) pp B-13, 1996
Alias: cp775
Alias: csPC775Baltic

Name: IBM850 [RFC1345,KXS2]
MIBenum: 2009
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp850
Alias: 850
Alias: csPC850Multilingual

Name: IBM851 [RFC1345,KXS2]
MIBenum: 2045
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp851
Alias: 851
Alias: csIBM851

Name: IBM852 [RFC1345,KXS2]
MIBenum: 2010
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp852
Alias: 852
Alias: csPCp852

Name: IBM855 [RFC1345,KXS2]
MIBenum: 2046
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp855
Alias: 855
Alias: csIBM855

Name: IBM857 [RFC1345,KXS2]
MIBenum: 2047
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp857
Alias: 857
Alias: csIBM857

Name: IBM860 [RFC1345,KXS2]
MIBenum: 2048
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp860

Alias: 860
Alias: csIBM860

Name: IBM861 [RFC1345,KXS2]
MIBenum: 2049
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp861
Alias: 861
Alias: cp-is
Alias: csIBM861

Name: IBM862 [RFC1345,KXS2]
MIBenum: 2013
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp862
Alias: 862
Alias: csPC862LatinHebrew

Name: IBM863 [RFC1345,KXS2]
MIBenum: 2050
Source: IBM Keyboard layouts and code pages, PN 07G4586 June 1991
Alias: cp863
Alias: 863
Alias: csIBM863

Name: IBM864 [RFC1345,KXS2]
MIBenum: 2051
Source: IBM Keyboard layouts and code pages, PN 07G4586 June 1991
Alias: cp864
Alias: csIBM864

Name: IBM865 [RFC1345,KXS2]
MIBenum: 2052
Source: IBM DOS 3.3 Ref (Abridged), 94X9575 (Feb 1987)
Alias: cp865
Alias: 865
Alias: csIBM865

Name: IBM866 [Pond]
MIBenum: 2086
Source: IBM NLDG Volume 2 (SE09-8002-03) August 1994
Alias: cp866
Alias: 866
Alias: csIBM866

Name: IBM868 [RFC1345,KXS2]

MIBenum: 2053
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP868
Alias: cp-ar
Alias: csIBM868

Name: IBM869 [RFC1345,KXS2]
MIBenum: 2054
Source: IBM Keyboard layouts and code pages, PN 07G4586 June 1991
Alias: cp869
Alias: 869
Alias: cp-gr
Alias: csIBM869

Name: IBM870 [RFC1345,KXS2]
MIBenum: 2055
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP870
Alias: ebcdic-cp-roece
Alias: ebcdic-cp-yu
Alias: csIBM870

Name: IBM871 [RFC1345,KXS2]
MIBenum: 2056
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP871
Alias: ebcdic-cp-is
Alias: csIBM871

Name: IBM880 [RFC1345,KXS2]
MIBenum: 2057
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp880
Alias: EBCDIC-Cyrillic
Alias: csIBM880

Name: IBM891 [RFC1345,KXS2]
MIBenum: 2058
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp891
Alias: csIBM891

Name: IBM903 [RFC1345,KXS2]
MIBenum: 2059
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp903

Alias: csIBM903

Name: IBM904 [RFC1345,KXS2]
MIBenum: 2060
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: cp904
Alias: 904
Alias: csIBBM904

Name: IBM905 [RFC1345,KXS2]
MIBenum: 2061
Source: IBM 3174 Character Set Ref, GA27-3831-02, March 1990
Alias: CP905
Alias: ebcdic-cp-tr
Alias: csIBM905

Name: IBM918 [RFC1345,KXS2]
MIBenum: 2062
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP918
Alias: ebcdic-cp-ar2
Alias: csIBM918

Name: IBM1026 [RFC1345,KXS2]
MIBenum: 2063
Source: IBM NLS RM Vol2 SE09-8002-01, March 1990
Alias: CP1026
Alias: csIBM1026

Name: EBCDIC-AT-DE [RFC1345,KXS2]
MIBenum: 2064
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csIBMEBCDICATDE

Name: EBCDIC-AT-DE-A [RFC1345,KXS2]
MIBenum: 2065
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICATDEA

Name: EBCDIC-CA-FR [RFC1345,KXS2]
MIBenum: 2066
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICCAFR

Name: EBCDIC-DK-NO [RFC1345,KXS2]
MIBenum: 2067

Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICDKNO

Name: EBCDIC-DK-NO-A [RFC1345,KXS2]
MIBenum: 2068
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICDKNOA

Name: EBCDIC-FI-SE [RFC1345,KXS2]
MIBenum: 2069
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICFISE

Name: EBCDIC-FI-SE-A [RFC1345,KXS2]
MIBenum: 2070
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICFISEA

Name: EBCDIC-FR [RFC1345,KXS2]
MIBenum: 2071
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICFR

Name: EBCDIC-IT [RFC1345,KXS2]
MIBenum: 2072
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICIT

Name: EBCDIC-PT [RFC1345,KXS2]
MIBenum: 2073
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICPT

Name: EBCDIC-ES [RFC1345,KXS2]
MIBenum: 2074
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICES

Name: EBCDIC-ES-A [RFC1345,KXS2]
MIBenum: 2075
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICESA

Name: EBCDIC-ES-S [RFC1345,KXS2]
MIBenum: 2076
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987

Alias: csEBCDICESS

Name: EBCDIC-UK [RFC1345,KXS2]
MIBenum: 2077
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICUK

Name: EBCDIC-US [RFC1345,KXS2]
MIBenum: 2078
Source: IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987
Alias: csEBCDICUS

Name: UNKNOWN-8BIT [RFC1428]
MIBenum: 2079
Alias: csUnknown8BiT

Name: MNEMONIC [RFC1345,KXS2]
MIBenum: 2080
Source: RFC 1345, also known as "mnemonic+ascii+38"
Alias: csMnemonic

Name: MNEM [RFC1345,KXS2]
MIBenum: 2081
Source: RFC 1345, also known as "mnemonic+ascii+8200"
Alias: csMnem

Name: VISCII [RFC1456]
MIBenum: 2082
Source: RFC 1456
Alias: csVISCII

Name: VIQR [RFC1456]
MIBenum: 2083
Source: RFC 1456
Alias: csVIQR

Name: KOI8-R (preferred MIME name) [RFC1489]
MIBenum: 2084
Source: RFC 1489, based on GOST-19768-74, ISO-6937/8,
INIS-Cyrillic, ISO-5427.
Alias: csKOI8R

Name: KOI8-U [RFC2319]
MIBenum: 2088
Source: RFC 2319

Name: IBM00858
MIBenum: 2089
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM00858) [Mahdi]
Alias: CCSID00858
Alias: CP00858
Alias: PC-Multilingual-850+euro

Name: IBM00924
MIBenum: 2090
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM00924) [Mahdi]
Alias: CCSID00924
Alias: CP00924
Alias: ebcdic-Latin9--euro

Name: IBM01140
MIBenum: 2091
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM01140) [Mahdi]
Alias: CCSID01140
Alias: CP01140
Alias: ebcdic-us-37+euro

Name: IBM01141
MIBenum: 2092
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM01141) [Mahdi]
Alias: CCSID01141
Alias: CP01141
Alias: ebcdic-de-273+euro

Name: IBM01142
MIBenum: 2093
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM01142) [Mahdi]
Alias: CCSID01142
Alias: CP01142
Alias: ebcdic-dk-277+euro
Alias: ebcdic-no-277+euro

Name: IBM01143
MIBenum: 2094
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM01143) [Mahdi]
Alias: CCSID01143
Alias: CP01143
Alias: ebcdic-fi-278+euro
Alias: ebcdic-se-278+euro

Name: IBM01144
MIBenum: 2095

Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM01144) [Mahdi]
Alias: CCSID01144
Alias: CP01144
Alias: ebcdic-it-280+euro

Name: IBM01145
MIBenum: 2096
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM01145) [Mahdi]
Alias: CCSID01145
Alias: CP01145
Alias: ebcdic-es-284+euro

Name: IBM01146
MIBenum: 2097
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM01146) [Mahdi]
Alias: CCSID01146
Alias: CP01146
Alias: ebcdic-gb-285+euro

Name: IBM01147
MIBenum: 2098
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM01147) [Mahdi]
Alias: CCSID01147
Alias: CP01147
Alias: ebcdic-fr-297+euro

Name: IBM01148
MIBenum: 2099
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM01148) [Mahdi]
Alias: CCSID01148
Alias: CP01148
Alias: ebcdic-international-500+euro

Name: IBM01149
MIBenum: 2100
Source: IBM See (http://www.iana.org/assignments/charset-reg/IBM01149) [Mahdi]
Alias: CCSID01149
Alias: CP01149
Alias: ebcdic-is-871+euro

Name: Big5-HKSCS [Yick]
MIBenum: 2101
Source: See (http://www.iana.org/assignments/charset-reg/Big5-HKSCS)
Alias: None

Name: IBM1047 [Robrigado]

MIBenum: 2102
Source: IBM1047 (EBCDIC Latin 1/Open Systems)
http://www-1.ibm.com/servers/eserver/iseries/software/globalization/pdf/cp01047z.pdf
Alias: IBM-1047

Name: PTCP154 [Uskov]
MIBenum: 2103
Source: See (http://www.iana.org/assignments/charset-reg/PTCP154)
Alias: csPTCP154
Alias: PT154
Alias: CP154
Alias: Cyrillic-Asian

Name: Amiga-1251
MIBenum: 2104
Source: See (http://www.amiga.ultranet.ru/Amiga-1251.html)
Alias: Ami1251
Alias: Amiga1251
Alias: Ami-1251
(Aliases are provided for historical reasons and should not be used)
[Malyshev]

Name: KOI7-switched
MIBenum: 2105
Source: See <http://www.iana.org/assignments/charset-reg/KOI7-switched>
Aliases: None

Name: UNICODE-1-1 [RFC1641]
MIBenum: 1010
Source: RFC 1641
Alias: csUnicode11

Name: SCSU
MIBenum: 1011
Source: SCSU See (http://www.iana.org/assignments/charset-reg/SCSU) [Scherer]
Alias: None

Name: UTF-7 [RFC2152]
MIBenum: 1012
Source: RFC 2152
Alias: None

Name: UTF-16BE [RFC2781]
MIBenum: 1013
Source: RFC 2781
Alias: None

Name: UTF-16LE [RFC2781]
MIBenum: 1014
Source: RFC 2781
Alias: None

Name: UTF-16 [RFC2781]
MIBenum: 1015
Source: RFC 2781
Alias: None

Name: CESU-8 [Phipps]
MIBenum: 1016
Source: <http://www.unicode.org/unicode/reports/tr26>
Alias: csCESU-8

Name: UTF-32 [Davis]
MIBenum: 1017
Source: <http://www.unicode.org/unicode/reports/tr19/>
Alias: None

Name: UTF-32BE [Davis]
MIBenum: 1018
Source: <http://www.unicode.org/unicode/reports/tr19/>
Alias: None

Name: UTF-32LE [Davis]
MIBenum: 1019
Source: <http://www.unicode.org/unicode/reports/tr19/>
Alias: None

Name: BOCU-1 [Scherer]
MIBenum: 1020
Source: http://www.unicode.org/notes/tn6/
Alias: csBOCU-1

Name: UNICODE-1-1-UTF-7 [RFC1642]
MIBenum: 103
Source: RFC 1642
Alias: csUnicode11UTF7

Name: UTF-8 [RFC3629]
MIBenum: 106
Source: RFC 3629
Alias: None

Name: ISO-8859-13
MIBenum: 109
Source: ISO See (http://www.iana.org/assignments/charset-reg/iso-8859-13)[Tumasonis]
Alias: None

Name: ISO-8859-14
MIBenum: 110
Source: ISO See (http://www.iana.org/assignments/charset-reg/iso-8859-14) [Simonsen]
Alias: iso-ir-199
Alias: ISO_8859-14:1998
Alias: ISO_8859-14
Alias: latin8
Alias: iso-celtic
Alias: l8

Name: ISO-8859-15
MIBenum: 111
Source: ISO
Please see: <http://www.iana.org/assignments/charset-reg/ISO-8859-15>
Alias: ISO_8859-15
Alias: Latin-9

Name: ISO-8859-16
MIBenum: 112
Source: ISO
Alias: iso-ir-226
Alias: ISO_8859-16:2001
Alias: ISO_8859-16
Alias: latin10
Alias: l10

Name: GBK
MIBenum: 113
Source: Chinese IT Standardization Technical Committee
Please see: <http://www.iana.org/assignments/charset-reg/GBK>
Alias: CP936
Alias: MS936
Alias: windows-936

Name: GB18030
MIBenum: 114
Source: Chinese IT Standardization Technical Committee
Please see: <http://www.iana.org/assignments/charset-reg/GB18030>
Alias: None

Name: OSD_EBCDIC_DF04_15

MIBenum: 115
Source: Fujitsu-Siemens standard mainframe EBCDIC encoding
Please see: <http://www.iana.org/assignments/charset-reg/OSD-EBCDIC-DF04-15>
Alias: None

Name: OSD_EBCDIC_DF03_IRV
MIBenum: 116
Source: Fujitsu-Siemens standard mainframe EBCDIC encoding
Please see: <http://www.iana.org/assignments/charset-reg/OSD-EBCDIC-DF03-IRV>
Alias: None

Name: OSD_EBCDIC_DF04_1
MIBenum: 117
Source: Fujitsu-Siemens standard mainframe EBCDIC encoding
Please see: <http://www.iana.org/assignments/charset-reg/OSD-EBCDIC-DF04-1>
Alias: None

Name: JIS_Encoding
MIBenum: 16
Source: JIS X 0202-1991. Uses ISO 2022 escape sequences to
shift code sets as documented in JIS X 0202-1991.
Alias: csJISEncoding

Name: Shift_JIS (preferred MIME name)
MIBenum: 17
Source: This charset is an extension of csHalfWidthKatakana by
adding graphic characters in JIS X 0208. The CCS's are
JIS X0201:1997 and JIS X0208:1997. The
complete definition is shown in Appendix 1 of JIS
X0208:1997.
This charset can be used for the top-level media type "text".
Alias: MS_Kanji
Alias: csShiftJIS

Name: Extended_UNIX_Code_Packed_Format_for_Japanese
MIBenum: 18
Source: Standardized by OSF, UNIX International, and UNIX Systems
Laboratories Pacific. Uses ISO 2022 rules to select
code set 0: US-ASCII (a single 7-bit byte set)
code set 1: JIS X0208-1990 (a double 8-bit byte set)
restricted to A0-FF in both bytes
code set 2: Half Width Katakana (a single 7-bit byte set)
requiring SS2 as the character prefix
code set 3: JIS X0212-1990 (a double 7-bit byte set)
restricted to A0-FF in both bytes
requiring SS3 as the character prefix

Alias: csEUCPkdFmtJapanese
Alias: EUC-JP (preferred MIME name)

Name: Extended_UNIX_Code_Fixed_Width_for_Japanese
MIBenum: 19
Source: Used in Japan. Each character is 2 octets.
code set 0: US-ASCII (a single 7-bit byte set)
1st byte = 00
2nd byte = 20-7E
code set 1: JIS X0208-1990 (a double 7-bit byte set)
restricted to A0-FF in both bytes
code set 2: Half Width Katakana (a single 7-bit byte set)
1st byte = 00
2nd byte = A0-FF
code set 3: JIS X0212-1990 (a double 7-bit byte set)
restricted to A0-FF in
the first byte
and 21-7E in the second byte
Alias: csEUCFixWidJapanese

Name: ISO-10646-UCS-Basic
MIBenum: 1002
Source: ASCII subset of Unicode. Basic Latin = collection 1
See ISO 10646, Appendix A
Alias: csUnicodeASCII

Name: ISO-10646-Unicode-Latin1
MIBenum: 1003
Source: ISO Latin-1 subset of Unicode. Basic Latin and Latin-1
Supplement = collections 1 and 2. See ISO 10646,
Appendix A. See RFC 1815.
Alias: csUnicodeLatin1
Alias: ISO-10646

Name: ISO-10646-J-1
Source: ISO 10646 Japanese, see RFC 1815.

Name: ISO-Unicode-IBM-1261
MIBenum: 1005
Source: IBM Latin-2, -3, -5, Extended Presentation Set, GCSGID: 1261
Alias: csUnicodeIBM1261

Name: ISO-Unicode-IBM-1268
MIBenum: 1006
Source: IBM Latin-4 Extended Presentation Set, GCSGID: 1268
Alias: csUnicodeIBM1268

Name: ISO-Unicode-IBM-1276
MIBenum: 1007
Source: IBM Cyrillic Greek Extended Presentation Set, GCSGID: 1276
Alias: csUnicodeIBM1276

Name: ISO-Unicode-IBM-1264
MIBenum: 1008
Source: IBM Arabic Presentation Set, GCSGID: 1264
Alias: csUnicodeIBM1264

Name: ISO-Unicode-IBM-1265
MIBenum: 1009
Source: IBM Hebrew Presentation Set, GCSGID: 1265
Alias: csUnicodeIBM1265

Name: ISO-8859-1-Windows-3.0-Latin-1 [HP-PCL5]
MIBenum: 2000
Source: Extended ISO 8859-1 Latin-1 for Windows 3.0.
PCL Symbol Set id: 9U
Alias: csWindows30Latin1

Name: ISO-8859-1-Windows-3.1-Latin-1 [HP-PCL5]
MIBenum: 2001
Source: Extended ISO 8859-1 Latin-1 for Windows 3.1.
PCL Symbol Set id: 19U
Alias: csWindows31Latin1

Name: ISO-8859-2-Windows-Latin-2 [HP-PCL5]
MIBenum: 2002
Source: Extended ISO 8859-2. Latin-2 for Windows 3.1.
PCL Symbol Set id: 9E
Alias: csWindows31Latin2

Name: ISO-8859-9-Windows-Latin-5 [HP-PCL5]
MIBenum: 2003
Source: Extended ISO 8859-9. Latin-5 for Windows 3.1
PCL Symbol Set id: 5T
Alias: csWindows31Latin5

Name: Adobe-Standard-Encoding [Adobe]
MIBenum: 2005
Source: PostScript Language Reference Manual
PCL Symbol Set id: 10J
Alias: csAdobeStandardEncoding

Name: Ventura-US [HP-PCL5]
MIBenum: 2006
Source: Ventura US. ASCII plus characters typically used in
publishing, like pilcrow, copyright, registered, trade mark,
section, dagger, and double dagger in the range A0 (hex)
to FF (hex).
PCL Symbol Set id: 14J
Alias: csVenturaUS

Name: Ventura-International [HP-PCL5]
MIBenum: 2007
Source: Ventura International. ASCII plus coded characters similar
to Roman8.
PCL Symbol Set id: 13J
Alias: csVenturaInternational

Name: PC8-Danish-Norwegian [HP-PCL5]
MIBenum: 2012
Source: PC Danish Norwegian
8-bit PC set for Danish Norwegian
PCL Symbol Set id: 11U
Alias: csPC8DanishNorwegian

Name: PC8-Turkish [HP-PCL5]
MIBenum: 2014
Source: PC Latin Turkish. PCL Symbol Set id: 9T
Alias: csPC8Turkish

Name: IBM-Symbols [IBM-CIDT]
MIBenum: 2015
Source: Presentation Set, CPGID: 259
Alias: csIBMSymbols

Name: IBM-Thai [IBM-CIDT]
MIBenum: 2016
Source: Presentation Set, CPGID: 838
Alias: csIBMThai

Name: HP-Legal [HP-PCL5]
MIBenum: 2017
Source: PCL 5 Comparison Guide, Hewlett-Packard,
HP part number 5961-0510, October 1992
PCL Symbol Set id: 1U
Alias: csHPLegal

Name: HP-Pi-font [HP-PCL5]

MIBenum: 2018
Source: PCL 5 Comparison Guide, Hewlett-Packard,
HP part number 5961-0510, October 1992
PCL Symbol Set id: 15U
Alias: csHPPiFont

Name: HP-Math8 [HP-PCL5]
MIBenum: 2019
Source: PCL 5 Comparison Guide, Hewlett-Packard,
HP part number 5961-0510, October 1992
PCL Symbol Set id: 8M
Alias: csHPMath8

Name: Adobe-Symbol-Encoding [Adobe]
MIBenum: 2020
Source: PostScript Language Reference Manual
PCL Symbol Set id: 5M
Alias: csHPPSMath

Name: HP-DeskTop [HP-PCL5]
MIBenum: 2021
Source: PCL 5 Comparison Guide, Hewlett-Packard,
HP part number 5961-0510, October 1992
PCL Symbol Set id: 7J
Alias: csHPDesktop

Name: Ventura-Math [HP-PCL5]
MIBenum: 2022
Source: PCL 5 Comparison Guide, Hewlett-Packard,
HP part number 5961-0510, October 1992
PCL Symbol Set id: 6M
Alias: csVenturaMath

Name: Microsoft-Publishing [HP-PCL5]
MIBenum: 2023
Source: PCL 5 Comparison Guide, Hewlett-Packard,
HP part number 5961-0510, October 1992
PCL Symbol Set id: 6J
Alias: csMicrosoftPublishing

Name: Windows-31J
MIBenum: 2024
Source: Windows Japanese. A further extension of Shift_JIS
to include NEC special characters (Row 13), NEC
selection of IBM extensions (Rows 89 to 92), and IBM
extensions (Rows 115 to 119). The CCS's are

JIS X0201:1997, JIS X0208:1997, and these extensions.
This charset can be used for the top-level media type "text",
but it is of limited or specialized use (see RFC2278).
PCL Symbol Set id: 19K
Alias: csWindows31J

Name: GB2312 (preferred MIME name)
MIBenum: 2025
Source: Chinese for People's Republic of China (PRC) mixed one byte,
two byte set:
20-7E = one byte ASCII
A1-FE = two byte PRC Kanji
See GB 2312-80
PCL Symbol Set Id: 18C
Alias: csGB2312

Name: Big5 (preferred MIME name)
MIBenum: 2026
Source: Chinese for Taiwan Multi-byte set.
PCL Symbol Set Id: 18T
Alias: csBig5

Name: windows-1250
MIBenum: 2250
Source: Microsoft (http://www.iana.org/assignments/charset-reg/windows-1250) [Lazhintseva]
Alias: None

Name: windows-1251
MIBenum: 2251
Source: Microsoft (http://www.iana.org/assignments/charset-reg/windows-1251) [Lazhintseva]
Alias: None

Name: windows-1252
MIBenum: 2252
Source: Microsoft (http://www.iana.org/assignments/charset-reg/windows-1252) [Wendt]
Alias: None

Name: windows-1253
MIBenum: 2253
Source: Microsoft (http://www.iana.org/assignments/charset-reg/windows-1253) [Lazhintseva]
Alias: None

Name: windows-1254
MIBenum: 2254
Source: Microsoft (http://www.iana.org/assignments/charset-reg/windows-1254) [Lazhintseva]
Alias: None

Name: windows-1255
MIBenum: 2255
Source: Microsoft (http://www.iana.org/assignments/charset-reg/windows-1255) [Lazhintseva]
Alias: None

Name: windows-1256
MIBenum: 2256
Source: Microsoft (http://www.iana.org/assignments/charset-reg/windows-1256) [Lazhintseva]
Alias: None

Name: windows-1257
MIBenum: 2257
Source: Microsoft (http://www.iana.org/assignments/charset-reg/windows-1257) [Lazhintseva]
Alias: None

Name: windows-1258
MIBenum: 2258
Source: Microsoft (http://www.iana.org/assignments/charset-reg/windows-1258) [Lazhintseva]
Alias: None

Name: TIS-620
MIBenum: 2259
Source: Thai Industrial Standards Institute (TISI) [Tantsetthi]

Name: HZ-GB-2312
MIBenum: 2085
Source: RFC 1842, RFC 1843 [RFC1842, RFC1843]

REFERENCES

[RFC1345] Simonsen, K., "Character Mnemonics & Character Sets", RFC 1345, Rationel Almen Planlaegning, Rationel Almen Planlaegning, June 1992.

[RFC1428] Vaudreuil, G., "Transition of Internet Mail from Just-Send-8 to 8bit-SMTP/MIME", RFC1428, CNRI, February 1993.

[RFC1456] Vietnamese Standardization Working Group, "Conventions for Encoding the Vietnamese Language VISCII: VIetnamese Standard Code for Information Interchange VIQR: VIetnamese Quoted-Readable Specification Revision 1.1", RFC 1456, May 1993.

[RFC1468] Murai, J., Crispin, M., and E. van der Poel, "Japanese Character Encoding for Internet Messages", RFC 1468, Keio University, Panda Programming, June 1993.

[RFC1489] Chernov, A., "Registration of a Cyrillic Character Set", RFC1489, RELCOM Development Team, July 1993.

[RFC1554] Ohta, M., and K. Handa, "ISO-2022-JP-2: Multilingual Extension of ISO-2022-JP", RFC1554, Tokyo Institute of Technology, ETL, December 1993.

[RFC1556] Nussbacher, H., "Handling of Bi-directional Texts in MIME", RFC1556, Israeli Inter-University, December 1993.

[RFC1557] Choi, U., Chon, K., and H. Park, "Korean Character Encoding for Internet Messages", KAIST, Solvit Chosun Media, December 1993.

[RFC1641] Goldsmith, D., and M. Davis, "Using Unicode with MIME", RFC1641, Taligent, Inc., July 1994.

[RFC1642] Goldsmith, D., and M. Davis, "UTF-7", RFC1642, Taligent, Inc., July 1994.

[RFC1815] Ohta, M., "Character Sets ISO-10646 and ISO-10646-J-1", RFC 1815, Tokyo Institute of Technology, July 1995.

[Adobe] Adobe Systems Incorporated, PostScript Language Reference Manual, second edition, Addison-Wesley Publishing Company, Inc., 1990.

[ECMA Registry] ISO-IR: International Register of Escape Sequences http://www.itscj.ipsj.or.jp/ISO-IE/ Note: The current registration authority is IPSJ/ITSCJ, Japan.

[HP-PCL5] Hewlett-Packard Company, "HP PCL 5 Comparison Guide", (P/N 5021-0329) pp B-13, 1996.

[IBM-CIDT] IBM Corporation, "ABOUT TYPE: IBM's Technical Reference for Core Interchange Digitized Type", Publication number S544-3708-01

[RFC1842] Wei, Y., J. Li, and Y. Jiang, "ASCII Printable Characters-Based Chinese Character Encoding for Internet

Messages", RFC 1842, Harvard University, Rice University, University of Maryland, August 1995.

[RFC1843] Lee, F., "HZ - A Data Format for Exchanging Files of Arbitrarily Mixed Chinese and ASCII Characters", RFC 1843, Stanford University, August 1995.

[RFC2152] Goldsmith, D., M. Davis, "UTF-7: A Mail-Safe Transformation Format of Unicode", RFC 2152, Apple Computer, Inc., Taligent Inc., May 1997.

[RFC2279] Yergeau, F., "UTF-8, A Transformation Format of ISO 10646", RFC 2279, Alis Technologies, January, 1998.

[RFC2781] Hoffman, P., Yergeau, F., "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC3629, November 2003.

PEOPLE

[KXS2] Keld Simonsen <Keld.Simonsen@dkuug.dk>

[Choi] Woohyong Choi <whchoi@cosmos.kaist.ac.kr>

[Davis] Mark Davis, <mark@unicode.org>, April 2002.

[Lazhintseva] Katya Lazhintseva, <katyal@MICROSOFT.com>, May 1996.

[Mahdi] Tamer Mahdi, <tamer@ca.ibm.com>, August 2000.

[Malyshev] Michael Malyshev, <michael_malyshev@mail.ru>, January 2004

[Murai] Jun Murai <jun@wide.ad.jp>

[Nussbacher] Hank Nussbacher, <hank@vm.tau.ac.il>

[Ohta] Masataka Ohta, <mohta@cc.titech.ac.jp>, July 1995.

[Phipps] Toby Phipps, <tphipps@peoplesoft.com>, March 2002.

[Pond] Rick Pond, <rickpond@vnet.ibm.com>, March 1997.

[Robrigado] Reuel Robrigado, <reuelr@ca.ibm.com>, September 2002.

[Scherer] Markus Scherer, <markus.scherer@jtcsv.com>, August 2000,
September 2002.

[Simonsen] Keld Simonsen, <Keld.Simonsen@rap.dk>, August 2000.

[Tantsetthi] Trin Tantsetthi, <trin@mozart.inet.co.th>, September 1998.

[Tumasonis] Vladas Tumasonis, <vladas.tumasonis@maf.vu.lt>, August 2000.

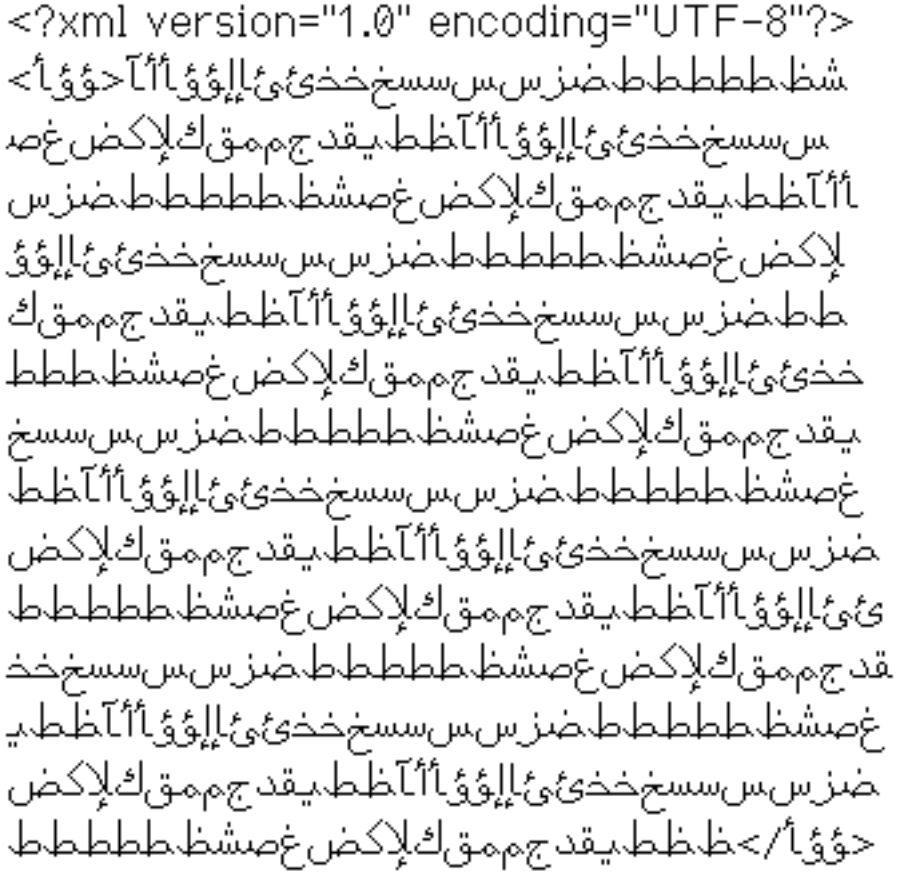
[Uskov] Alexander Uskov, <auskov@idc.kz>, September 2002.

[Wendt] Chris Wendt, <christw@microsoft.com>, December 1999.

[Yick] Nicky Yick, <cliac@itsd.gcn.gov.hk>, October 2000.

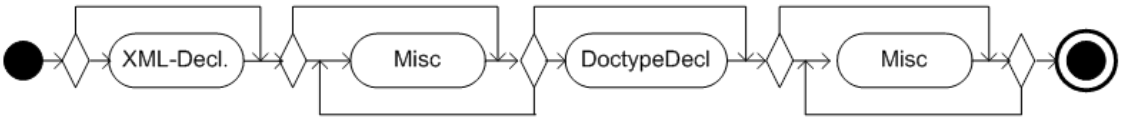
[]

```
<?xml version="1.0" encoding="Shift_JIS" standalone="yes"?>
<kougi>
  <title begin="二千一年三月二十一日三時三十分＋九時"＞選挙義務講義ＸＭＬ＜/title>
  <organization>アウグスブルグ大学コンピュータ・サイエンス過分、工学と家政と形成の大学</organization>
</kougi>
```

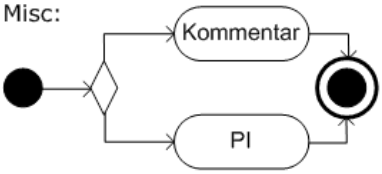

<?xml version="1.0" encoding="UTF-8"?>

Dokument:

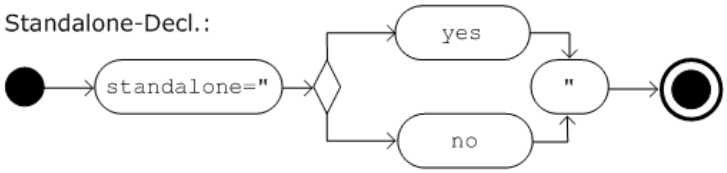
Prolog:



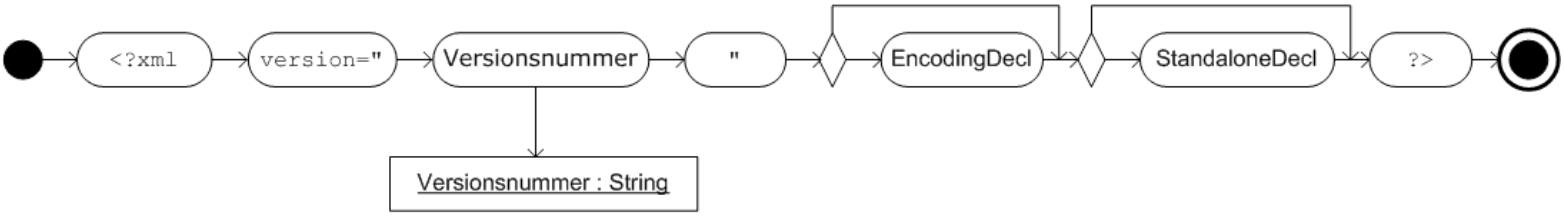
Misc:



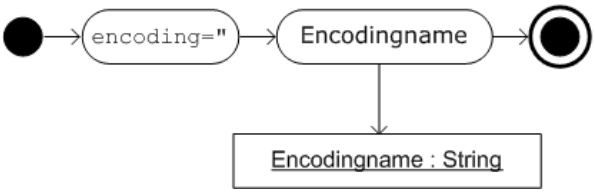
Standalone-Decl.:



XML-Decl.:



Encoding-Decl.:



Computervermittelte Kommunikation

Kapitel 12: OSI-Schicht 6: Presentation Layer -- Datendarstellungsschicht

Teil I: Allgemeines, ASN.1, ASCII, ISO ISO 8859, UNICODE, ISO/IEC 10646, UCS, UTF, PostScript, Acrobat

---

von Margarete Payer

mailto: payer@hbi-stuttgart.de

Zitierweise / cite as:

> **Payer, Margarete <1942 -- >:** Computervermittelte Kommunikation. -- Kapitel 12: OSI-Schicht 6: Presentation Layer -- Datendarstellungsschicht. -- Teil 1: Allgemeines, ASN.1, ASCII, ISO ISO 8859, UNICODE, ISO/IEC 10646, UCS, UTF, PostScript, Acrobat. -- Fassung vom 2001-01-21. -- URL: http://www.payer.de/cmc/cmcs1201.htm. -- [Stichwort].

Erstmals publiziert: 1995

Überarbeitungen: 23. Juni 1997; 16.6.1999 [grundlegende Neubearbeitung und Erweiterung]; 2001-01-21 [Verbesserung kleiner Fehler]

Anlass: Lehrveranstaltungen an der HBI Stuttgart

Zur Inhaltsübersicht von Margarete Payer: Computervermittelte Kommunikation.

12.0. Übersicht

12.1. Merkmale der Datendarstellungsschicht

Die Datendarstellungsschicht dient dazu, die Daten so darzustellen, wie sie tatsächlich auf der Leitung übertragen werden sollen. Diese Darstellung muss nicht identisch sein mit der Darstellung auf der Anwendungsschicht.

Die Darstellungsschicht ist zuständig für:

- die Darstellung (Syntax) der Daten, die übertragen werden sollen
- die Darstellung der Datenstrukturen
- die Darstellung der Aktionen an diesen Datenstrukturen.

Die Darstellungsschicht ist nur für die Syntax (die Darstellung der Daten) zuständig, nicht für die Semantik (d.h. die Bedeutung der Daten). Für die Semantik sind nur die Anwendungen zuständig. Die Darstellungsschicht soll gewährleisten, **dass die Anwendungen Syntax-unabhängig miteinander kommunizieren können**. Die Protokolle der Darstellungsschicht wandeln also die anwendungsspezifischen Syntaxe in eine gemeinsame Syntax um, sodass sich die Anwendungen nicht um die Kompatibilität ihrer Syntax zu kümmern brauchen.

Aufgaben der Datendarstellungsschicht:

- **Datenkompression** (data compression) durch:
  - Nachrichtenreduktion durch Kürzung redundanter Teile
  - Datenvorverarbeitung durch intelligente Terminals

    - Messwertreduzierung: es werden nur Werte übertragen, die sich gegenüber der vorhergehenden Übertragung geändert haben
    - Arbeiten mit definierten Datenstrukturen
- **Umcodierung** (data format conversion) z.B. bei Kommunikation zwischen Rechnern mit ASCII und EBCDI.
- **Datenverschlüsselung und -entschlüsselung** (data encryption / decryption)
- **Datenbank-Zugriff und -Verwaltung** (data base management / access) (z.B. Zugriffsberechtigung, Entlastung der Datenübertragung durch intelligente Terminals)

Funktionen innerhalb der Datendarstellungsschicht:

- Anforderung der Eröffnung einer Sitzung Daten-Übertragung
- Abstimmung über Syntax
- Umwandlung der Syntax, inklusive Datenumwandlung,
- Formatierung und spezielle Umwandlungen wie Datenkompression
- Anforderung der Beendigung einer Sitzung

12.2. Protokolle für die Datendarstellungsschicht

OSI Service Definition:

X.216 Presentation service definition

ISO 8822 Connection oriented presentation service definition

OSI Protocol Specification:

X.226 Presentation protocol specification

ISO 8823 Connection oriented presentation protocol specification

ISO 8824: Abstract Syntax Notation 1 **(ASN.1)**

LAN:

ISO 8822: Connection oriented presentation service definition

ISO 8823: Connection oriented presentation protocol specification

ISO 8824: Abstract syntax notation One (ASN.1)

ISO 8825: Basic encoding rules for ASN.1

Packet-switched data network:

X.216: OSI Presentation service definition note

Public-switched telephone network:

T.50: International reference alphabet -- 7-bit coded character set for information interchange

T.51: Latin based coded character sets for telematic services

---

12.3. ASN.1 -- Abstract Syntax Notation One

12.3.1. Weiterführende Ressourcen

Eine sehr klare Darstellung von ASN.1 ist in:

> **Stallings, William:** Data and computer communications. -- 4. ed. -- London [u.a.] : Prentice Hall, 1994. -- 875 S. -- ISBN 0024154253. -- S. 639-672. -- {Wenn Sie HIER klicken, können Sie dieses Buch bei **amazon.de** bestellen}

12.3.2. Merkmale von ASN.1

ISO 8824 (CCITT/ITU X.208) -- Abstract Syntax Notation One (ASN.1)

ISO 8825 (CCITT/ITU X.209) -- Basic Encoding Rules (BER)

ASN.1 bietet eine Grammatik zur Definition von Datenstrukturen sowie Festlegungen zur Umsetzung von Datenstrukturen und Elementen in ein netzeinheitliches Format (Transfer Syntax).

ASN.1 hat sich weitgehend durchgesetzt bei der Entwicklung von OSI-bezogenen Standards und TCP/IP-bezogenen Standards. Man verwendet ASN.1 um das Format von mittels der Protokolle ausgetauschten Daten sowie die diesbezüglichen Operationen zu definieren.

Die ASN.1-Umwandlung kann bis zu 80% (!) des CPU-Aufwandes für ein Paket bis zur Applikation hin ausmachen. ASN.1 ist nicht flexibel, sodass eine Verbesserung durch übliche Techniken der Leistungssteigerung (z.B. Parallelverarbeitung) nicht möglich ist.

ASN.1 ist eine Notation zur Beschreibung

- abstrakter Datentypen (abstract data types)
- Werte (values)

12.3.3. Grundbegriffe von ASN.1

12.3.3.1. Module

Der Grundbaustein einer ASN.1-Spezifikation ist das Modul (module).

ASN.1 kann benutzt werden, um Datenstrukturen zu definieren. Diese Definition geschieht in Form eines benannten Moduls. Der Name des Moduls wird dann zur Bezeichnung der Datenstruktur verwendet.

Struktur eines Modul:

```
<modulreference> DEFINITIONS::=
BEGIN
 EXPORTS
 IMPORTS
 AssignmentList
END
```

Erklärung:

modulreference:
Name des Moduls
EXPORTS:
Definitionen, die aus diesem Modul von anderen Modulen übernommen werden können
IMPORTS:
Definitionen, die aus anderen Modulen in dieses Modul übernommen werden sollen
AssignmentList:
Typen-Zuweisungen (type assignments), Wert-Zuweisungen (value assignments), Macrodefinitionen
Typen- und Wert-Zuweisungen haben die Form:

<name>::<description>

12.3.3.2. Darstellungs-Konventionen

ASN.1 types und values werden in einer Programmiersprache-artigen Notation dargestellt. Dabei gelten folgende Regeln:

- das Layout hat keine Bedeutung. Mehrfach-Spatien und Zeilenvorschub werden als einfaches Spatium betrachtet
- kommentierende Bemerkungen stehen zwischen -- und --, bzw. zwischen -- und Zeilenvorschub
- identifiers (Namen von values und fields) und type references (Namen von types) bestehen aus Groß- und Kleinbuchstaben, Ziffern, Bindestrichen (-) und Spatien. Identifiers beginnen mit Kleinbuchstaben, references beginnen mit Großbuchstaben

---

12.3.3.3. Abstakte Daten-Typen (abstract data types)

Ein type ist eine Menge von Werten (values). Für einige types gibt es eine endliche Anzahl von möglichen Werten, für andere eine unendliche. Ein Wert ist umgekehrt ein Element der type-Menge.

Arten von types:

- **simple types:** sind elementar und haben keine weiteren Komponenten

- **structured types:** bestehen aus Komponenten
- **tagged types:** sind von anderen types abgeleitet
- other types:
  - CHOICE: ein oder mehrere Alternativen
  - ANY: ein beliebiger Wert eines beliebigen type

Types und Werten kann man mit mittels des ASN.1 **assignement operators ::=** einen Namen zuordnen. Dieser Name kann dann verwendet werden bei der Definition anderer types und Werte.

Jeder ASN.1 type außer CHOICE und ANY hat einen tag (Identifikator). Jeder tag besteht aus einer tag-classe und einer nicht-negativen tag-number.

Tag classes:

- **universal:** für types, deren Bedeutung in allen Anwendungen gleich ist. In ISO 8824 (X.208) definiert
- **application-wide:** für types, deren Bedeutung für eine bestimmte Anwendung spezifisch ist, z.B. für X.500 Directory Services. Types in zwei verschiedenen Anwendungen können denselben application tag haben, aber dennoch zwei unterschiedliche, anwendungsspezifische Bedeutungen
- **private:** für types, deren Bedeutung innerhalb eines bestimmten Unternehmens einheitlich ist
- **context-specific:** für types, deren Bedeutung spezifisch ist für Komponenten innerhalb eines structured type. Solche tags für Komponenten können innerhalb zweier verschiedener structured types dasselbe tag haben, aber dennoch unterschiedliche, kontextspezifische Bedeutungen haben

---

Universal types (Auswahl):

- **Basic Types**
  - UNIVERSAL 1: BOOLEAN. Werte: TRUE, FALSE
  - UNIVERSAL 2: INTEGER: positive und negative ganze Zahlen
  - UNIVERSAL 3: BIT STRING: eine beliebige Reihe von Bits (0, 1)
  - UNIVERSAL 4: OCTET STRING: eine beliebige Reihe von Oktetten (8bit-Werten in dezimaler Darstellung)
  - UNIVERSAL 9: REAL: reelle Zahlen
  - UNIVERSAL 10: ENUMERATED: Aufzählung von Werten, die ein Datentyp annehmen darf
- **Object Types**
  - UNIVERSAL 6: OBJECT IDENTIFIER: eine Reihe von Zeichen, die z.B. einen Algorithmus oder einen Attribute type bestimmen
  - UNIVERSAL 7: OBJECT DESCRIPTOR
- **Character String Types:**
  - UNIVERSAL 18: NumericString: Ziffern 1 bis 9, Spatium
  - UNIVERSAL 19: PrintableString: eine beliebige Reihe von druckbaren Zeichen
  - UNIVERSAL 22: IA5String: eine beliebige Reihe von ASCII-Zeichen
  - UNIVERSAL 25: GraphicString: nach ISO 8824
  - UNIVERSAL 27: GeneralString: allgemeiner Zeichen-String
- **Miscellaneous Types:**
  - UNIVERSAL 5: NULL
  - UNIVERSAL 8: EXTERNAL: in einem externen, Nicht-ASN.1-Dokument definierter Typ
  - UNIVERSAL 24: GeneralizedTime
- **Structured Types:**

- UNIVERSAL 16: SEQUENCE eine geordnete Ansammlung von ein oder mehreren types; SEQUENCE-OF eine geordnete Anordnung von null oder mehreren Vorkommen eines einzigen type
- UNIVERSAL 17: SET eine ungeordnete Ansammlung von ein oder mehreren types; SET OF eine ungeordnete Anordnung von null oder mehreren Vorkommen eines einzigen type

---

12.3.3.4. Beispiel der Definition einer Datenstruktur

Informelle Beschreibung eines persönlichen Datensatzes	**ASN.1 Beschreibung des nebenstehenden einzelnen Datensatzes (record value)**

Name: John P Smith
 	
Title: Director
Employee Number: 51
Date of Hire: 17 September 1971
Name of Spouse: Mary T Smith
Number of Children: 2
 	
Child Information:
Name: Ralph T Smith
Date of Birth: 11 November 1957
 	
Child Information:
Name: Susan B Jones
Date of Birth: 17 July 1959
	`{ {givenName "John", initial "P", familyName "Smith"},`
	
	`title "Director"`
	`number51`
	`dateOfHire "19710917"`
	`nameOfSpouse{givenName "Mary", initial "T", familyName "Smith"},`
	
	`children`
	`{ { {givenName "Ralph", initial "T", familyName "Smith"}`
	`dateOfBirth "19571111"`
	
	`{ {givenName "Susan", initial "B", familyName "Jones"}`
	`dateOfBirth "19590717"}}}`

ASN.1 Beschreibung der Struktur des obigen Datensatzes:

```
PersonelRecord ::= [APPLICATION 0] IMPLICIT SET {
 Name,
 title [0] VisibleString,
 number EmployeeNumber,
 dateOfHire [1] Date,
 nameOfSpouse [2] Name,
 children [3] IMPLICIT SEQUENCE OF ChildInformation DEFAULT {}}

ChildInformation ::= SET {
 Name,
 dateOfBirth [0] Date}

Name ::= [APPLICATION 1] IMPLICIT SEQUENCE {
 givenName VisibleString,
 initial VisibleString,
```

```
 familyName Visible String }

EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER

Date ::= [APPLICATION 3] IMPLICIT VisibleString -- YYYYMMDD
```

Erklärung im Einzelnen in

Stallings, William: Data and computer communications. -- 4. ed. -- London [u.a.] : Prentice Hall, 1994. -- 875 S. -- ISBN 0024154253. -- S. 651-653. -- {Wenn Sie HIER klicken, können Sie dieses Buch bei **amazon.de** bestellen}

12.3.3.5. BER -- Basic Encoding Rules

Die Basic Encoding Rules (BER) geben an, wie man einen ASN.1 value als Oktett-Reihe darstellen kann.

Es gibt drei Darstellungsmethoden. Die Wahl richtet sich nach der Art des value und danach, ob die Länge des value zuvor schon bekannt ist.

- **primitive, definite-length encoding:** für simple types
- **constructed, definite-length encoding:** für structured types mit fester Feldlänge sowie für simple string types mit fester Feldlänge
- **constructed, indefinite-length encoding:** für structured types mit offener Feldlänge sowie für simple string types mit offener Feldlänge

Bei jeder dieser Darstellungsmethoden hat die BER-Kodierung drei bzw. vier Teile:

- **identifier octets:** identifizieren die class und tag number des value und zeigen an, ob die Methode primitive oder constructed ist
- **length octets:** gibt bei den definite length Methoden die Feldlänge an, bei der constructed indefinite-length Methode gibt es an, dass die Feldlänge offen ist
- **contents octets:** geben bei der primitive Methode den Wert des value an, bei den constructed Methoden geben sie die Verkettung der BER-Kodierungen der Komponenten des Wertes an
- **end-of-contents octets:** nur bei contructed, indefinite-length encoding. Dort markieren sie das Ende des Datenfeldes (contents octets)

12.4. Formate zum Austausch von Dokumenten

Es gibt viele Versuche, hard- und software-unabhängige Formate für Dokumente zu schaffen und durchzusetzen. Bisher herrscht aber immer noch Chaos, und ohne Umwandlungsprogramme kommt man kaum aus.

12.4.1. 7-Bit-Kodierung: ASCII-Text

ASCII *(American Standard Code for Information Interchange)* ist ein ISO *(International Organization for Standardization)* Standard (ISO/IEC 646).

ASCII ist ein 8-Bit-Code:

- 7 Bit dienen der Zeichendefinition, d.h. er kann nur 128 Zeichen und Steuerzeichen (Hex 00 bis 20, 7F) definieren
- 1 Bit ist ein Fehler-Check-Bit (Checksum)

Tabelle: US-ASCII (128 Zeichen) mit Hexadezimalkodierung ([ZeilenNr][SpaltenNr]): (A = 41; a = 61)

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Leider blieb es nicht bei US-ASCII, sondern es wurden regionale ASCIIs entwickelt, um z.B. die Umlaute zu kodieren. Dies führt zu den bekannten Problemen mit der Darstellung von Umlauten (die dann z.B. als "[" wiedergegeben werden). So gibt es folgende ASCII-Varianten: International Reference Version (ISO 646. IRV), Deutschland (ISO 646.DE), Schweiz (ISO 646.CH), Französisch-Kanada (ISO 646.CA), Spanien (ISO 646. ES), Finnland (ISO 646.FI) , Frankreich (ISO 646.FR), Großbritannien (ISO 646.GB), Italien (ISO 646.NL), Norwegen/Dänemark (ISO 646.NO), Portugal (ISO 646.PT) , Schweden (ISO 646.SE), Korea (KS C 5363), China (GB 1988-80), Japan (JIS X 201). Alle sind nicht voll miteinander kompatibel.

ASCII-Text kann von fast allen Textbearbeitungsprogrammen verwertet werden. ASCII erlaubt aber keine unterschiedlichen Schriften, Schriftarten, kompliziertere Textformatierungen, Grafiken, Farben. SGML, XML und HTML haben den großen Vorteil, dass ihre Kodierungen in reinem US-ASCII sind!

Weiterführende Ressourcen zu ASCII:

Yahoo Categories:

- http://www.yahoo.com/Computers_and_Internet/Information_and_Documentation/Data_Formats/**ASCII**/. -- Zugriff am 11.6.1999

12.4.2. 8-Bit-Kodierung: ISO 8859, ISCII

Da moderne Computer weniger fehleranfällig sind als ältere Systeme, hat man das 8. Bit auch zur Zeichendefinition verwendet und **Extended ASCII** entwickelt, das 256 Zeichen und Steuerzeichen definiert. Diese 8-Bit-Kodierungen werden als ISO 8859-n international standardisiert.:

ISO 8859-n	Name	Sprachspezifische Zeichensätze
ISO 8859-1	**Latin-1**	für Englisch, Dänisch, Niederländisch, Finnisch, Französisch, Deutsch, Isländisch, Irisch, Italienisch, Bahasa Malaysisch, Norwegisch, Portugiesisch, Spanisch, Schwedisch, Tagalog (Philippinen)
ISO 8859-2	**Latin-2**	für Albanisch, Tschechisch, Deutsch, Ungarisch, Polnisch, Rumänisch, Kroatisch, Slowakisch, Slowenisch, Schwedisch
ISO 8859-3	**Latin-3**	für Afrikaans, Katalanisch, Französisch, Galizisch, Italienisch, Maltesisch, Türkisch
ISO 8859-4	**Latin-4**	für Dänisch, Estnisch, Finnisch, Deutsch, Grönländisch, Lappisch, Litauisch, Norwegisch, Schwedisch. Jetzt ersetzt durch ISO 8859-10: Latin-6
ISO 8859-5	**Latin/Cyrillic**	Bulgarisch, Belorussisch, Mazedonisch, Russisch, Serbisch, Ukrainisch
ISO 8859-6	**Latin/Arabic**	Arabisch ohne Punktation
ISO 8859-7	**Latin/Greek**	Griechisch
ISO 8859-8	**Latin/Hebrew**	Hebräisch ohne Punktation
ISO 8859-9	**Latin-5**	Finnisch, Französisch, Deutsch, Irisch, Italienisch, Norwegisch, Portugiesisch, Spanisch, Schwedisch, Türkisch
ISO 8859-10	**Latin-6**	Dänisch, Estnisch, Faroerisch, Finnisch, Deutsch, Grönländisch (Inuit), Isländisch, Lappisch (Samisch), Litauisch, Lettisch, Norwegisch, Schwedisch
ISO 8859-14 [Entwurf]	**Latin-7**	Gälische Sprachen

Neun indische Hauptschriften (Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Kannada, Telugu, Malayalam) sind als **ISCII (Indian Standard Codes for Information Interchange)** in 8-Bit-Kode definiert (Indian Standard IS 13194:1991)..

Weiterführende Ressourcen zu ISO 8859:

- The **ISO 8859 Series** / Roman Czyborra -- URL: http://mirage.irdu.nus.sg/multilingual/unicode/misc/iso-8859.html. -- Zugriff am 16.6.1999. -- [Zeichensätze der ganzen ISO 8859 Serie]

12.4.3. UNICODE, ISO/IEC 10646, UCS, UTF

Um alle Zeichensätze der Welt, sowohl Alphabete, nichtalphabetische sprachbezogene Zeichen (z.B. Chinesisch) sowie Symbole, mathematische Operatoren, chemische Zeichen usw. zu kodieren, wurden zwei Standards entwickelt:

- **Unicode,** eine einheitliche **16- Bit**-(2 Byte)-Kodierung: erlaubt die Kodierung von 65536 unterschiedlichen Zeichen
- **ISO 10646 -- Information Technology -- Universal Multiple-Octet Coded Character Set (UCS)**, eine Kodierung, die bis zu **32 Bit** (4 Byte) lang ist: erlaubt die Kodierung von über 2 Milliarden (256^4) (!) Zeichen

Die historische Abfolge ist:

- 1991: Unicode, Version 1.0: Grundlage für Entwurf 2 zu ISO 10646
- 1993: ISO 10646 und Unicode, Version 1.1. Beide stimmen überein
- 1996: Unicode, Version 2.0, stimmt überein mit: ISO 10646 + Verbesserungen
- 1998 Unicode, Version 2.1

ISO 10646 besteht aus zwei miteinander kompatiblen Kodierungsschemata:

	Beispiel "T" (ASCII 54)	
	binär kodiert	**hexadezimal kodiert**
UCS-2: Kodierung mit **2 Byte** (16 Bit)	0000 0000 0101 0100	0054 = 54
UCS-4: Kodierung mit **4 Byte** (32 Bit)	0000 0000 0000 0000 0000 0000 0101 0100	0000 0054 = 54

ISO 10646 ist kompatibel mit ISO 646 (US-ASCII) und ISO 8859 (Latin-1), d.h. die ersten 128 Zeichen haben (mit Ausnahme der führenden Nullen) in UCS dieselben Werte wie in US-ASCII., die ersten 256 Zeichen dieselben Werte wie ISO 8859 (Latin-1).

Die Abwärtskompatibilität zeigt folgende Tabelle für die Kodierung von "A":

Standard	**Bits**	**Binärkodierung**	**Hexadezimal-Kodierung**
ISO 646 (US-ASCII)	7	100 0001	41
ISO 8859-1 (Latin-1)	8	0100 0001	41
UCS-2, Unicode	16	0000 0000 0100 0001	41

UCS-4	32	0000 0000 0000 0000 0000 0000 0100 0001	41

Den Bytegruppen in UCS wurden folgende Namen zugeordnet:

0000 0000 (4. Byte)	0000 0000 (3. Byte)	0000 0000 (2. Byte)	0000 0000 (1. Byte)
group	plane	row	cell

Da man eine UCS-2-Kodierung in UCS-4 als 00 00 xx xx darstellen kann, enthält UCS-2 alle Zeichen von group 00, plane 00 von UCS-4. Die plane von UCS-2 nennt man **Basic Multilingual Plane (BMP)**. Es gibt noch keine Zeichensätze, die außerhalb von BMP kodiert sind.

Unicode und UCS sind voll kompatibel, d.h. jede Unicode-Kodierung ist identisch mit der entsprechenden UCS-2-Kodierung.

Die UCS-Kodierung kann, wenn sie viele Nullen enthält, sehr speicherfressend sein, deshalb entwickelte man **UTF-8** ***(UCS Transformation Format, 8-bit form)***: Wenn der Hauptteil des Textes aus mit hohen Zahlen kodierten Zeichen besteht ist UTF-8 speicherfressender als UCS, bei niedrig kodierten Zeichen erspart es bis zu 25% Speicherplatz. Andere UTFs haben die unten angegebenen Funktionen.

UTF-8 ***(UCS Transformation Format, 8-bit form)*** **(ISO/IEC 10646 AM1):** benutzt 1 Byte für ASCII-Zeichen und 2 bis 6 Byte für die übrigen Zeichen	Beispiel "T" (ASCII 54): UCS-4: 0000 0000 0000 0000 0000 0000 0101 0100 = Hex 00000054 = 54 UTF-8: 0101 0100 (d.h. alle führenden Nullen werden weggelassen) = Hex 54
UTF-7 ***(UCS Transformation Format, 7-bit form)*****:** Unicode-Kodierung für Anwendungen, die das 8. Datenbit übergehen, z.B. e-mail nach dem SMTP (Simple Mail Transfer Protokol) des Internet (s. RFC 1642 : UTF-7 : A Mail-Safe Transformation Format of Unicode. -- URL: http://www.cis.ohio-state.edu/htbin/rfc/rfc1642.html. -- Zugriff am 14.6.1999). In MIME setzt man den character set identifier auf "UNICODE-1-1-UTF-7".	
UTF-16 ***(UCS Transformation Format for Planes of Group 00)***: erlaubt es, unter UNICODE (UCS-2) Zeichen aus UCS-4 darzustellen (genau von **16** zusätzlichen planes); dies tut man, indem manche Werte reserviert sind, um auf höhere planes zu schalten.	

Alle XML-Implementierungen **müssen** UTF-8 und UTF-16 unterstützen.

ASCII, UCS, UNICODE und UTF sind miteinander voll kompatibel. Ihr gegenseitiges Verhältnis zeigt folgende Grafik (nicht maßstabgetreu!):

Den Zeichensätzen werden in UCS bzw. Unicode zusammenhängende Bereiche zugeordnet, deren Größe vom Umfang des Zeichensatzes abhängt. Je vier Bit werden als Hexadezimalzahl (0 bis F) kodiert und als Kennzeichnung für Unicode *U+* davor gesetzt. Bisher wurden ca. 39000 Zeichen kodiert, davon ca. 21000 chinesische Zeichen!

In XML werden Unicode-Zeichen folgendermaßen kodiert:

&#x[Unicode Hexadezimalkodierung ohne führende Nullen]**;**

Einige Beispiele, die das Prinzip klar machen sollen:

Lateinschrift Basic Latin: Bereich U+**00**00 bis U+**00**7F

Zeichen	Binärcode	Hexadezimalcode	XML Codierung	Darstellung in UNICODE-fähigem Browser
A	**0000 0000** 0100 0001	U+**00**41	A	A
B	**0000 0000** 0100 0010	U+**00**42	B	B
%	**0000 0000** 0010 0101	U+**00**25	%	%

Kyrillisch: Bereich U+**04**00 bis **04**FF

Zeichen	Hexadezimalcode	XML Codierung	Darstellung in UNICODE-fähigem Browser
а [a]	U+**04**30	а	а
б [b]	U+**04**31	б	б
в [v]	U+**04**32	в	в

Devanagari (Hindi und Sanskrit): Bereich U+**09**00 bis U+**09**7F

Zeichen	Hexadezimalcode	XML Codierung
अ [a]	U+**09**05	अ
आ [â]	U+**09**06	आ

इ [i]	U+**09**07	इ

Thai: Bereich U+**0E**00 bis U+**0E**7F

Zeichen	Hexadezimalcode	XML Codierung	Darstellung in UNICODE-fähigem Browser
ง [ng]	U+**0E**07	ง	•
จ [c]	U+**0E**08	จ	•
ฉ [ch]	U+**0E**09	ฉ	•

Chinesisch: CJK Ideographs: Bereich U+**4E00** bis U+**9FA5**

Zeichen	Hexadezimalcode	XML Codierung	Darstellung in UNICODE-fähigem Browser
丈 [zhang = ein best. Längenmaß usw.]	U+**4E**08	丈	丈
亃 [lin]	U+**4E**83	亃	亃
犬 [quan = Hund]]	U+**72**AC	犬	犬
魩 [wen = Fisch]	U+**9B**69	魩	魩

Etwas für Pharmakologen und Botaniker (nur mit UNICODE-fähigem Browser lesbar): 药 茶 莓 茉 莉 茄 菜 草

Die wichtigste und die schwierigste Aufgabe der Kodierung sind die chinesisch-japanisch-koreanisch-[klassisch-vietnamesischen] Ideographe. Einige wichtige Punkte in der Geschichte dieser Kodierung:

- 1980: Chinese Character Code for Information Exchange (**CCCII**), in Taiwan entwickelt, kommt in Gebrauch
- 1981: Takahashi Tokutaro, Japan's National Diet Library, schlägt einen Standard für alle ostasiatischen Länder vor
- 1989 CCII wird für bibliographische Zwecke in den USA als ANSI Z39.64 angepasst und normiert: East Asian Character Code (**EACC**)
- 1986: bei Xerox beginnt man eine Han character cross-reference database
- 1988 RLIN (Research Libraries Information Network) beginnt CJK (Chinese-Japanese-Korean) Thesaurus: dient dem Unterhalt von EACC
- 1989: die Xerox- und die RLIN-database werden verschmolzen, dies führt zum ersten Entwurf des Unicode

character set, das im selben Jahr der ISO zum Einschluss in ISO 10646 vorgeschlagen wird
- 1990: Gründung einer Arbeitsgruppe mit Mitgliedern aus allen ostasiatischen Ländern: CJK-JRG (Chinese/ Japanese/Korean Joint Research Group)
- 1990: China kündet an, dass es vor Abschluss eines eigenen Entwurfs für ein Han character set (**GB 13000**) steht. Das Unicode Consortium und das Center for Computer and Information Development Research, die Computer-Standard-Organisation der Volksrepublik China beschließen ihre beiden Vorschläge zu einem einzigen Vorschlag zu vereinen
- 1992: Die CJK-JRG legt *Unified Repertoire and Ordering (**URO) 2.0*** vor. URO wird sowohl Bestandteil von Unicode 1.0 als auch von ISO 10646
- 1993: CJK-JRG wird subgroup von ISO und wird umbenannt in Ideographic Rapporteur Group (IRG). Die IRG ist verantwortlich für die Weiterentwicklung von URO 2.0
- 1994: die IRG beschließt, auch klassische vietnamesische Ideographe aufzunehmen
- 1995: es liegen über 21000 Ideographe vor, deren zusätzliche Aufnahme in URO 2.0 in Erwägung gezogen wird. Für die endgültige Code-Zuweisung ist nicht die IRG zuständig, sondern ISO SC2/WG2

Mathematische Operatoren: U+**22**00 to U+**22**FF

Zeichen	Hexadezimalcode	XML Codierung
∛	U+**22**1B	&#x221B;
∮	U+**22**2E	&#x222E;
∩	U+**22**C2	&#x22C2;

Die gegenwärtig gültige Version des Unicode-Standards (2.1) unterstützt offiziell die im Folgenden genannten Schriften und Zeichensätze. Die Reihenfolge der Aufzählung entspricht der Abfolge in der Zuteilung der 16-Bit-Codes. Die Links verweisen auf Zeichentafeln mit der Kodierung der einzelnen Zeichen. (Zugriff auf alle Links am 11.6.1999).

- Steuerzeichen und Lateinschrift
  - C0 Controls and Basic Latin
  - C1 Controls and Latin-1 Supplement
  - Latin Extended-A
  - Latin Extended-B
- Internationale Lautschrift
  - IPA (International Phonetic Association] Extensions
- Spacing Modifier Letters
- Combining Diacritical Marks
- Greek
- Cyrillic
- Armenian
- Hebrew
- Arabic
- Indische Schriften
  - Devanagari
  - Bengali
  - Gurmukhi
  - Gujarati
  - Oriya

    - Tamil
    - Telugu
    - Kannada
    - Malayalam
- Thaischriften:
    - Thai
    - Lao
- Tibetan
- Georgian
- Hangul Jamo
- Latin Extended Additional
- Greek Extended
- Sonderzeichen, Symbole, Graphisches:
    - General Punctuation
    - Superscripts and Subscripts
    - Currency Symbols
    - Combining Diacritical Marks for Symbols
    - Letterlike Symbols
    - Number Forms
    - Arrows
    - Mathematical Operators
    - Miscellaneous Technical
    - Control Pictures
    - Optical Character Recognition
    - Enclosed Alphanumerics
    - Box Drawing
    - Block Elements
    - Geometric Shapes
    - Miscellaneous Symbols
    - Dingbats
- CJK -- Chinesisch, Japanisch, Koreanisch:
    - CJK Symbols and Punctuation
    - Hiragana
    - Katakana
    - Bopomofo
    - Hangul Compatibility Jamo
    - Kanbun
    - Enclosed CJK Letters and Months
    - CJK Compatibility
    - CJK Ideographs
    - Hangul Syllables
    - High Surrogates
    - High Private Use Surrogates
    - Low Surrogates
    - Private Use Area
    - CJK Compatibility Ideographs
- Typographisches und Kalligraphisches:
    - Alphabetic Presentation Forms

    - Arabic Presentation Forms-A
    - Combining Half Marks
    - CJK Compatibility Forms
    - Small Form Variants
    - Arabic Presentation Forms-B
    - Halfwidth and Fullwidth Forms
- Specials

Weiterführende Ressourcen zu UNICODE:

Organisationen:

Unicode Consortium. -- URL: http://www.unicode.org.. -- Zugriff am 11.6.1999 -- [**Die** Informationsquelle; sehr ergiebig!]

Ressourcen in Printform:

The **Unicode Standard Version 2.0** / The Unicoder Consortium. -- Reading [u.a.] : Addison-Wesley, ©1996. -- getrennte Zählung [960 S.] : Ill. + 1 CD-ROM. -- ISBN 0201483459. -- [Unerlässlich für jede Implementierung von UNICODE]. -- {Wenn Sie HIER klicken, können Sie dieses Buch bei **amazon.de** bestellen}

12.4.4. PostScript

PostScript ist eine von Adobe entwickelte und 1985 vorgestellte Layoutdefinitions-Sprache, die von vielen Softwarepaketen und Betriebssystemen unterstützt wird. PostScript erlaubt, kompliziert formatierte Dokumente mit unterschiedlichen Schriften, Schriftarten, Grafiken, Farben usw. zu definieren. PostScript macht die Files aber sehr groß (z.B. 100 Druckseiten u.U. = 40 Mb (!) PostScript). Auch gibt es durchaus lästige Inkompatibilitäten zwischen PostScript-Files, die mit verschiedenen Programmen erstellt wurden. PostScript wird vor allem zur Steuerung von Druckern und Fotosatzgeräten benutzt. Softwarepakete, die PostScript zur Druckerausgabe unterstützen, ermöglichen trotzdem oft nicht eine Bildschirmwiedergabe von PostScript-Dateien. Die Umsetzung einer Postscript-Datei zur Druckausgabe erfolgt innerhalb des Druckers durch einen speziellen Postscript-Interpreter. Mit *Display PostScript* besteht die Möglichkeit, auch die Bildschirmausgabe durch PostScript zu steuern.

Weiterführende Ressourcen zu Postscript:

Yahoo Categories:

- http://www.yahoo.com/Computers_and_Internet/Programming_Languages/**PostScript/**. -- Zugriff am 16.6.1999

12.4.5. Adobe Acrobat

Acrobat® ist eine von Adobe 1993 vorgestellte Software zum Datenaustausch. Adobe Acrobat hat ähnliche Features wie PostScript, Acrobat-Files sind aber weniger umfangreich als PostScript-Files. Lange Zeit war der Acrobat Reader nicht kostenlos erhältlich, was der Verbreitung von Adobe Acrobat sehr im Wege stand. Jetzt ist der Adobe Acrobat Reader Freeware und Files im Acrobat-Format .pdf (Portable Document Format) werden auch immer häufiger als Web-Seiten angeboten. Da pdf-Files viel umfangreicher und umständlicher in der Handhabung sind als html-Files, scheint mir die Publikation im WWW mittels von pdf-Files nur in den Fällen berechtigt, in denen ein unveränderliches Layout zwingend zum angebotenen Inhalt gehört (sonst gehört das in die Kategorie: "Protzige Form soll mangelhaften Inhalt verbergen").

Weiterführende Ressourcen Adobe Acrobat:

Yahoo Categories:

- http://www.yahoo.com/Business_and_Economy/Companies/Computers/Software/Graphics/Adobe_Systems__Inc_/Products_and_Services/Products/**Acrobat/**. -- Zugriff am 16.6.1999

Zu Kapitel 12, Teil 2

UTF-8 and Unicode FAQ for Unix/Linux

by Markus Kuhn

This text is a very comprehensive one-stop information resource on how you can use Unicode/UTF-8 on POSIX systems (Linux, Unix). You will find here both introductory information for every user, as well as detailed references for the experienced developer.

Unicode is well on the way to replace ASCII, ISO 8859 and EUC at all levels. It allows you to handle not only text in practically any script and language used on this planet, it also provides you with a comprehensive set of mathematical and technical symbols to simplify scientific information exchange.

With the UTF-8 encoding, Unicode can be used in a convenient and backwards compatible way in environments that, like Unix, were designed entirely around ASCII. UTF-8 is the way in which Unicode is used under Unix, Linux, and similar systems. It is now time to make sure that you are well familiar with it and that your software supports UTF-8 smoothly.

Contents

What are UCS and ISO 10646?

The international standard **ISO 10646** defines the **Universal Character Set (UCS)**. UCS is a superset of all other character set standards. It guarantees round-trip compatibility to other character sets. No information will be lost if you convert any text string to UCS and then back to the original encoding.

UCS contains the characters required to represent practically all known languages. This includes not only the Latin, Greek, Cyrillic, Hebrew, Arabic, Armenian, and Georgian scripts, but also Chinese, Japanese and Korean Han ideographs as well as scripts such as Hiragana, Katakana, Hangul, Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Thai, Lao, Khmer, Bopomofo, Tibetian, Runic, Ethiopic, Canadian Syllabics, Cherokee, Mongolian, Ogham, Myanmar, Sinhala, Thaana, Yi, and others. For scripts not yet covered, research on how to best encode them for computer usage is still going on and they will be added eventually. This includes not only Cuneiform, Hieroglyphs and various Indo-European languages, but even some selected artistic scripts such as Tolkien's Tengwar and Cirth. UCS also covers a large number of graphical, typographical, mathematical and scientific symbols, including those provided by TeX, PostScript, APL, the International Phonetic Alphabet (IPA), MS-DOS, MS-Windows, Macintosh, OCR fonts, as well as many word processing and publishing systems, and more are being added.

ISO 10646 defines formally a 31-bit character set. The most commonly used characters, including all those found in older encoding standards, have been placed in one of the first 65534 positions (0x0000 to 0xFFFD). This 16-bit subset of UCS is called the **Basic Multilingual Plane (BMP)** or Plane 0. The characters that were later added outside the 16-bit BMP are mostly for specialist applications such as historic scripts and scientific notation. Current plans are that there will never be characters assigned outside the 21-bit code space from 0x000000 to 0x10FFFF, which covers a bit over one million potential future characters. The ISO 10646-1 standard was first published in 1993 and defines the architecture of the character set and the content of the BMP. A second part ISO 10646-2 was added in 2001 and defines characters encoded outside the BMP. New characters are still being added on a continuous basis, but the existing characters will not be changed any more and are stable.

UCS assigns to each character not only a code number but also an official name. A hexadecimal number that represents a UCS or Unicode value is commonly preceded by "U+" as in U+0041 for the character "Latin capital letter A". The UCS characters U+0000 to U+007F are identical to those in US-ASCII (ISO

646 IRV) and the range U+0000 to U+00FF is identical to ISO 8859-1 (Latin-1). The range U+E000 to U+F8FF and also larger ranges outside the BMP are reserved for private use. UCS also defines several methods for encoding a string of characters as a sequence of bytes, such as UTF-8 and UTF-16.

The full references for the two parts of the UCS standard are

- International Standard ISO/IEC 10646-1, Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane. Second edition, International Organization for Standardization, Geneva, 2000.
- International Standard ISO/IEC 10646-2, Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 2: Supplementary Planes. First edition, International Organization for Standardization, Geneva, 2001.

The standards can be ordered online from ISO as a set of PDF files on CD-ROM for 83 CHF (~54 EUR, ~63 USD, ~37 GBP) each.

What are combining characters?

Some code points in UCS have been assigned to **combining characters**. These are similar to the non-spacing accent keys on a typewriter. A combining character is not a full character by itself. It is an accent or other diacritical mark that is added to the previous character. This way, it is possible to place any accent on any character. The most important accented characters, like those used in the orthographies of common languages, have codes of their own in UCS to ensure backwards compatibility with older character sets. They are known as **precomposed characters**. Precomposed characters are available in UCS for backwards compatibility with older encodings that have no combining characters, such as ISO 8859. The combining-character mechanism allows one to add accents and other diacritical marks to any character. This is especially important for scientific notations such as mathematical formulae and the International Phonetic Alphabet, where any possible combination of a base character and one or several diacritical marks could be needed.

Combining characters follow the character which they modify. For example, the German umlaut character Ä ("Latin capital letter A with diaeresis") can either be represented by the precomposed UCS code U+00C4, or alternatively by the combination of a normal "Latin capital letter A" followed by a "combining diaeresis": U+0041 U+0308. Several combining characters can be applied when it is necessary to stack multiple accents or add combining marks both above and below the base character. The Thai script, for example, needs up to two combining characters on a single base character.

What are UCS implementation levels?

Not all systems can be expected to support all the advanced mechanisms of UCS, such as combining characters. Therefore, ISO 10646 specifies the following three implementation levels:

Level 1
: Combining characters and Hangul Jamo characters are not supported.
[Hangul Jamo are an alternative representation of precomposed modern Hangul syllables as a sequence of consonants and vowels. They are required to fully support the Korean script including Middle Korean.]

Level 2
: Like level 1, however in some scripts, a fixed list of combining characters is now allowed (e.g., for Hebrew, Arabic, Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugo, Kannada, Malayalam, Thai and Lao). These scripts cannot be represented adequately in UCS without support for at least certain combining characters.

Level 3
: All UCS characters are supported, such that, for example, mathematicians can place a tilde or an arrow (or both) on any character.

Has UCS been adopted as a national standard?

Yes, a number of countries have published national adoptions of ISO 10646, sometimes after adding additional annexes with cross-references to older national standards, implementation guidelines, and specifications of various national implementation subsets:

- China: GB 13000.1-93
- Japan: JIS X 0221-1:2001
- Korea: KS X 1005-1:1995 (includes ISO 10646-1:1993 amendments 1-7)
- Vietnam: TCVN 6909:2001
  (This "16-bit Coded Vietnamese Character Set" is a small UCS subset and to be implemented for data interchange with and within government agencies as of 2002-07-01.)
- Iran: ISIRI 6219:2002, Information Technology — Persian Information Interchange and Display Mechanism, using Unicode. (This is not a version or subset of ISO 10646, but a separate document that provides additional national guidance and clarification on handling the Persian language and the Arabic script in Unicode.)

What is Unicode?

In the late 1980s, there have been two independent attempts to create a single unified character set. One was the ISO 10646 project of the International Organization for Standardization (ISO), the other was the Unicode Project organized by a consortium of (initially mostly US) manufacturers of multi-lingual software. Fortunately, the participants of both projects realized in around 1991 that two different unified character sets is not exactly what the world needs. They joined their efforts and worked together on creating a single code table. Both projects still exist and publish their respective standards independently, however the Unicode Consortium and ISO/IEC JTC1/SC2 have agreed to keep the code tables of the Unicode and ISO 10646 standards compatible and they closely coordinate any further extensions. Unicode 1.1 corresponded to ISO 10646-1:1993, Unicode 3.0 corresponded to ISO 10646-1:2000, Unicode 3.2 added ISO 10646-2:2001, and Unicode 4.0 corresponds to the forthcoming third version of ISO 10646. All Unicode versions since 2.0 are compatible, only new characters will be added, no existing characters will be removed or renamed in the future.

The Unicode Standard can be ordered like any normal book, for instance via amazon.com for around 75 USD:

The Unicode Consortium: The Unicode Standard, Version 4.0,

Addison-Wesley, 2003,
ISBN 0-321-18578-1.

If you work frequently with text processing and character sets, you definitely should get a copy. Unicode 4.0 is also available online.

So what is the difference between Unicode and ISO 10646?

The Unicode Standard published by the Unicode Consortium corresponds to ISO 10646 at implementation level 3. All characters are at the same positions and have the same names in both standards.

The Unicode Standard defines in addition much more semantics associated with some of the characters and is in general a better reference for implementors of high-quality typographic publishing systems. Unicode specifies algorithms for rendering presentation forms of some scripts (say Arabic), handling of bi-directional texts that mix for instance Latin and Hebrew, algorithms for sorting and string comparison, and much more.

The ISO 10646 standard on the other hand is not much more than a simple character set table, comparable to the old ISO 8859 standards. It specifies some terminology related to the standard, defines some encoding alternatives, and it contains specifications of how to use UCS in connection with other established ISO standards such as ISO 6429 and ISO 2022. There are other closely related ISO standards, for instance ISO 14651 on sorting UCS strings. A nice feature of the ISO 10646-1 standard is that it provides CJK example glyphs in five different style variants, while the Unicode standard shows the CJK ideographs only in a Chinese variant.

What is UTF-8?

UCS and Unicode are first of all just code tables that assign integer numbers to characters. There exist several alternatives for how a sequence of such characters or their respective integer values can be represented as a sequence of bytes. The two most obvious encodings store Unicode text as sequences of either 2 or 4 bytes sequences. The official terms for these encodings are UCS-2 and UCS-4, respectively. Unless otherwise specified, the most significant byte comes first in these (Bigendian convention). An ASCII or Latin-1 file can be transformed into a UCS-2 file by simply inserting a 0x00 byte in front of every ASCII byte. If we want to have a UCS-4 file, we have to insert three 0x00 bytes instead before every ASCII byte.

Using UCS-2 (or UCS-4) under Unix would lead to very severe problems. Strings with these encodings can contain as parts of many wide characters bytes like '\0' or '/' which have a special meaning in filenames and other C library function parameters. In addition, the majority of UNIX tools expects ASCII files and can't read 16-bit words as characters without major modifications. For these reasons, **UCS-2** is not a suitable external encoding of **Unicode** in filenames, text files, environment variables, etc.

The **UTF-8** encoding defined in ISO 10646-1:2000 Annex D and also described in RFC 3629 as well as section 3.9 of the Unicode 4.0 standard does not have these problems. It is clearly the way to go for using **Unicode** under Unix-style operating systems.

UTF-8 has the following properties:

- UCS characters U+0000 to U+007F (ASCII) are encoded simply as bytes 0x00 to 0x7F (ASCII compatibility). This means that files and strings which contain only 7-bit ASCII characters have the same encoding under both ASCII and UTF-8.
- All UCS characters >U+007F are encoded as a sequence of several bytes, each of which has the most significant bit set. Therefore, no ASCII byte (0x00-0x7F) can appear as part of any other character.
- The first byte of a multibyte sequence that represents a non-ASCII character is always in the range 0xC0 to 0xFD and it indicates how many bytes follow for this character. All further bytes in a multibyte sequence are in the range 0x80 to 0xBF. This allows easy resynchronization and makes the encoding stateless and robust against missing bytes.
- All possible 2^{31} UCS codes can be encoded.
- UTF-8 encoded characters may theoretically be up to six bytes long, however 16-bit BMP characters are only up to three bytes long.
- The sorting order of Bigendian UCS-4 byte strings is preserved.
- The bytes 0xFE and 0xFF are never used in the UTF-8 encoding.

The following byte sequences are used to represent a character. The sequence to be used depends on the Unicode number of the character:

U-00000000 - U-0000007F:	0*xxxxxxx*
U-00000080 - U-000007FF:	110*xxxxx* 10*xxxxxx*
U-00000800 - U-0000FFFF:	1110*xxxx* 10*xxxxxx* 10*xxxxxx*
U-00010000 - U-001FFFFF:	11110*xxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*
U-00200000 - U-03FFFFFF:	111110*xx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*
U-04000000 - U-7FFFFFFF:	1111110*x* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*

The *xxx* bit positions are filled with the bits of the character code number in binary representation. The rightmost *x* bit is the least-significant bit. Only the shortest possible multibyte sequence which can represent the code number of the character can be used. Note that in multibyte sequences, the number of leading 1 bits in the first byte is identical to the number of bytes in the entire sequence.

Examples: The Unicode character `U+00A9 = 1010 1001` (copyright sign) is encoded in UTF-8 as

```
11000010 10101001 = 0xC2 0xA9
```

and character `U+2260 = 0010 0010 0110 0000` (not equal to) is encoded as:

```
11100010 10001001 10100000 = 0xE2 0x89 0xA0
```

The official name and spelling of this encoding is UTF-8, where UTF stands for **U**CS **T**ransformation **F**ormat. Please do not write UTF-8 in any documentation text in other ways (such as utf8 or UTF_8), unless of course you refer to a variable name and not the encoding itself.

An important note for developers of UTF-8 decoding routines: For security reasons, a UTF-8 decoder must not accept UTF-8 sequences that are longer than necessary to encode a character. For example, the character U+000A (line feed) must be accepted from a UTF-8 stream **only** in the form 0x0A, but not in any of the following five possible overlong forms:

```
0xC0 0x8A
0xE0 0x80 0x8A
0xF0 0x80 0x80 0x8A
0xF8 0x80 0x80 0x80 0x8A
0xFC 0x80 0x80 0x80 0x80 0x8A
```

Any overlong UTF-8 sequence could be abused to bypass UTF-8 substring tests that look only for the shortest possible encoding. All overlong UTF-8 sequences start with one of the following byte patterns:

1100000*x* (10*xxxxxx*)
11100000 100*xxxxx* (10*xxxxxx*)
11110000 1000*xxxx* (10*xxxxxx* 10*xxxxxx*)
11111000 10000*xxx* (10*xxxxxx* 10*xxxxxx* 10*xxxxxx*)
11111100 100000*xx* (10*xxxxxx* 10*xxxxxx* 10*xxxxxx* 10*xxxxxx*)

Also note that the code positions U+D800 to U+DFFF (UTF-16 surrogates) as well as U+FFFE and U+FFFF must not occur in normal UTF-8 or UCS-4 data. UTF-8 decoders should treat them like malformed or overlong sequences for safety reasons.

Markus Kuhn's UTF-8 decoder stress test file contains a systematic collection of malformed and overlong UTF-8 sequences and will help you to verify the robustness of your decoder.

Who invented UTF-8?

The encoding known today as UTF-8 was invented by Ken Thompson. It was born during the evening hours of 1992-09-02 in a New Jersey diner, where he designed it in the presence of Rob Pike on a placemat (see Rob Pike's UTF-8 history). It replaced an earlier attempt to design a FSS/UTF (file system safe UCS transformation format) that was circulated in an X/Open working document in August 1992 by Gary Miller (IBM), Greger Leijonhufvud and John Entenmann (SMI) as a replacement for the division-heavy UTF-1 encoding from the first edition of ISO 10646-1. By the end of the first week of September 1992, Pike and Thompson had turned AT&T Bell Lab's Plan9 into the world's first operating system to

use UTF-8. They reported about their experience at the USENIX Winter 1993 Technical Conference, San Diego, January 25-29, 1993, Proceedings, pp. 43-50. FSS/UTF was briefly also referred to as UTF-2 and later renamed into UTF-8, and pushed through the standards process by the X/Open Joint Internationalization Group XOJIG.

Where do I find nice UTF-8 example files?

A few interesting UTF-8 example files for tests and demonstrations are:

- UTF-8 Sampler web page by the Kermit project
- Markus Kuhn's example plain-text files, including among others the classic demo, decoder test, TeX repertoire, WGL4 repertoire, euro test pages, and Robert Brady's IPA lyrics.
- Unicode Transcriptions

What different encodings are there?

Both the UCS and Unicode standards are first of all large tables that assign to every character an integer number. If you use the term "UCS", "ISO 10646", or "Unicode", this just refers to a mapping between characters and integers. This does not yet specify how to store these integers as a sequence of bytes in memory.

ISO 10646-1 defines the UCS-2 and UCS-4 encodings. These are sequences of 2 bytes and 4 bytes per character, respectively. ISO 10646 was from the beginning designed as a 31-bit character set (with possible code positions ranging from U-00000000 to U-7FFFFFFF), however it took until 2001 for the first characters to be assigned beyond the Basic Multilingual Plane (BMP), that is beyond the first 2^{16} character positions (see ISO 10646-2 and Unicode 3.1). UCS-4 can represent all UCS and Unicode characters, UCS-2 can represent only those from the BMP (U+0000 to U+FFFF).

"Unicode" originally implied that the encoding was UCS-2 and it initially didn't make any provisions for characters outside the BMP (U+0000 to U+FFFF). When it became clear that more than 64k characters would be needed for certain special applications (historic alphabets and ideographs, mathematical and musical typesetting, etc.), Unicode was turned into a sort of 21-bit character set with possible code points in the range U-00000000 to U-0010FFFF. The 2×1024 surrogate characters (U+D800 to U+DFFF) were introduced into the BMP to allow 1024×1024 non-BMP characters to be represented as a sequence of two 16-bit surrogate characters. This way UTF-16 was born, which represents the extended "21-bit" Unicode in a way backwards compatible with UCS-2. The term UTF-32 was introduced in Unicode to describe a 4-byte encoding of the extended "21-bit" Unicode. UTF-32 is the exact same thing as UCS-4, except that by definition UTF-32 is never used to represent characters above U-0010FFFF, while UCS-4 can cover all 2^{31} code positions up to U-7FFFFFFF. The ISO 10646 working group has agreed to modify their standard to exclude code positions beyond U-0010FFFF, in order to turn the new UCS-4 and UTF-32 into practically the same thing.

In addition to all that, UTF-8 was introduced to provide an ASCII backwards compatible multi-byte encoding. The definitions of UTF-8 in UCS and Unicode differed originally slightly, because in UCS, up to 6-byte long UTF-8 sequences were possible to represent characters up to U-7FFFFFFF, while in

Unicode only up to 4-byte long UTF-8 sequences are defined to represent characters up to U-0010FFFF. (The difference was in essence the same as between UCS-4 and UTF-32.)

No endianess is implied by the encoding names UCS-2, UCS-4, UTF-16, and UTF-32, though ISO 10646-1 says that Bigendian should be preferred unless otherwise agreed. It has become customary to append the letters "BE" (Bigendian, high-byte first) and "LE" (Littleendian, low-byte first) to the encoding names in order to explicitly specify a byte order.

In order to allow the automatic detection of the byte order, it has become customary on some platforms (notably Win32) to start every Unicode file with the character U+FEFF (ZERO WIDTH NO-BREAK SPACE), also known as the Byte-Order Mark (BOM). Its byte-swapped equivalent U+FFFE is not a valid Unicode character, therefore it helps to unambiguously distinguish the Bigendian and Littleendian variants of UTF-16 and UTF-32.

A full featured character encoding converter will have to provide the following 13 encoding variants of Unicode and UCS:

> UCS-2, UCS-2BE, UCS-2LE, UCS-4, UCS-4LE, UCS-4BE, UTF-8, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE

Where no byte order is explicitly specified, use the byte order of the CPU on which the conversion takes place and in an input stream swap the byte order whenever U+FFFE is encountered. The difference between outputting UCS-4 versus UTF-32 and UTF-16 versus UCS-2 lies in handling out-of-range characters. The fallback mechanism for non-representable characters has to be activated in UTF-32 (for characters > U-0010FFFF) or UCS-2 (for characters > U+FFFF) even where UCS-4 or UTF-16 respectively would offer a representation.

Really just of historic interest are UTF-1, UTF-7, SCSU and a dozen other less widely publicised UCS encoding proposals with various properties, none of which ever enjoyed any significant use. Their use should be avoided.

A good encoding converter will also offer options for adding or removing the BOM:

- Unconditionally prefix the output text with U+FEFF.
- Prefix the output text with U+FEFF unless it is already there.
- Remove the first character if it is U+FEFF.

It has also been suggested to use the UTF-8 encoded BOM (0xEF 0xBB 0xBF) as a signature to mark the beginning of a UTF-8 file. This practice should definitely **not** be used on POSIX systems for several reasons:

- On POSIX systems, the locale and not magic file type codes define the encoding of plain text files. Mixing the two concepts would add a lot of complexity and break existing functionality.
- Adding a UTF-8 signature at the start of a file would interfere with many established conventions such as the kernel looking for "#!" at the beginning of a plaintext executable to locate the appropriate interpreter.

- Handling BOMs properly would add undesirable complexity even to simple programs like `cat` or `grep` that mix contents of several files into one.

In addition to the encoding alternatives, Unicode also specifies various Normalization Forms, which provide reasonable subsets of Unicode, especially to remove encoding ambiguities caused by the presence of precomposed and compatibility characters:

- **Normalization Form D (NFD):** Split up (decompose) precomposed characters into combining sequences where possible, e.g. use U+0041 U+0308 (LATIN CAPITAL LETTER A, COMBINING DIAERESIS) instead of U+00C4 (LATIN CAPITAL LETTER A WITH DIAERESIS). Also avoid deprecated characters, e.g. use U+0041 U+030A (LATIN CAPITAL LETTER A, COMBINING RING ABOVE) instead of U+212B (ANGSTROM SIGN).
- **Normalization Form C (NFC):** Use precomposed characters instead of combining sequences where possible, e.g. use U+00C4 ("Latin capital letter A with diaeresis") instead of U+0041 U +0308 ("Latin capital letter A", "combining diaeresis"). Also avoid deprecated characters, e.g. use U+00C5 (LATIN CAPITAL LETTER A WITH RING ABOVE) instead of U+212B (ANGSTROM SIGN).
  NFC is the preferred form for Linux and WWW.
- **Normalization Form KD (NFKD):** Like NFD, but avoid in addition the use of compatibility characters, e.g. use "fi" instead of U+FB01 (LATIN SMALL LIGATURE FI).
- **Normalization Form KC (NFKC):** Like NFC, but avoid in addition the use of compatibility characters, e.g. use "fi" instead of U+FB01 (LATIN SMALL LIGATURE FI).

A full-featured character encoding converter should also offer conversion between normalization forms. Care should be used with mapping to NFKD or NFKC, as semantic information might be lost (for instance U+00B2 (SUPERSCRIPT TWO) maps to 2) and extra mark-up information might have to be added to preserve it (e.g., `<SUP>2</SUP>` in HTML).

What programming languages support Unicode?

More recent programming languages that were developed after around 1993 already have special data types for Unicode/ISO 10646-1 characters. This is the case with Ada95, Java, TCL, Perl, Python, C# and others.

ISO C 90 specifies mechanisms to handle multi-byte encoding and wide characters. These facilities were improved with Amendment 1 to ISO C 90 in 1994 and even further improvements were made in the ISO C 99 standard. These facilities were designed originally with various East-Asian encodings in mind. They are on one side slightly more sophisticated than what would be necessary to handle UCS (handling of "shift sequences"), but also lack support for more advanced aspects of UCS (combining characters, etc.). UTF-8 is an example of what the ISO C standard calls multi-byte encoding. The type *wchar_t*, which in modern environments is usually a signed 32-bit integer, can be used to hold Unicode characters.

Unfortunately, *wchar_t* was already widely used for various Asian 16-bit encodings throughout the 1990s. Therefore, the ISO C 99 standard was bound by backwards compatibility. It could not be changed to require *wchar_t* to be used with UCS, like Java and Ada95 managed to do. However, the C compiler can at least signal to an application that *wchar_t* is guaranteed to hold UCS values in all locales. To do

so, it defines the macro `__STDC_ISO_10646__` to be an integer constant of the form *yyyymm*L. The year and month refer to the version of ISO/IEC 10646 and its amendments that have been implemented. For example, `__STDC_ISO_10646__ == 200009L` if the implementation covers ISO/IEC 10646-1:2000.

How should Unicode be used under Linux?

Before UTF-8 emerged, Linux users all over the world had to use various different language-specific extensions of ASCII. Most popular were ISO 8859-1 and ISO 8859-2 in Europe, ISO 8859-7 in Greece, KOI-8 / ISO 8859-5 / CP1251 in Russia, EUC and Shift-JIS in Japan, BIG5 in Taiwan, etc. This made the exchange of files difficult and application software had to worry about various small differences between these encodings. Support for these encodings was usually incomplete, untested, and unsatisfactory, because the application developers rarely used all these encodings themselves.

Because of these difficulties, major Linux distributors and application developers have now started to phase out these older legacy encodings in favour of UTF-8. UTF-8 support has improved dramatically over the last few years and ever more people now use UTF-8 on a daily basis in

- text files (source code, HTML files, email messages, etc.)
- file names
- standard input and standard output, pipes
- environment variables
- cut and paste selection buffers
- telnet, modem, and serial port connections to terminal emulators

and in any other places where byte sequences used to be interpreted in ASCII.

In UTF-8 mode, terminal emulators such as xterm or the Linux console driver transform every keystroke into the corresponding UTF-8 sequence and send it to the stdin of the foreground process. Similarly, any output of a process on stdout is sent to the terminal emulator, where it is processed with a UTF-8 decoder and then displayed using a 16-bit font.

Full Unicode functionality with all bells and whistles (e.g. high-quality typesetting of the Arabic and Indic scripts) can only be expected from sophisticated multi-lingual word-processing packages. What Linux supports today on a broad base is far simpler and mainly aimed at replacing the old 8- and 16-bit character sets. Linux terminal emulators and command line tools usually only support a Level 1 implementation of ISO 10646-1 (no combining characters), and only scripts such as Latin, Greek, Cyrillic, Armenian, Georgian, CJK, and many scientific symbols are supported that need no further processing support. At this level, UCS support is very comparable to ISO 8859 support and the only significant difference is that we have now thousands of different characters available, that characters can be represented by multibyte sequences, and that ideographic Chinese/Japanese/Korean characters require two terminal character positions (double-width).

Level 2 support in the form of combining characters for selected scripts (in particular Thai) and Hangul Jamo is in parts also available (i.e., some fonts, terminal emulators and editors support it via simple overstringing), but precomposed characters should be preferred over combining character sequences

where available. More formally, the preferred way of encoding text in Unicode under Linux should be *Normalization Form C* as defined in Unicode Technical Report #15.

One influential non-POSIX PC operating system vendor (whom we shall leave unnamed here) suggested that all Unicode files should start with the character ZERO WIDTH NOBREAK SPACE (U+FEFF), which is in this role also referred to as the "signature" or "byte-order mark (BOM)", in order to identify the encoding and byte-order used in a file. Linux/Unix does **not** use any BOMs and signatures. They would break far too many existing ASCII syntax conventions (such as scripts starting with `#!`). On POSIX systems, the selected locale identifies already the encoding expected in all input and output files of a process. It has also been suggested to call UTF-8 files without a signature "UTF-8N" files, but this non-standard term is usually not used in the POSIX world.

Before you start experimenting with UTF-8 under Linux, update your installation to a recent distribution with up-to-date UTF-8 support. This is particular the case if you use an installation older than SuSE 8.1 or Red Hat 8.0. Before these, UTF-8 support was far too limited and experimental to be recommendable for daily use.

Red Hat Linux 8.0 was the first distribution to make UTF-8 even the default encoding for all locales other than Chinese/Japanese/Korean.

How do I have to modify my software?

If you are a developer, there are several approaches to add UTF-8 support. We can split them into two categories, which I will call soft and hard conversion. In soft conversion, data is kept in its UTF-8 form everywhere and only very few software changes are necessary. In hard conversion, any UTF-8 data that the program reads will be converted into wide-character arrays and will be handled as such everywhere inside the application. Strings will only be converted back to UTF-8 at output time. Internally, a character remains a fixed-size memory object.

We can also distinguish hard-wired and locale-dependent approaches for supporting UTF-8, depending on how much the string processing relies on the standard library. C offers a number of string processing functions designed to handle arbitrary locale-specific multibyte encodings. An application programmer who relies entirely on these can remain unaware of the actual details of the UTF-8 encoding. Chances are then that by merely changing the locale setting, several other multi-byte encodings (such as EUC) will automatically be supported as well. The other way a programmer can go is to hardcode knowledge about UTF-8 into the application. This may lead in some situations to significant performance improvements. It may be the best approach for applications that will only be used with ASCII and UTF-8.

Even where support for every multi-byte encoding supported by libc is desired, it may well be worth to add extra code optimized for UTF-8. Thanks to UTF-8's self-synchronizing features, it can be processed very efficiently. The locale-dependent libc string functions can be two orders of magnitude slower than equivalent hardwired UTF-8 procedures. A bad teaching example was GNU grep 2.5.1, which relied entirely on locale-dependent libc functions such as `mbrlen()` for its generic multi-byte encoding support. This made it about 100× slower in multibyte mode than in single-byte mode! Other applications with hardwired support for UTF-8 regular expressions (e.g., Perl 5.8) do not suffer this dramatic slowdown.

Most applications can do very fine with just soft conversion. This is what makes the introduction of UTF-8 on Unix feasible at all. To name two trivial examples, programs such as `cat` and `echo` do not have to be modified at all. They can remain completely ignorant as to whether their input and output is ISO 8859-2 or UTF-8, because they handle just byte streams without processing them. They only recognize ASCII characters and control codes such as `'\n'` which do not change in any way under UTF-8. Therefore the UTF-8 encoding and decoding is done for these applications completely in the terminal emulator.

A small modification will be necessary for any program that determines the number of characters in a string by counting the bytes. With UTF-8, as with other multi-byte encodings, where the length of a text string is of concern, programmers have to distinguish clearly between

1. the number of bytes,
2. the number of characters,
3. the display width (e.g., the number of cursor position cells in a VT100 terminal emulator)

of a string.

C's `strlen(s)` function always counts the *number of bytes*. This is the number relevant, for example, for memory management (determination of string buffer sizes). Where the output of strlen is used for such purposes, no change will be necessary.

The *number of characters* can be counted in C in a portable way using `mbstowcs(NULL,s,0)`. This works for UTF-8 like for any other supported encoding, as long as the appropriate locale has been selected. A hard-wired technique to count the number of characters in a UTF-8 string is to count all bytes except those in the range 0x80 - 0xBF, because these are just continuation bytes and not characters of their own. However, the need to count characters arises surprisingly rarely in applications.

In applications written for ASCII or ISO 8859, a far more common use of `strlen` is to predict the *number of columns* that the cursor of the terminal will advance if a string is printed. With UTF-8, neither a byte nor a character count will predict the display width, because ideographic characters (Chinese, Japanese, Korean) will occupy two column positions, whereas control and combining characters occupy none. To determine the width of a string on the terminal screen, it is necessary to decode the UTF-8 sequence and then use the `wcwidth` function to test the display width of each character, or `wcswidth` to measure the entire string.

For instance, the `ls` program had to be modified, because without knowing the column widths of filenames, it cannot format the table layout in which it presents directories to the user. Similarly, all programs that assume somehow that the output is presented in a fixed-width font and format it accordingly have to learn how to count columns in UTF-8 text. Editor functions such as deleting a single character have to be slightly modified to delete all bytes that might belong to one character. Affected were for instance editors (`vi`, `emacs`, `readline`, etc.) as well as programs that use the `ncurses` library.

Any Unix-style kernel can do fine with soft conversion and needs only very minor modifications to fully support UTF-8. Most kernel functions that handle strings (e.g. file names, environment variables, etc.) are not affected at all by the encoding. Modifications were necessary in Linux the following places:

- The console display and keyboard driver (another VT100 emulator) have to encode and decode UTF-8 and should support at least some subset of the Unicode character set. This had already been available in Linux as early as kernel 1.2 (send ESC %G to the console to activate UTF-8 mode).
- External file system drivers such as VFAT and WinNT have to convert file name character encodings. UTF-8 is one of the available conversion options, and the `mount` command has to tell the kernel driver that user processes shall see UTF-8 file names. Since VFAT and WinNT use already Unicode anyway, UTF-8 is the only available encoding that guarantees a lossless conversion here.
- The tty driver of any POSIX system supports a "cooked" mode, in which some primitive line editing functionality is available. In order to allow the character erase function to work properly, `stty` has to set a UTF-8 mode in the tty driver such that it does not count continuation bytes in the range 0x80-0xBF as characters. There exist some Linux patches for `stty` and the kernel tty driver from Bruno Haible.

C support for Unicode and UTF-8

Starting with GNU glibc 2.2, the type `wchar_t` is officially intended to be used only for 32-bit ISO 10646 values, independent of the currently used locale. This is signalled to applications by the definition of the `__STDC_ISO_10646__` macro as required by ISO C99. The ISO C multi-byte conversion functions (`mbsrtowcs()`, `wcsrtombs()`, etc.) are fully implemented in glibc 2.2 or higher and can be used to convert between `wchar_t` and any locale-dependent multibyte encoding, including UTF-8, ISO 8859-1, etc.

For example, you can write

```
#include <stdio.h>
#include <locale.h>

int main()
{
  if (!setlocale(LC_CTYPE, "")) {
    fprintf(stderr, "Can't set the specified locale! "
            "Check LANG, LC_CTYPE, LC_ALL.\n");
    return 1;
  }
  printf("%ls\n", L"Schöne Grüße");
  return 0;
}
```

Call this program with the locale setting `LANG=de_DE` and the output will be in ISO 8859-1. Call it with `LANG=de_DE.UTF-8` and the output will be in UTF-8. The `%ls` format specifier in `printf` calls `wcsrtombs` in order to convert the wide character argument string into the local-dependent multi-byte encoding.

Many of C's string functions are locale-independent and they just look at zero-terminated byte sequences:

```
strcpy strncpy strcat strncat strcmp strncmp strdup strchr strrchr
strcspn strspn strpbrk strstr strtok
```

Some of these (e.g. strcpy) can equally be used for single-byte (ISO 8859-1) and multi-byte (UTF-8) encoded character sets, as they need no notion of how many byte long a character is, while others (e.g., strchr) depend on one character being encoded in a single char value and are of less use for UTF-8 (strchr still works fine if you just search for an ASCII character in a UTF-8 string).

Other C functions are locale dependent and work in UTF-8 locales just as well:

```
strcoll strxfrm
```

How should the UTF-8 mode be activated?

If your application is soft converted and does not use the standard locale-dependent C multibyte routines (`mbsrtowcs()`, `wcsrtombs()`, etc.) to convert everything into `wchar_t` for processing, then it might have to find out in some way, whether it is supposed to assume that the text data it handles is in some 8-bit encoding (like ISO 8859-1, where 1 byte = 1 character) or UTF-8. Once everyone uses only UTF-8, you can just make it the default, but until then both the classical 8-bit sets and UTF-8 may still have to be supported.

The first wave of applications with UTF-8 support used a whole lot of different command line switches to activate their respective UTF-8 modes, for instance the famous `xterm -u8`. That turned out to be a very bad idea. Having to remember a special command line option or other configuration mechanism for *every* application is very tedious, which is why command line options are **not** the proper way of activating a UTF-8 mode.

The proper way to activate UTF-8 is the POSIX locale mechanism. A locale is a configuration setting that contains information about culture-specific conventions of software behaviour, including the character encoding, the date/time notation, alphabetic sorting rules, the measurement system and common office paper size, etc. The names of locales usually consist of ISO 639-1 language and ISO 3166-1 country codes, sometimes with additional encoding names or other qualifiers.

You can get a list of all locales installed on your system (usually in `/usr/lib/locale/`) with the command `locale -a`. Set the environment variable `LANG` to the name of your preferred locale. When a C program executes the `setlocale(LC_CTYPE, "")` function, the library will test the environment variables `LC_ALL`, `LC_CTYPE`, and `LANG` in that order, and the first one of these that has a value will determine which locale data is loaded for the `LC_CTYPE` category (which controls the multibyte conversion functions). The locale data is split up into separate categories. For example, `LC_CTYPE` defines the character encoding and `LC_COLLATE` defines the string sorting order. The `LANG` environment variable is used to set the default locale for all categories, but the `LC_*` variables can be used to override individual categories. Don't worry too much about the country identifiers in the locales. Locales such as `en_GB` (English in Great Britain) and `en_AU` (English in Australia) differ usually only in the `LC_MONETARY` category (name of currency, rules for printing monetary amounts), which practically no Linux application ever uses. `LC_CTYPE=en_GB` and `LC_CTYPE=en_AU` have

exactly the same effect.

You can query the name of the character encoding in your current locale with the command `locale charmap`. This should say `UTF-8` if you successfully picked a UTF-8 locale in the LC_CTYPE category. The command `locale -m` provides a list with the names of all installed character encodings.

If you use exclusively C library multibyte functions to do all the conversion between the external character encoding and the `wchar_t` encoding that you use internally, then the C library will take care of using the right encoding according to `LC_CTYPE` for you and your program does not even have to know explicitly what the current multibyte encoding is.

However, if you prefer not to do everything using the libc multi-byte functions (e.g., because you think this would require too many changes in your software or is not efficient enough), then your application has to find out for itself when to activate the UTF-8 mode. To do this, on any X/Open compliant systems, where `<langinfo.h>` is available, you can use a line such as

```
  utf8_mode = (strcmp(nl_langinfo(CODESET), "UTF-8") == 0);
```

in order to detect whether the current locale uses the UTF-8 encoding. You have of course to add a `setlocale(LC_CTYPE, "")` at the beginning of your application to set the locale according to the environment variables first. The standard function call `nl_langinfo(CODESET)` is also what `locale charmap` calls to find the name of the encoding specified by the current locale for you. It is available on pretty much every modern Unix now. FreeBSD added `nl_langinfo(CODESET)` support with version 4.6 (2002-06). If you need an autoconf test for the availability of `nl_langinfo(CODESET)`, here is the one Bruno Haible suggested:

```
======================== m4/codeset.m4 ================================
#serial AM1

dnl From Bruno Haible.

AC_DEFUN([AM_LANGINFO_CODESET],
[
  AC_CACHE_CHECK([for nl_langinfo and CODESET], am_cv_langinfo_codeset,
    [AC_TRY_LINK([#include <langinfo.h>],
      [char* cs = nl_langinfo(CODESET);],
      am_cv_langinfo_codeset=yes,
      am_cv_langinfo_codeset=no)
    ])
  if test $am_cv_langinfo_codeset = yes; then
    AC_DEFINE(HAVE_LANGINFO_CODESET, 1,
      [Define if you have <langinfo.h> and nl_langinfo(CODESET).])
  fi
])
=======================================================================
```

[You could also try to query the locale environment variables yourself without using `setlocale()`. In

the sequence `LC_ALL`, `LC_CTYPE`, `LANG`, look for the first of these environment variables that has a value. Make the UTF-8 mode the default (still overridable by command line switches) when this value contains the substring `UTF-8`, as this indicates reasonably reliably that the C library has been asked to use a UTF-8 locale. An example code fragment that does this is

```
  char *s;
  int utf8_mode = 0;

  if (((s = getenv("LC_ALL"))   && *s) ||
      ((s = getenv("LC_CTYPE")) && *s) ||
      ((s = getenv("LANG"))     && *s)) {
    if (strstr(s, "UTF-8"))
      utf8_mode = 1;
  }
```

This relies of course on all UTF-8 locales having the name of the encoding in their name, which is not always the case, therefore the `nl_langinfo()` query is clearly the better method. If you are really concerned that calling `nl_langinfo()` might not be portable enough, there is also Markus Kuhn's portable public domain `nl_langinfo(CODESET)` emulator for systems that don't have the real thing (and another one from Bruno Haible), and you can use the `norm_charmap()` function to standardize the output of the `nl_langinfo(CODESET)` on different platforms.]

How do I get a UTF-8 version of xterm?

The xterm version that comes with XFree86 4.0 or higher (maintained by Thomas Dickey) includes UTF-8 support. To activate it, start xterm in a UTF-8 locale and use a font with `iso10646-1` encoding, for instance with

```
  LC_CTYPE=en_GB.UTF-8 xterm \
    -fn '-Misc-Fixed-Medium-R-SemiCondensed--13-120-75-75-C-60-
ISO10646-1'
```

and then cat some example file, such as UTF-8-demo.txt in the newly started xterm and enjoy what you see.

If you are not using XFree86 4.0 or newer, then you can alternatively download the latest xterm development version separately and compile it yourself with "`./configure --enable-wide-chars ; make`" or alternatively with "`xmkmf; make Makefiles; make; make install; make install.man`".

If you do not have UTF-8 locale support available, use command line option `-u8` when you invoke xterm to switch input and output to UTF-8.

How much of Unicode does xterm support?

Xterm in XFree86 4.0.1 only supported Level 1 (no combining characters) of ISO 10646-1 with fixed character width and left-to-right writing direction. In other words, the terminal semantics were basically the same as for ISO 8859-1, except that it can now decode UTF-8 and can access 16-bit characters.

With XFree86 4.0.3, two important functions were added:

- automatic switching to a double-width font for CJK ideographs
- simple overstriking combining characters

If the selected normal font is *X* × *Y* pixels large, then xterm will attempt to load in addition a *2X* × *Y* pixels large font (same XLFD, except for a doubled value of the `AVERAGE_WIDTH` property). It will use this font to represent all Unicode characters that have been assigned the *East Asian Wide (W)* or *East Asian FullWidth (F)* property in Unicode Technical Report #11.

The following fonts coming with XFree86 4.x are suitable for display of Japanese and Korean Unicode text with terminal emulators and editors:

```
  6x13    -Misc-Fixed-Medium-R-SemiCondensed--13-120-75-75-C-60-
ISO10646-1
  6x13B   -Misc-Fixed-Bold-R-SemiCondensed--13-120-75-75-C-60-ISO10646-
1
  6x13O   -Misc-Fixed-Medium-O-SemiCondensed--13-120-75-75-C-60-
ISO10646-1
  12x13ja -Misc-Fixed-Medium-R-Normal-ja-13-120-75-75-C-120-ISO10646-1

  9x18    -Misc-Fixed-Medium-R-Normal--18-120-100-100-C-90-ISO10646-1
  9x18B   -Misc-Fixed-Bold-R-Normal--18-120-100-100-C-90-ISO10646-1
  18x18ja -Misc-Fixed-Medium-R-Normal-ja-18-120-100-100-C-180-ISO10646-
1
  18x18ko -Misc-Fixed-Medium-R-Normal-ko-18-120-100-100-C-180-ISO10646-
1
```

Some simple support for nonspacing or enclosing combining characters (i.e., those with general category code Mn or Me in the Unicode database) is now also available, which is implemented by just overstriking (logical OR-ing) a base-character glyph with up to two combining-character glyphs. This produces acceptable results for accents below the base line and accents on top of small characters. It also works well, for example, for Thai and Korean Hangul Conjoining Jamo fonts that were specifically designed for use with overstriking. However, the results might not be fully satisfactory for combining accents on top of tall characters in some fonts, especially with the fonts of the "fixed" family. Therefore precomposed characters will continue to be preferable where available.

The fonts below that come with XFree86 4.x are suitable for display of Latin etc. combining characters (extra head-space). Other fonts will only look nice with combining accents on small x-high characters.

```
  6x12    -Misc-Fixed-Medium-R-Semicondensed--12-110-75-75-C-60-
ISO10646-1
```

```
  9x18    -Misc-Fixed-Medium-R-Normal--18-120-100-100-C-90-ISO10646-1
  9x18B   -Misc-Fixed-Bold-R-Normal--18-120-100-100-C-90-ISO10646-1
```

The following fonts coming with XFree86 4.x are suitable for display of Thai combining characters:

```
  6x13    -Misc-Fixed-Medium-R-SemiCondensed--13-120-75-75-C-60-
ISO10646-1
  9x15    -Misc-Fixed-Medium-R-Normal--15-140-75-75-C-90-ISO10646-1
  9x15B   -Misc-Fixed-Bold-R-Normal--15-140-75-75-C-90-ISO10646-1
  10x20   -Misc-Fixed-Medium-R-Normal--20-200-75-75-C-100-ISO10646-1
  9x18    -Misc-Fixed-Medium-R-Normal--18-120-100-100-C-90-ISO10646-1
```

The fonts 18x18ko, 18x18Bko, 16x16Bko, and 16x16ko are suitable for displaying Hangul Jamo (using the same simple overstriking character mechanism used for Thai).

A note for programmers of text mode applications:

With support for CJK ideographs and combining characters, the output of xterm behaves a little bit more like with a proportional font, because a Latin/Greek/Cyrillic/etc. character requires one column position, a CJK ideograph two, and a combining character zero.

The Open Group's Single UNIX Specification specifies the two C functions wcwidth() and wcswidth() that allow an application to test how many column positions a character will occupy:

```
  #include <wchar.h>
  int wcwidth(wchar_t wc);
  int wcswidth(const wchar_t *pwcs, size_t n);
```

Markus Kuhn's free wcwidth() implementation can be used by applications on platforms where the C library does not yet provide a suitable function.

Xterm will for the foreseeable future probably not support the following functionality, which you might expect from a more sophisticated full Unicode rendering engine:

- bidirectional output of Hebrew and Arabic characters
- substitution of Arabic presentation forms
- substitution of Indic/Syriac ligatures
- arbitrary stacks of combining characters

Hebrew and Arabic users will therefore have to use application programs that reverse and left-pad Hebrew and Arabic strings before sending them to the terminal. In other words, the bidirectional processing has to be done by the application and not by xterm. The situation for Hebrew and Arabic improves over ISO 8859 at least in the form of the availability of precomposed glyphs and presentation forms. It is far from clear at the moment, whether bidirectional support should really go into xterm and how precisely this should work. Both ISO 6429 = ECMA-48 and the Unicode bidi algorithm provide alternative starting points. See also ECMA Technical Report TR/53.

If you plan to support bidirectional text output in your application, have a look at either Dov Grobgeld's FriBidi or Mark Leisher's Pretty Good Bidi Algorithm, two free implementations of the Unicode bidi algorithm.

Xterm currently does not support the Arabic, Syriac, or Indic text formatting algorithms, although Robert Brady has published some experimental patches towards bidi support. It is still unclear whether it is feasible or preferable to do this in a VT100 emulator at all. Applications can apply the Arabic and Hangul formatting algorithms themselves easily, because xterm allows them to output the necessary presentation forms. For Hangul, Unicode contains the presentation forms needed for modern (post-1933) Korean orthography. For Indic scripts, the X font mechanism at the moment does not even support the encoding of the necessary ligature variants, so there is little xterm could offer anyway. Applications requiring Indic or Syriac output should better use a proper Unicode X11 rendering library such as Pango instead of a VT100 emulator like xterm.

Where do I find ISO 10646-1 X11 fonts?

Quite a number of Unicode fonts have become available for X11 over the past few months, and the list is growing quickly:

- Markus Kuhn together with a number of other volunteers has extended the old `-misc-fixed-*-iso8859-1` fonts that come with X11 towards a repertoire that covers all European characters (Latin, Greek, Cyrillic, intl. phonetic alphabet, mathematical and technical symbols, in some fonts even Armenian, Georgian, Katakana, Thai, and more). For more information see the Unicode fonts and tools for X11 page. These fonts are now also distributed with XFree86 4.0.1 or higher.
- Markus has also prepared ISO 10646-1 versions of all the Adobe and B&H BDF fonts in the X11R6.4 distribution. These fonts already contained the full PostScript font repertoire (around 30 additional characters, mostly those used also by CP1252 MS-Windows, e.g. smart quotes, dashes, etc.), which were however not available under the ISO 8859-1 encoding. They are now all accessible in the ISO 10646-1 version, along with many additional precomposed characters covering ISO 8859-1,2,3,4,9,10,13,14,15. These fonts are now also distributed with XFree86 4.1 or higher.
- XFree86 4.0 comes with an integrated TrueType font engine that can make available any Apple/ Microsoft font to your X application in the ISO 10646-1 encoding.
- Some future XFree86 release might also remove most old BDF fonts from the distribution and replace them with ISO 10646-1 encoded versions. The X server will be extended with an automatic encoding converter that creates other font encodings such as ISO 8859-* from the ISO 10646-1 font file on-the-fly when such a font is requested by old 8-bit software. Modern software should preferably use the ISO 10646-1 font encoding directly.
- ClearlyU (cu12) is a 12 point, 100 dpi proportional ISO 10646-1 BDF font for X11 with over 3700 characters by Mark Leisher (example images).
- The Electronic Font Open Laboratory in Japan is also working on a family of Unicode bitmap fonts.
- Dmitry Yu. Bolkhovityanov created a Unicode VGA font in BDF for use by text mode IBM PC

emulators etc.

- Roman Czyborra's GNU Unicode font project works on collecting a complete and free 8×16/16×16 pixel Unicode font. It currently covers over 34000 characters.
- etl-unicode is an ISO 10646-1 BDF font prepared by Primoz Peterlin.
- Primoz Peterlin has also started the freefont project, which extends to better UCS coverage some of the 35 core PostScript outline fonts that URW++ donated to the ghostscript project, with the help of pfaedit.
- George Williams has created a Type1 Unicode font family, which is also available in BDF. He also developed the PfaEdit PostScript and bitmap font editor.
- EversonMono is a shareware monospaced font with over 3000 European glyphs, also available from the DKUUG server.
- Birger Langkjer has prepared a Unicode VGA Console Font for Linux.
- Alan Wood has a list of Microsoft fonts that support various Unicode ranges.

Unicode X11 font names end with `-ISO10646-1`. This is now the officially registered value for the X Logical Font Descriptor (XLFD) fields `CHARSET_REGISTRY` and `CHARSET_ENCODING` for all Unicode and ISO 10646-1 16-bit fonts. The `*-ISO10646-1` fonts contain some unspecified subset of the entire Unicode character set, and users have to make sure that whatever font they select covers the subset of characters needed by them.

The `*-ISO10646-1` fonts usually also specify a `DEFAULT_CHAR` value that points to a special non-Unicode glyph for representing any character that is not available in the font (usually a dashed box, the size of an H, located at 0x00). This ensures that users at least see clearly that there is an unsupported character. The smaller fixed-width fonts such as 6x13 etc. for xterm will never be able to cover all of Unicode, because many scripts such as Kanji can only be represented in considerably larger pixel sizes than those widely used by European users. Typical Unicode fonts for European usage will contain only subsets of between 1000 and 3000 characters, such as the CEN MES-3 repertoire.

You might notice that in the *-ISO10646-1 fonts the shapes of the ASCII quotation marks has slightly changed to bring them in line with the standards and practice on other platforms.

What are the issues related to UTF-8 terminal emulators?

VT100 terminal emulators accept ISO 2022 (=ECMA-35) ESC sequences in order to switch between different character sets.

UTF-8 is in the sense of ISO 2022 an "other coding system" (see section 15.4 of ECMA 35). UTF-8 is outside the ISO 2022 SS2/SS3/G0/G1/G2/G3 world, so if you switch from ISO 2022 to UTF-8, all SS2/SS3/G0/G1/G2/G3 states become meaningless until you leave UTF-8 and switch back to ISO 2022. UTF-8 is a stateless encoding, i.e. a self-terminating short byte sequence determines completely which character is meant, independent of any switching state. G0 and G1 in ISO 10646-1 are those of ISO 8859-1, and G2/G3 do not exist in ISO 10646, because every character has a fixed position and no switching takes place. With UTF-8, it is not possible that your terminal remains switched to strange graphics-character mode after you accidentally dumped a binary file to it. This makes a terminal in UTF-8 mode much more robust than with ISO 2022 and it is therefore useful to have a way of locking a terminal into

UTF-8 mode such that it can't accidentally go back to the ISO 2022 world.

The ISO 2022 standard specifies a range of ESC % sequences for leaving the ISO 2022 world (designation of other coding system, DOCS), and a number of such sequences have been registered for UTF-8 in section 2.8 of the ISO 2375 International Register of Coded Character Sets:

- `ESC %G` activates UTF-8 with an unspecified implementation level from ISO 2022 in a way that allows to go back to ISO 2022 again.
- `ESC %@` goes back from UTF-8 to ISO 2022 in case UTF-8 had been entered via `ESC %G`.
- `ESC %/G` switches to UTF-8 Level 1 with no return.
- `ESC %/H` switches to UTF-8 Level 2 with no return.
- `ESC %/I` switches to UTF-8 Level 3 with no return.

While a terminal emulator is in UTF-8 mode, any ISO 2022 escape sequences such as for switching G2/G3 etc. are ignored. The only ISO 2022 sequence on which a terminal emulator might act in UTF-8 mode is `ESC %@` for returning from UTF-8 back to the ISO 2022 scheme.

UTF-8 still allows you to use C1 control characters such as CSI, even though UTF-8 also uses bytes in the range 0x80-0x9F. It is important to understand that a terminal emulator in UTF-8 mode must apply the UTF-8 decoder to the incoming byte stream **before** interpreting any control characters. C1 characters are UTF-8 decoded just like any other character above U+007F.

Many text-mode applications available today expect to speak to the terminal using a legacy encoding or to use ISO 2022 sequences for switching terminal fonts. In order to use such applications within a UTF-8 terminal emulator, it is possible to use a conversion layer that will translate between ISO 2022 and UTF-8 on the fly. One such utility is Juliusz Chroboczek's luit. If all you need is ISO 8859 support in a UTF-8 terminal, you can also use screen (version 4.0 or newer) by Michael Schröder and Jürgen Weigert. As implementation of ISO 2022 is a complex and error-prone task, better avoid implementing ISO 2022 yourself. Implement only UTF-8 and point users who need ISO 2022 at luit (or screen).

What UTF-8 enabled applications are available?

Warning: As of mid-2003, this section is becoming increasingly incomplete. UTF-8 support is now a pretty standard feature for most well-maintained packages. This list will soon have to be converted into a list of the most popular programs that still have problems with UTF-8.

Terminal emulation and communication

- xterm as shipped with XFree86 4.0 or higher works correctly in UTF-8 locales if you use an *-iso10646-1 font. Just try it with for example `LC_CTYPE=en_GB.UTF-8 xterm -fn '-Misc-Fixed-Medium-R-Normal--18-120-100-100-C-90-ISO10646-1'`.
- C-Kermit has supported UTF-8 as the transfer, terminal, and file character set since version 7.0.
- mlterm is a multi-lingual terminal emulator that supports UTF-8 among many other encodings, combining characters, XIM.
- Edmund Grimley Evans extended the BOGL Linux framebuffer graphics library with UCS font

support and built a simple UTF-8 console terminal emulator called `bterm` with it.
- Uterm purports to be a UTF-8 terminal emulator for the Linux framebuffer console.

Editing and word processing

- Vim (the popular clone of the classic vi editor) supports UTF-8 with wide characters and up to two combining characters starting from version 6.0.
- Emacs 21.2 has quite good basic UTF-8 support in the form of the `mule-utf-8` coding system. This is expected to improve significantly once ongoing work to change the internal encoding of Emacs/MULE entirely to UTF-8 is completed, which is planned for Emacs 22.
- Yudit is Gaspar Sinai's free X11 Unicode editor.
- Mined 2000 by Thomas Wolff is a very nice UTF-8 capable text editor, ahead of the competition with features such as not only support of double-width and combining characters, but also bidirectional scripts, keyboard mappings for a wide range of scripts, script-dependent highlighting, etc.
- [NEW] JOE is a popular WordStar-like editor that supports UTF-8 as of version 3.0.
- Cooledit offers UTF-8 and UCS support starting with version 3.15.0.
- QEmacs is a small editor for use on UTF-8 terminals.
- less is a popular plain-text file viewer that had UTF-8 support since version 348. (Version 358 had a bug related to the handling of UTF-8 characters and backspace underlining/boldification as used by nroff/man, for which a patch is available, version 381 still has problems with UTF-8 characters in the search-mode input line.)
- GNU bash and readline provide single-line editors and they introduced support for multi-byte character encodings such as UTF-8 with versions bash 2.05b and readline 4.3.
- gucharmap and UMap are tools to select and paste any Unicode character into your application.
- [NEW] LaTeX has supported UTF-8 in its base package since March 2004 (still experimental). You can simply write `\usepackage[utf8]{inputenc}` and then encode at least some of TeX's standard character repertoire in UTF-8 in your LaTeX sources. (Before that, UTF-8 was already available in the form of Dominique Unruh's package, which covered far more characters and was rather resource hungry.)
- Abiword.

Programming

- Perl offers proper Unicode and UTF-8 support starting with version 5.8. Strings are now tagged in memory as either byte strings or character strings, and the latter are stored internally as UTF-8 but appear to the programmer just as sequences of UCS characters. There is now also comprehensive support for encoding conversion and normalization included. Read "man perluniintro" for details.
- Python got Unicode support added in version 1.6.
- Tcl/Tk started using Unicode as its base character set with version 8.1. ISO10646-1 fonts are supported in Tk from version 8.3.3 or newer.
- CLISP can work with all multi-byte encodings (including UTF-8) and with the functions `char-width` and `string-width` there is an API comparable to `wcwidth()` and `wcswidth()` available.

Mail and Internet

- The Mutt email client has worked since version 1.3.24 in UTF-8 locales. When compiled and linked with ncursesw (ncurses built with wide-character support), Mutt 1.3.x works decently in UTF-8 locales under UTF-8 terminal emulators such as xterm.
- Exmh is a GUI frontend for the MH mail system and partially supports Unicode starting with version 2.1.1 if Tcl/Tk 8.3.3 or newer is used. To enable displaying UTF-8 email, make sure you have the *-iso10646-1 fonts installed and add to .Xdefaults the line "exmh.mimeUCharsets: utf-8". Much of the Exmh-internal MIME charset-set mechanics however still dates from the days before Tcl 8.1, therefore ignores Tcl/Tk's more recent Unicode support, and could now be simplified and improved significantly. In particular, writing or replying to UTF-8 mail is still broken.
- Most modern web browsers such as Mozilla have pretty decent UTF-8 support today.

Printing

- Cedilla is Juliusz Chroboczek's best-effort Unicode to PostScript text printer.
- Markus Kuhn's hpp is a very simple plain text formatter for HP PCL printers that supports the repertoire of characters covered by the standard PCL fixed-width fonts in all the character encodings for which your C library has a locale mapping. Markus Kuhn's utf2ps is an early quick-and-dirty proof-of-concept UTF-8 formatter for PostScript, that was only written to demonstrate which character repertoire can easily be printed using only the standard PostScript fonts and was never intended to be actually used.
- The Common UNIX Printing System comes with a texttops tool that converts plaintext UTF-8 to PostScript.
- txtbdf2ps by Serge Winitzki is a Perl script to print UTF-8 plaintext to PostScript using BDF pixel fonts.

Misc

- The PostgreSQL DBMS had support for UTF-8 since version 7.1, both as the frontend encoding, and as the backend storage encoding. Data conversion between frontend and backend encodings is performed automatically.
- FIGlet is a tool to output banner text in large letters using monospaced characters as block graphics elements and added UTF-8 support in version 2.2.
- Charlint is a character normalization tool for the W3C character model.
- The first available UTF-8 tools for Unix came out of the Plan9 project, Bell Lab's Unix successor and the world's first operating system using UTF-8. Plan9's Sam editor and 9term terminal emulator have also been ported to Unix. Wily started out as a Unix implementation of the Plan9 Acme editor and is a mouse-oriented, text-based working environment for programmers.
- The Gnumeric spreadsheet is fully Unicode based from version 1.1.
- The Heirloom Toolchest is a collection of standard Unix utilities derived from original Unix material released as open source by Caldera with support for multibyte character sets, especially UTF-8.

- convmv is a tool to convert the filenames in entire directory trees from a legacy encoding to UTF-8.

What patches to improve UTF-8 support are available?

Many of these already have been included in the respective main distribution.

- The Advanced Utility Development subgroup of the OpenI18N (formerly Li18nux) project have prepared various internationalization patches for tools such as cut, fold, glibc, join, sed, uniq, xterm, etc. that might improve UTF-8 support.
- A collection of UTF-8 patches for various tools as well as a UTF-8 support status list is in Bruno Haible's Unicode-HOWTO.
- Bruno Haible has also prepared various patches for stty, the Linux kernel tty, etc.
- The multilingualization patch (w3m-m17n) for the text-mode web browser w3m allows you to view documents in all the common encodings on a UTF-8 terminal like xterm (also switch option "Use alternate expression with ASCII for entity" to OFF after pressing "o"). Another multilingual version (w3mmee) is available as well (haven't tried that yet).

Are there free libraries for dealing with Unicode available?

- Ulrich Drepper's GNU C library glibc has featured since version 2.2 full multi-byte locale support for UTF-8, a Unicode sorting order algorithm, and it can recode into many other encodings. All current Linux distributions come with glibc 2.2 or newer, so you definitely should upgrade now if you are still using an earlier Linux C library.
- The International Components for Unicode (ICU) (formerly IBM Classes for Unicode) have become what is probably the most powerful cross-platform standard library for more advanced Unicode character processing functions.
- X.Net's xIUA is a package designed to retrofit existing code for ICU support by providing locale management so that users do not have to modify internal calling interfaces to pass locale parameters. It uses more familiar APIs, for example to collate you use xiua_strcoll, and is thread safe.
- Mark Leisher's UCData Unicode character property and bidi library as well as his `wchar_t` support test code.
- Bruno Haible's libiconv character-set conversion library provides an iconv() implementation, for use on systems which don't have one, or whose implementation cannot convert from/to Unicode. It also contains the libcharset character-encoding query library that allows applications to determine in a highly portable way the character encoding of the current locale, avoiding the portability concerns of using nl_langinfo(CODESET) directly.
- Bruno Haible's libutf8 provides various functions for handling UTF-8 strings, especially for platforms that do not yet offer proper UTF-8 locales.
- Tom Tromey's libunicode library is part of the Gnome Desktop project, but can be built independently of Gnome. It contains various character class and conversion functions. (CVS)
- FriBidi is Dov Grobgeld's free implementation of the Unicode bidi algorithm.
- Markus Kuhn's free wcwidth() implementation can be used by applications on platforms where

the C library does not yet provide an equivalent function to find, how many column positions a character or string will occupy on a UTF-8 terminal emulator screen.

- Markus Kuhn's transtab is a transliteration table for applications that have to make a best-effort conversion from Unicode to ASCII or some 8-bit character set. It contains a comprehensive list of substitution strings for Unicode characters, comparable to the fallback notations that people use commonly in email and on typewriters to represent unavailable characters. The table comes in ISO/IEC TR 14652 format, to allow simple inclusion into POSIX locale definition files.

What is the status of Unicode support for various X widget libraries?

- The Pango - Unicode and Complex Text Processing project added full-featured Unicode support to GTK+.
- Qt supported the use of *-ISO10646-1 fonts since version 2.0.
- A UTF-8 extension for the Fast Light Tool Kit was prepared by Jean-Marc Lienher, based on his Xutf8 Unicode display library.

What packages with UTF-8 support are currently under development?

- Native Unicode support is planned for Emacs 22. If you are interested in contributing/testing, please ask Eli Zaretskii to put you onto the `emacs-unicode@gnu.org` mailing list.
- The Linux Console Project works on a complete revision of the VT100 emulator built into the Linux kernel, which will improve the simplistic UTF-8 support already there.

How does UTF-8 support work under Solaris?

Starting with Solaris 2.8, UTF-8 is at least partially supported. To use it, just set one of the UTF-8 locales, for instance by typing

```
setenv LANG en_US.UTF-8
```

in a C shell.

Now the `dtterm` terminal emulator can be used to input and output UTF-8 text and the `mp` print filter will print UTF-8 files on PostScript printers. The `en_US.UTF-8` locale is at the moment supported by Motif and CDE desktop applications and libraries, but not by OpenWindows, XView, and OPENLOOK DeskSet applications and libraries.

For more information, read Sun's Overview of en_US.UTF-8 Locale Support web page.

Can I use UTF-8 on the Web?

Yes. There are two ways in which a HTTP server can indicate to a client that a document is encoded in UTF-8:

- Make sure that the HTTP header of a document contains the line

  ```
  Content-Type: text/html; charset=utf-8
  ```

  if the file is HTML, or the line

  ```
  Content-Type: text/plain; charset=utf-8
  ```

  if the file is plain text. How this can be achieved depends on your web server. If you use Apache and you have a subdirecory in which all *.html or *.txt files are encoded in UTF-8, then create there a file .htaccess and add to it the two lines

  ```
  AddType text/html;charset=UTF-8 html
  AddType text/plain;charset=UTF-8 txt
  ```

  A webmaster can modify /etc/httpd/mime.types to make the same change for all subdirectories simultaneously.
- If you can't influence the HTTP headers that the web server prefixes to your documents automatically, then add in a HTML document under HEAD the element

  ```
  <META http-equiv=Content-Type content="text/html; charset=UTF-8">
  ```

  which usually has the same effect. This obviously works only for HTML files, not for plain text. It also announces the encoding of the file to the parser only after the parser has already started to read the file, so it is clearly the less elegant approach.

The currently most widely used browsers support UTF-8 well enough to generally recommend UTF-8 for use on web pages. The old Netscape 4 browser used an annoyingly large single font for displaying any UTF-8 document. Best upgrade to Mozilla, Netscape 6 or some other recent browser (Netscape 4 is generally very buggy and not maintained any more).

There is also the question of how non-ASCII characters entered into HTML forms are encoded in the subsequent HTTP GET or POST request that transfers the field contents to a CGI script on the server. Unfortunately, both standardization and implementation are still a huge mess here, as discussed in the FORM submission and i18n tutorial by Alan Flavell. We can only hope that a practice of doing all this in UTF-8 will emerge eventually. See also the discussion about Mozilla bug 18643.

How are PostScript glyph names related to UCS codes?

See Adobe's Unicode and Glyph Names guide.

Are there any well-defined UCS subsets?

With over 40000 characters, a full and complete Unicode implementation is an enormous project. However, it is often sufficient (especially for the European market) to implement only a few hundred or thousand characters as before and still enjoy the simplicity of reaching all required characters in just one single simple encoding via Unicode. A number of different UCS subsets already have been established:

- The Windows Glyph List 4.0 (WGL4) is a set of 650 characters that covers all the 8-bit MS-DOS, Windows, Mac, and ISO code pages that Microsoft had used before. All Windows fonts now cover at least the WGL4 repertoire. WGL4 is a superset of CEN MES-1. (WGL4 test file).
- Three European UCS subsets MES-1, MES-2, and MES-3 have been defined by the European standards committee CEN/TC304 in CWA 13873:
    - MES-1 is a very small Latin subset with only 335 characters. It contains exactly all characters found in ISO 6937 plus the EURO SIGN. This means MES-1 contains all characters of ISO 8859 parts 1,2,3,4,9,10,15. [Note: If your aim is to provide only the cheapest and simplest reasonable Central European UCS subset, I would implement MES-1 plus the following important 14 additional characters found in Windows code page 1252 but not in MES-1: U+0192, U+02C6, U+02DC, U+2013, U+2014, U+201A, U+201E, U+2020, U+2021, U+2022, U+2026, U+2030, U+2039, U+203A.]
    - MES-2 is a Latin/Greek/Cyrillic/Armenian/Georgian subset with 1052 characters. It covers every language and every 8-bit code page used in Europe (not just the EU!) and European language countries. It also adds a small collection of mathematical symbols for use in technical documentation. MES-2 is a superset of MES-1. If you are developing only for a European or Western market, MES-2 is the recommended repertoire. [Note: For bizarre committee-politics reasons, the following eight WGL4 characters are missing from MES-2: U+2113, U+212E, U+2215, U+25A1, U+25AA, U+25AB, U+25CF, U+25E6. If you implement MES-2, you should definitely also add those and then you can claim WGL4 conformance in addition.]
    - MES-3 is a very comprehensive UCS subset with 2819 characters. It simply includes every UCS collection that seemed of potential use to European users. This is for the more ambitious implementors. MES-3 is a superset of MES-2 and WGL4.
- JIS X 0221-1995 specifies 7 non-overlapping UCS subsets for Japanese users:
    - Basic Japanese (6884 characters): JIS X 0208-1997, JIS X 0201-1997
    - Japanese Non-ideographic Supplement (1913 characters): JIS X 0212-1990 non-kanji, plus various other non-kanji
    - Japanese Ideographic Supplement 1 (918 characters): some JIS X 0212-1990 kanji
    - Japanese Ideographic Supplement 2 (4883 characters): remaining JIS X 0212-1990 kanji
    - Japanese Ideographic Supplement 3 (8745 characters): remaining Chinese characters
    - Full-width Alphanumeric (94 characters): for compatibility
    - Half-width Katakana (63 characters): for compatibility
- The ISO 10646 standard splits up its repertoire into a number of collections that can be used to define and document implemented subsets. Unicode defines similar, but not quite identical, blocks of characters, which correspond to sections in the Unicode standard.
- RFC 1815 is a memo written in 1995 by someone who obviously didn't like ISO 10646 and was unaware of JIS X 0221-1995. It discusses a UCS subset called "ISO-10646-J-1" consisting of 14 UCS collections, some of which are intersected with JIS X 0208. This is just what a particular font

in an old Japanese Windows NT version from 1995 happened to implement. RFC 1815 is completely obsolete and irrelevant today and should best be ignored.

- Markus Kuhn has defined in the ucs-fonts.tar.gz README three UCS subsets TARGET1, TARGET2, TARGET3 that are sensible extensions of the corresponding MES subsets and that were the basis for the completion of this xterm font package.

Markus Kuhn's uniset Perl script allows convenient set arithmetic over UCS subsets for anyone who wants to define a new one or wants to check coverage of an implementation.

What issues are there to consider when converting encodings

The Unicode Consortium maintains a collection of mapping tables between Unicode and various older encoding standards. It is important to understand that the primary purpose of these tables was to demonstrate that Unicode is a superset of the mapped legacy encodings, and to document the motivation and origin behind those Unicode characters that were included into the standard primarily for round-trip compatibility reasons with older character sets. The implementation of good character encoding conversion rountines is a significantly more complex task than just blindly applying these example mapping tables! This is because some character sets distinguish characters that others unify.

The Unicode mapping tables alone are to some degree well suited to directly convert text from the older encodings to Unicode. High-end conversion tools nevertheless should provide interactive mechanisms, where characters that are unified in the legacy encoding but distinguished in Unicode can interactively or semi-automatically be disambiguated on a case-by-case basis.

Conversion in the opposite direction from Unicode to a legacy character set requires non-injective (= many-to-one) extensions of these mapping tables. Several Unicode characters have to be mapped to a single code point in many legacy encodings. The Unicode consortium currently does not maintain standard many-to-one tables for this purpose and does not define any standard behavior of coded character set conversion tools.

Here are some examples for the many-to-one mappings that have to be handled when converting from Unicode into something else:

UCS characters	equivalent character	in target code
U+00B5 MICRO SIGN U+03BC GREEK SMALL LETTER MU	0xB5	ISO 8859-1
U+00C5 LATIN CAPITAL LETTER A WITH RING ABOVE U+212B ANGSTROM SIGN	0xC5	ISO 8859-1
U+03B2 GREEK CAPITAL LETTER BETA U+00DF LATIN SMALL LETTER SHARP S	0xE1	CP437

U+03A9 GREEK CAPITAL LETTER OMEGA U+2126 OHM SIGN	0xEA	CP437
U+03B5 GREEK SMALL LETTER EPSILON U+2208 ELEMENT OF	0xEE	CP437
U+005C REVERSE SOLIDUS U+FF3C FULLWIDTH REVERSE SOLIDUS	0x2140	JIS X 0208

A first approximation of such many-to-one tables can be generated from available normalization information, but these then still have to be manually extended and revised. For example, it seems obvious that the character 0xE1 in the original IBM PC character set was meant to be useable as both a Greek small beta (because it is located between the code positions for alpha and gamma) and as a German sharp-s character (because that code is produced when pressing this letter on a German keyboard). Similarly 0xEE can be either the mathematical element-of sign, as well as a small epsilon. These characters are not Unicode normalization equivalents, because although they look similar in low-resolution video fonts, they are very different characters in high-quality typography. IBM's tables for CP437 reflected one usage in some cases, Microsoft's the other, both equally sensible. A good code converter should aim to be compatible with both, and not just blindly use the Microsoft mapping table alone when converting from Unicode.

The Unicode database does contain in field 5 the Character Decomposition Mapping that can be used to generate some of the above example mappings automatically. As a rule, the output of a Unicode-to-Something converter should not depend on whether the Unicode input has first been converted into Normalization Form C or not. For equivalence information on Chinese, Japanese, and Korean Han/Kanji/Hanja characters, use the Unihan database. In the cases of the IBM PC characters in the above examples, where the normalization tables do not offer adequate mapping, the cross-references to similar looking characters in the Unicode book are a valuable source of suggestions for equivalence mappings. In the end, which mappings are used and which not is a matter of taste and observed usage.

The Unicode consortium used to maintain mapping tables to CJK character set standards, but has declared them to be obsolete, because their presence on the Unicode web server led to the development of a number of inadequate and naive EUC converters. In particular, the (now obsolete) CJK Unicode mapping tables had to be slightly modified sometimes to preserve information in combination encodings. For example, the standard mappings provide round-trip compatibility for conversion chains ASCII to Unicode to ASCII as well as for JIS X 0208 to Unicode to JIS X 0208. However, the EUC-JP encoding covers the union of ASCII and JIS X 0208, and the UCS repertoire covered by the ASCII and JIS X 0208 mapping tables overlaps for one character, namely U+005C REVERSE SOLIDUS. EUC-JP converters therefore have to use a slightly modified JIS X 0208 mapping table, such that the JIS X 0208 code 0x2140 (0xA1 0xC0 in EUC-JP) gets mapped to U+FF3C FULLWIDTH REVERSE SOLIDUS. This way, round-trip compatibility from EUC-JP to Unicode to EUC-JP can be guaranteed without any loss of information. Unicode Standard Annex #11: East Asian Width provides further guidance on this issue. Another problem area is compatibility with older conversion tables, as explained in an essay by Tomohiro Kubota.

In addition to just using standard normalization mappings, developers of code converters can also offer

transliteration support. Transliteration is the conversion of a Unicode character into a graphically and/or semantically similar character in the target code, even if the two are distinct characters in Unicode after normalization. Examples of transliteration:

UCS characters	equivalent character	in target code
U+0022 QUOTATION MARK U+201C LEFT DOUBLE QUOTATION MARK U+201D RIGHT DOUBLE QUOTATION MARK U+201E DOUBLE LOW-9 QUOTATION MARK U+201F DOUBLE HIGH-REVERSED-9 QUOTATION MARK	0x22	ISO 8859-1

The Unicode Consortium does not provide or maintain any standard transliteration tables at this time. CEN/TC304 has a draft report "European fallback rules" on recommended ASCII fallback characters for MES-2 in the pipeline, but this is not yet mature. Which transliterations are appropriate or not can in some cases depend on language, application field, and most of all personal preference. Available Unicode transliteration tables include, for example, those found in Bruno Haible's libiconv, the glibc 2.2 locales, and Markus Kuhn's transtab package.

Is X11 ready for Unicode?

The X11 R6.6 release (2001) is the latest version of the X Consortium's sample implementation of the X11 Window System standards. The bulk of the current X11 standards and the sample implementation pre-date widespread interest in Unicode under Unix. There are a number of problems and inconveniences for Unicode users in both that really should be fixed in the next X11 release:

- **UTF-8 cut and paste:** The ICCCM standard does not specify how to transfer UCS strings in selections. Some vendors have added UTF-8 as yet another encoding to the existing COMPOUND_TEXT mechanism (CTEXT). This is not a good solution for at least the following reasons:
  - CTEXT is a rather complicated ISO 2022 mechanism and Unicode offers the opportunity to provide not just another add-on to CTEXT, but to replace the entire monster with something far simpler, more convenient, and equally powerful.
  - Many existing applications can communicate selections via CTEXT, but do not support a newly added UTF-8 option. A user of CTEXT has to decide whether to use the old ISO 2022 encodings or the new UTF-8 encoding, but both cannot be offered simultaneously. In other words, adding UTF-8 to CTEXT seriously breaks backwards compatibility with existing CTEXT applications.
  - The current CTEXT specification even explicitly forbids the addition of UTF-8 in section 6: "ISO registered 'other coding systems' are not used in Compound Text; extended segments are the only mechanism for non-2022 encodings."

Juliusz Chroboczek has written an Inter-Client Exchange of Unicode Text draft proposal for an extension of the ICCCM to handle UTF-8 selections with a new UTF8_STRING atom that can be used as a property type and selection target. This clean approach fixes all of the above problems. UTF8_STRING is just as state-less and easy to use as the existing STRING atom (which is reserved exclusively for ISO 8859-1 strings and therefore not usable for UTF-8), and adding a new selection target allows applications to offer selections in both the old CTEXT and the new UTF8_STRING format simultaneously, which maximizes interoperability. The use of UTF8_STRING can be negociated between the selection holder and requestor, leading to no compatibility issues whatsoever. Markus Kuhn has prepared an ICCCM patch that adds the necessary definition to the standard. Current status: The UTF8_STRING atom has now been officially registered with X.Org, and an update of the ICCCM is expected for the next release.

- **Application window properties:** In order to assist the window manager in correctly labeling windows, the ICCCM 2.0 specification requires applications to assign properties such as WM_NAME, WM_ICON_NAME and WM_CLIENT_MACHINE to each window. The old ICCCM 2.0 (1993) defines these to be of the polymorphic type TEXT, which means that they can have their text encoding indicated using one of the property types STRING (ISO 8859-1), COMPOUND_TEXT (a ISO 2022 subset), or C_STRING (unknown character set). Simply adding UTF8_STRING as a new option for TEXT would break backwards compatibility with old window managers that do not know about this type. Therefore, the freedesktop.org draft standard developped in the Window Manager Specification Project adds new additional window properties _NET_WM_NAME, _NET_WM_ICON_NAME, etc. that have type UTF8_STRING.
- **Inefficient font data structures:** The Xlib API and X11 protocol data structures used for representing font metric information are extremely inefficient when handling sparsely populated fonts. The most common way of accessing a font in an X client is a call to XLoadQueryFont(), which allocates memory for an XFontStruct and fetches its content from the server. XFontStruct contains an array of XCharStruct entries (12 bytes each). The size of this array is the code position of the last character minus the code position of the first character plus one. Therefore, any "*-iso10646-1" font that contains both U+0020 and U+FFFD will cause an XCharStruct array with 65502 elements to be allocated (even for CharCell fonts), which requires 786 kilobytes of client-side memory and data transmission, even if the font contains only a thousand characters.

  A few workarounds have been used so far:

  - The non-Asian `-misc-fixed-*-iso10646-1` fonts that come with XFree86 4.0 contain no characters above U+31FF. This reduces the memory requirement to 153 kilobytes, which is still bad, but much less so. (There are actually many useful characters above U+31FF present in the BDF files, waiting for the day when this problem will be fixed, but they currently all have an encoding of -1 and are therefore ignored by the X server. If you need these characters, then just install the original fonts without applying the `bdftruncate` script).
  - Starting with XFree86 4.0.3, the truncation of a BDF font can also be done by specifying a character code subrange at the end of the XLFD, as described in the XLFD specification, section 3.1.2.12. For example,

    ```
    -Misc-Fixed-Medium-R-Normal--20-200-75-75-C-100-ISO10646-1
    [0x1200_0x137f]
    ```

will load only the Ethiopic part of this BDF font with a correspondingly nicely small XFontStruct. Earlier X server versions will simply ignore the font subset brackets and will give you the full font, so there is no compatibility problem with using that.
- Bruno Haible has written a BIGFONT protocol extension for XFree86 4.0, which uses a compressed transmission of XCharStruct from server to client and also uses shared memory in Xlib between several clients which have loaded the same font.

These workarounds do not solve the underlying problem that XFontStruct is unsuitable for sparsely populated fonts, but they do provide a significant efficiency improvement without requiring any changes in the API or client source code. One real solution would be to extend or replace XFontStruct with something slightly more flexible that contains a sorted list or hash table of characters as opposed to an array. This redesign of XFontStruct would at the same time also allow the addition of the urgently needed provisions for combining characters and ligatures.

Another approach would be to introduce a new font encoding, which could be called for instance "ISO10646-C" (the C stands for combining, complex, compact, or character-glyph mapped, as you prefer). In this encoding, the numbers assigned to each glyph are really font-specific glyph numbers and are not equivalent to any UCS character code positions. The information necessary to do a character-to-glyph mapping would have to be stored in to be standardized new properties. This new font encoding would be used by applications together with a few efficient C functions that perform the character-to-glyph code mapping:

- `makeiso10646cglyphmap(XFontStruct *font, iso10646cglyphmap *map)`
  Reads the character-to-glyph mapping table from the font properties into a compact and efficient in-memory representation.
- `freeiso10646cglyphmap(iso10646cglyphmap *map)`
  Frees that in-memory representation.
- `mbtoiso10646c(char *string, iso10646cglyphmap *map, XChar2b *output)`
  `wctoiso10646c(wchar_t *string, iso10646cglyphmap *map, XChar2b *output)`
  These take a Unicode character string and convert it into a `XChar2b` glyph string suitable for output by `XDrawString16` with the ISO10646-C font from which the `iso10646cglyphmap` was extracted.

ISO10646-C fonts would still be limited to having not more than 64 kibiglyphs, but these can come from anywhere in UCS, not just from the BMP. This solution also easily provides for glyph substitution, such that we can finally handle the Indic fonts. It solves the huge-XFontStruct problem of ISO10646-1, as XFontStruct grows now proportionally with the number of glyphs, not with the highest characters. It could also provide for simple overstriking combining characters, but then the glyphs for combining characters would have to be stored with negative width inside an ISO10646-C font. It can even provide support for variable combining accent positions, by having several alternative combining glyphs with accents at different heights for the same combining character, with the ligature substitution tables encoding which combining glyph to use with which base character.

TODO: write specification for ISO10646-C properties, write sample implementations of the mapping routines, and add these to xterm, GTK, and other applications and libraries. Any volunteers?

- **Keysyms:** The keysyms defined at the moment cover only a tiny repertoire of Unicode. Markus Kuhn has suggested (and implemented in xterm) that any UCS character in the range U-00000000 to U-00FFFFFF can be represented by a keysym value in the range 0x01000000 to 0x01ffffff. This admittedly does not cover the entire 31-bit space of UCS, but it does cover all the characters up to U-0010FFFF, which can be represented by UTF-16, and more, and it is very unlikely that higher UCS codes will ever be assigned by ISO (in fact there are proposals to remove the code space above U-0010FFFF from ISO 10646 in the future). So to get Unicode character U+ABCD you can directly use keysym 0x0100abcd. See also the file keysym2ucs.c in the xterm source code for a suggested conversion table between the classical keysyms and UCS, something which should also go into the X11 standard. Markus also wrote a proposed draft revision of the X protocol standard Appendix A: KEYSYM Encoding (PDF) that adds a UCS cross reference table. See also the X.Org wiki page on revising keysyms.
- **Combining characters:** The X11 specification does not support combining characters in any way. The font information lacks the data necessary to perform high-quality automatic accent placement (as it is found, for example, in all TeX fonts). Various people have experimented with implementing simplest overstriking combining characters using zero-width characters with ink on the left side of the origin, but details of how to do this exactly are unspecified (e.g., are zero-width characters allowed in CharCell and Monospaced fonts?) and this is therefore not yet widely established practice.
- **Ligatures:** The Indic scripts need font file formats that support ligature substitution, which is at the moment just as completely out of the scope of the X11 specification as are combining characters.
- **UTF-8 locales:** The X11 R6.4 sample implementation did not contain any support for UTF-8 locales. There is an old UTF locale, but it is incomplete and uses the now obsolete UTF-1 encoding. Implementing a UTF-8 locale not only requires the usual encoding conversion routines, but also various keyboard entry methods, ranging from mapping the existing ISO 8859 and keysym keyboards to UCS, over vastly extended support for the compose key and ISO 14755 hexadecimal entry of arbitrary characters to input entry support for Hangul and Han characters.
- **Sample implementation:** A number of comprehensive Unicode standard fonts as well as Unicode support for classic standard tools such as xterm, xfontsel, the window managers, etc. should be added to the sample implementation. Some work on this part has already been done within XFree86, other work is currently delayed by the fact that the previous points have not yet been resolved.

Several XFree86 team members are trying to work on these issues with X.Org, which is the official successor of the X Consortium and the Opengroup as the custodian of the X11 standards and the sample implementation. Things have been moving slowly. Support for UTF8_STRING, UCS keysyms, and ISO10646-1 extensions of the core fonts will hopefully make it into R6.7.1 in 2004. With regard to the other font related problems, the solution will probably be to dump the old server-side font mechanisms entirely and use instead XFree86's new Xft. Another work-in-progress is a new Standard Type Services (ST) framework that Sun has been working on and plans to donate to XFree86 and X.org very soon.

What are useful Perl one-liners for working with UTF-8?

These examples assume that you have Perl 5.8 or newer and you have activated a UTF-8 locale (i.e., "locale charmap" outputs "UTF-8").

Print the euro sign (U+20AC) to stdout:

```
  perl -e 'print pack('U',0x20ac)."\n"'
  perl -e 'print "\x{20ac}\n"'           # works only from U+0100
upwards
```

Locate malformed UTF-8 sequences:

```
  perl -ne 'use bytes;/^(([\x00-\x7f]|[\xc0-\xdf][\x80-\xbf]|[\xe0-
\xef][\x80-\xbf]{2}|[\xf0-\xf7][\x80-\xbf]{3})*)(.*)$/;print "$ARGV:
$.:".($-[3]+1).":$_" if length($3)'
```

Are there any good mailing lists on these issues?

You should certainly be on the `linux-utf8@nl.linux.org` mailing list. That's the place to meet for everyone interested in working towards better UTF-8 support for GNU/Linux or Unix systems and applications. To subscribe, send a message to linux-utf8-request@nl.linux.org with the subject `subscribe`. You can also browse the linux-utf8 archive.

There is also the `unicode@unicode.org` mailing list, which is the best way of finding out what the authors of the Unicode standard and a lot of other gurus have to say. To subscribe, send to unicode-request@unicode.org a message with the subject line "subscribe" and the text "subscribe *YOUR@EMAIL.ADDRESS* unicode".

The relevant mailing lists for discussions about Unicode support in Xlib and the X server are the fonts and i18n at xfree86.org mailing lists.

Further References

- Bruno Haible's Unicode HOWTO.
- The Unicode Standard, Version 4.0, Addison-Wesley, 2003. You definitely should have a copy of the standard if you are doing anything related to fonts and character sets.
- Ken Lunde's CJKV Information Processing, O'Reilly & Associates, 1999. This is clearly the best book available if you are interested in East Asian character sets.
- Unicode Technical Reports
- Mark Davis' Unicode FAQ
- ISO/IEC 10646-1:2000
- Frank Tang's Iñtërnâtiônàlizætiøn Secrets
- IBM's Unicode Zone
- Unicode Support in the Solaris 7 Operating Environment

- The USENIX Winter 1993 paper by Rob Pike and Ken Thompson on the introduction of UTF-8 under Plan9 reports about the experience gained when Plan9 migrated as the first operating system back in 1992 completely to UTF-8 (which was at the time still called UTF-2). A must read!
- OpenI18N is a project initiated by several Linux distributors to enhance Unicode support for free operating systems. It published the OpenI18N Globalization Specification, as well as some patches.
- The Online Single Unix Specification contains definitions of all the ISO C Amendment 1 function, plus extensions such as wcwidth().
- The Open Group's summary of ISO C Amendment 1.
- GNU libc
- The Linux Console Tools
- The Unicode Consortium character database and character set conversion tables are an essential resource for anyone developing Unicode related tools.
- Other conversion tables are available from Microsoft and Keld Simonsen's WG15 archive.
- Michael Everson's Unicode and JTC1/SC2/WG2 Archive contains online versions of many of the more recent ISO 10646-1 amendments, plus many other goodies. See also his Roadmaps to the Universal Character Set.
- An introduction into The Universal Character Set (UCS).
- Otfried Cheong's essay on Han Unification in Unicode
- The AMS STIX project revised and extended the mathematical characters for Unicode 3.2 and ISO 10646-2. They are now preparing a freely available the STIX Fonts family of fully hinted Type1 and TrueType fonts, covering the over 7700 characters needed for scientific publishing in a "Times compatible" design.
- Jukka Korpela's Soft hyphen (SHY) - a hard problem? is an excellent discussion of the controversy surrounding U+00AD.
- James Briggs' Perl, Unicode and I18N FAQ.
- Mark Davis discusses in Forms of Unicode the tradeoffs between UTF-8, UTF-16, and UCS-4 (now also called UTF-32 for political reasons). Doug Ewell wrote A survey of Unicode compression.
- Alan Wood has a good page on Unicode and Multilingual Support in Web Browsers and HTML.
- ISO/JTC1/SC22/WG20 produced various Unicode related standards such as the International String Ordering (ISO 14651) and the Cultural Convention Specification TR (ISO TR 14652) (an extension of the POSIX locale format that covers, for example, transliteration of wide character output).
- ISO/JTC1/SC2/WG2/IRG (Ideographic Rapporteur Group)
- The Letter Database answers queries on languages, character sets and names, as does the Zvon Character Search.
- Vietnamese Unicode FAQs
- China has specified in GB 18030 a new encoding of UCS for use in Chinese government systems that is backwards-compatible with the widely used GB 2312 and GBK encodings for Chinese. It seems though that the first version (released 2000-03) is somewhat buggy and will likely go through a couple more revisions, so use with care. GB 18030 is probably more of a temporary migration path to UCS and will probably not survive for long against UTF-8 or UTF-16, even in

Chinese government systems.

- Hong Kong Supplementary Character Set (HKSCS)
- Various people propose UCS alternatives: Rosetta, Bytext.
- Proceedings of the International Unicode Conferences: ICU13, ICU14, ICU15, ICU16, ICU17, ICU18, etc.
- This FAQ has been translated into other languages:
    - Korean: 2001-02

  Be aware that each translation reflects only some past version of this document, which I update several times per month and revise more thorougly once or twice each year.

I add new material to this document quite frequently, so please come back from time to time. Suggestions for improvement are very welcome. Please help to spread the word in the free software community about the importance of UTF-8.

Special thanks to Ulrich Drepper, Bruno Haible, Robert Brady, Juliusz Chroboczek, Shuhei Amakawa, Jungshik Shi, Robert Rogers and many others for valuable comments, and to SuSE GmbH, Nürnberg, for their support.

Markus Kuhn

created 1999-06-04 -- last modified 2004-04-23 -- http://www.cl.cam.ac.uk/~mgk25/unicode.html

SC UniPad The Unicode Text Editor

What is SC UniPad?

SC UniPad is a Unicode™ plain text editor for the Windows NT®, Windows 2000®, Windows 9x®, Windows ME® and Windows XP® operating systems.

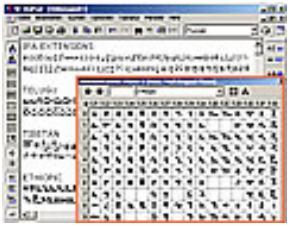
+++ Displays about 53000 Unicode characters instantly without installing extra fonts + On-screen soft keyboard + Over 60 built-in keyboard layouts + Character map for easy selection of any Unicode character + Import / export of over 60 codepages, encodings + Unicode formats UTF-8, UTF-16, UTF-32, UTF-7, Compression Scheme, \u + supports Unicode standard 4.01 + and more... +++

Download **your free version** now.

Select a picture to get more information about:


Editor


Keyboard Layout


Character Map

:: Register UniPad 1.10 ::

:: 2002-10-01
SC UniPad 1.0 available.
Read the press release.

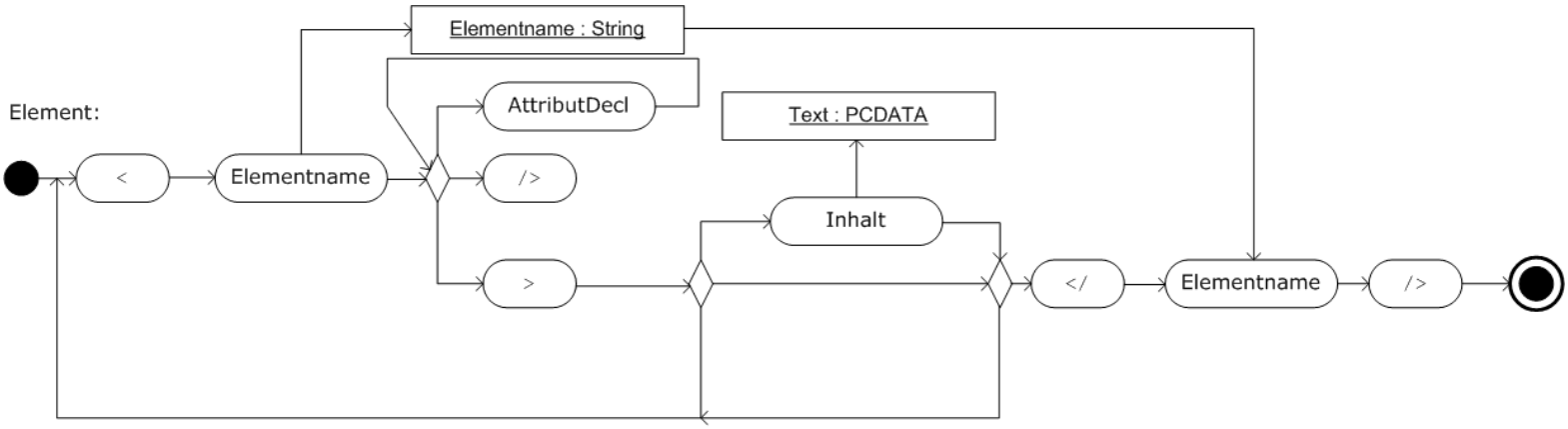
:: Latest UniPad version:
Free download!

:: Download additional keyboards!

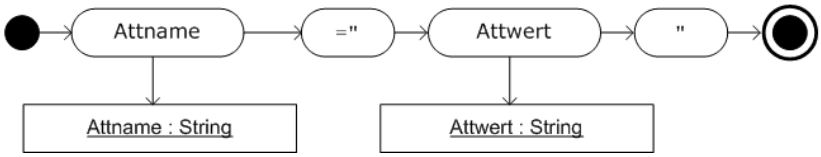
:: Supports more than 300 languages.

:: Supports over 53000 unicode characters.

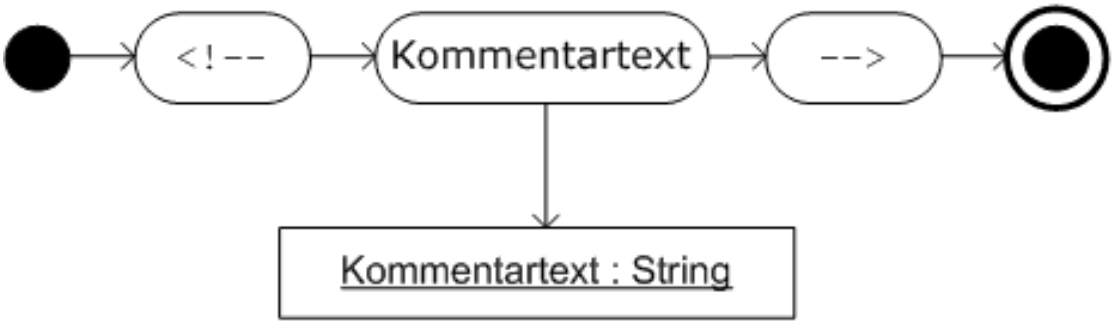
:: Who has downloaded UniPad so far? Excerpt from the Download Logfiles


Elementname : String
Element:
AttributDecl
Text : PCDATA
<
Elementname
/>
Inhalt
>
</
Elementname
/>

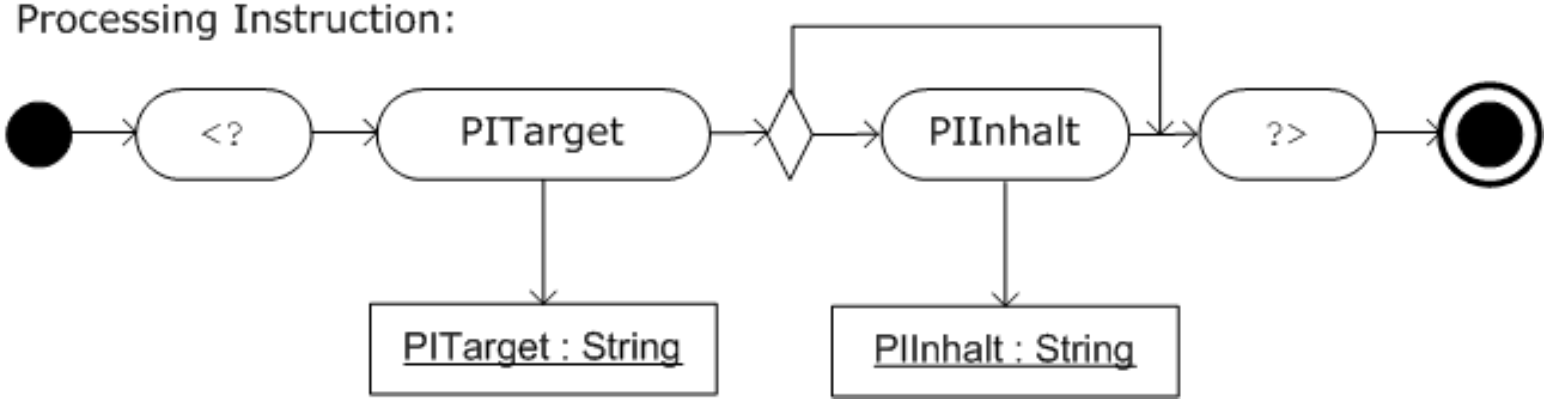
Attribut:

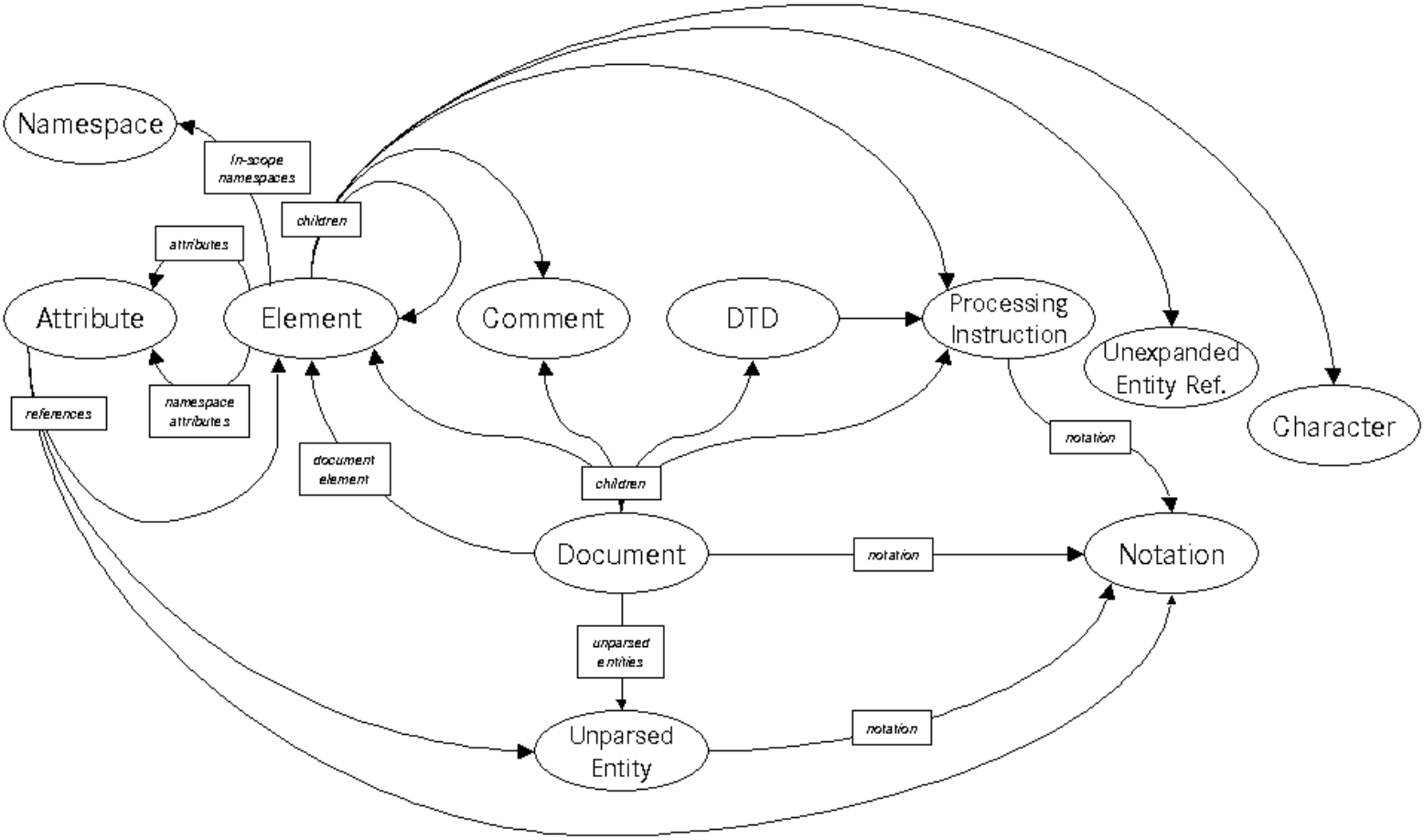

Attname
="
Attwert
"
Attname : String
Attwert : String

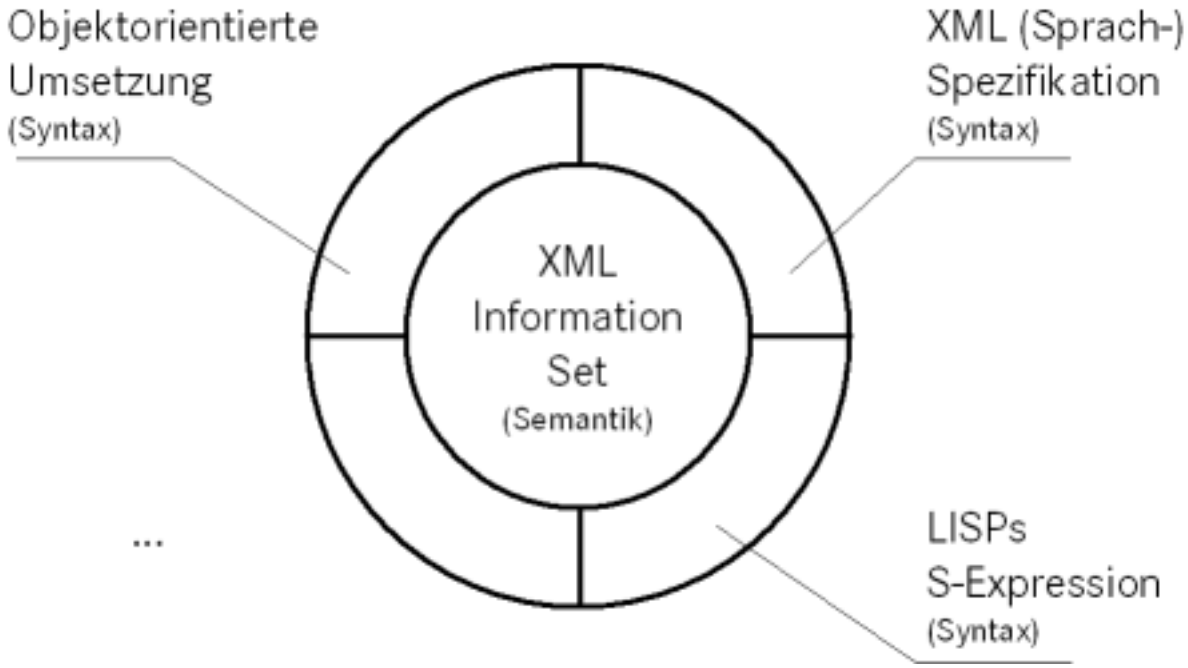
Kommentar:

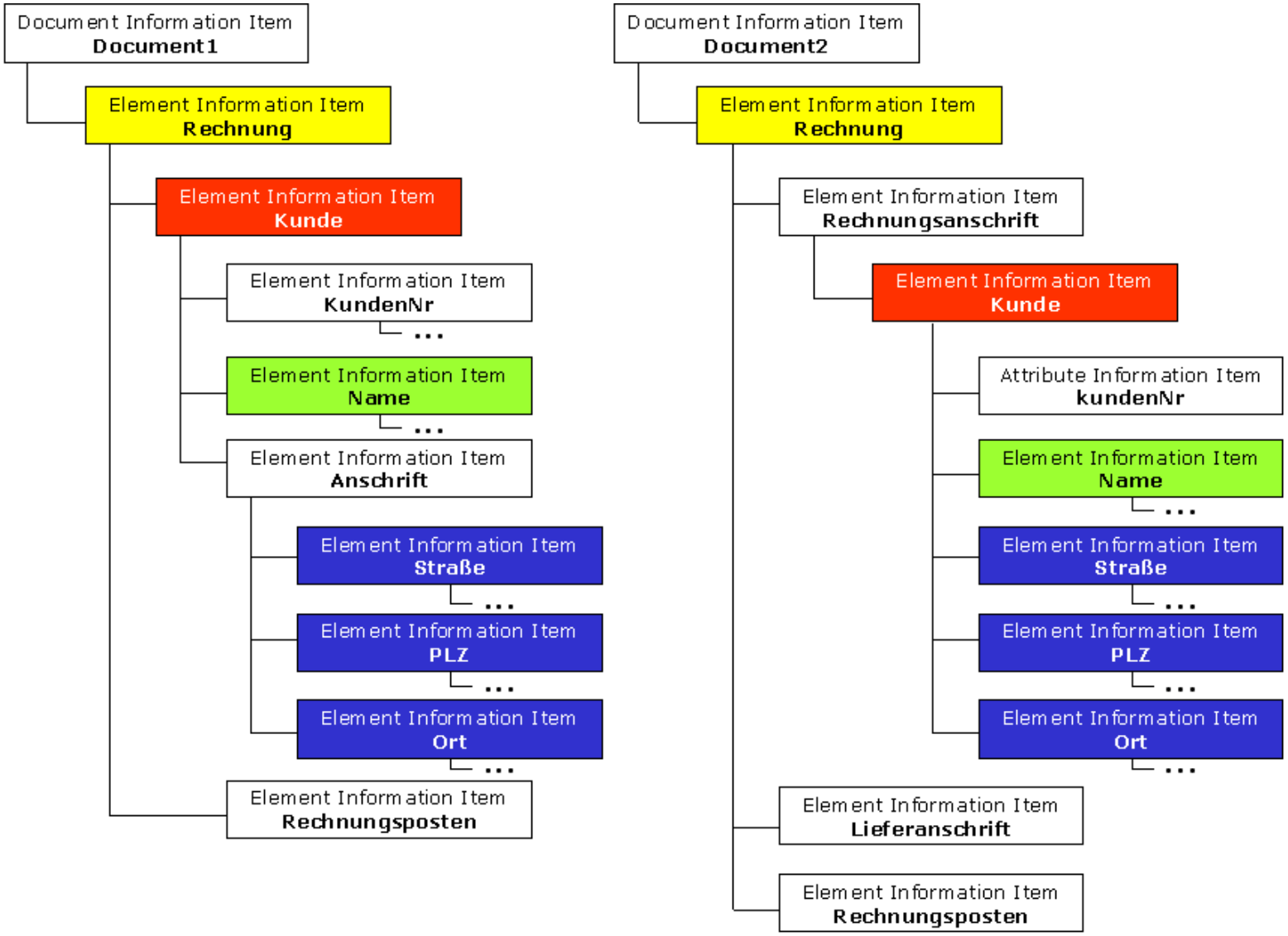


Processing Instruction:




Namespace
in-scope namespaces
children
attributes
Attribute
Element
Comment
DTD
Processing Instruction
Unexpanded Entity Ref.
Character
references
namespace attributes
document element
children
notation
Document
notation
Notation
unparsed entities
Unparsed Entity
notation


Objektorientierte
Umsetzung
(Syntax)
XML (Sprach-)
Spezifikation
(Syntax)
XML
Information
Set
(Semantik)
...
LISPs
S-Expression
(Syntax)


Document Information Item
Document1
Element Information Item
Rechnung
Element Information Item
Kunde
Element Information Item
KundenNr
...
Element Information Item
Name
...
Element Information Item
Anschrift
Element Information Item
Straße
...
Element Information Item
PLZ
...
Element Information Item
Ort
...
Element Information Item
Rechnungsposten
Document Information Item
Document2
Element Information Item
Rechnung
Element Information Item
Rechnungsanschrift
Element Information Item
Kunde
Attribute Information Item
kundenNr
Element Information Item
Name
...
Element Information Item
Straße
...
Element Information Item
PLZ
...
Element Information Item
Ort
...
Element Information Item
Lieferanschrift
Element Information Item
Rechnungsposten

Mathematical Markup Language (MathML) Version 2.0 (Second Edition)

W3C Recommendation 21 October 2003

This version:
http://www.w3.org/TR/2003/REC-MathML2-20031021/
Latest MathML 2 version:
http://www.w3.org/TR/MathML2/
Latest MathML Recommendation:
http://www.w3.org/TR/MathML/
Previous version:
http://www.w3.org/TR/2003/PER-MathML2-20030804/
Editors:
David Carlisle, NAG
Patrick Ion, Mathematical Reviews, American Mathematical Society
Robert Miner, Design Science, Inc.
Nico Poppelier, Penta Scope
Principal Authors:
Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, Stephen Watt

Please refer to the **errata** for this document, which may include some normative corrections.

In addition to the HTML version, this document is also available in these non-normative formats: diff marked HTML version, XHTML+MathML version, PDF (paper formatted) version, PDF (screen formatted) version, zip archive of XML sources and stylesheets, and zip archive of (X)HTML documents.

See also **translations**.

Abstract

This specification defines the Mathematical Markup Language, or MathML. MathML is an XML application for describing mathematical notation and capturing both its structure and content. The goal of MathML is to enable mathematics to be served, received, and processed on the World Wide Web, just as HTML has enabled this functionality for text.

This specification of the markup language MathML is intended primarily for a readership consisting of those who will be developing or implementing renderers or editors using it, or software that will communicate using MathML as a protocol for input or output. It is *not* a User's Guide but rather a reference document.

This document begins with background information on mathematical notation, the problems it poses, and the philosophy underlying the solutions MathML 2.0 proposes. MathML can be used to encode both mathematical notation and mathematical content. About thirty of the MathML tags describe abstract notational structures, while another about one hundred and fifty provide a way of unambiguously specifying the intended meaning of an expression. Additional chapters discuss how the MathML content and presentation elements interact, and how MathML renderers might be implemented and should interact with browsers. Finally, this document addresses the issue of MathML characters and their relation to fonts.

While MathML is human-readable, it is anticipated that, in all but the simplest cases, authors will use equation editors, conversion programs, and other specialized software tools to generate MathML. Several versions of such MathML tools already exist, and a number of others, both freely available software and commercial products, are under development.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is a revised edition of a document that has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation.

This document has been produced by the W3C Math Working Group as part of W3C Math Activity. The goals of the W3C Math Working Group are discussed in the W3C Math WG Charter (revised February 2000 and June 2001 from the original of 11 June 1998). A list of participants in the W3C Math Working Group is available.

The MathML 2.0 (Second Edition) specification was reviewed extensively during its development, as provided by the W3C Process. During that period the W3C Math Working Group participants encouraged implementation using the specification and comment on it; a report on Implementation and Interoperability experiences and issues has been made public. It is intended that this will be updated from time to time by the continuing work of the W3C that oversees the MathML 2.0 Recommendation. The W3C Math Activity maintains a public Web page http://www.w3.org/Math/ which contains further background information.

The preparation of a Second Edition of the MathML 2.0 Specification allows the revision of that document to provide a coherent whole containing corrections to all the known errata and clarifications of some smaller issues that proved problematic. It is not the occasion for any fundamental changes in the language MathML 2.0. To clarify this a diff-marked HTML version of this Second Edition is made available.

Public discussion of MathML and issues of support through the W3C for mathematics on the Web takes place on the public mailing list of the Math Working Group (list archives). To subscribe send an email to www-math-request@w3.org with the word `subscribe` in the subject line.

Please report errors in this document to www-math@w3.org.

Patent disclosures relevant to this specification may be found on the Math Working Group's patent disclosure page.

The basic structure of this document is the same as that of the earlier MathML 2.0 Recommendation [MathML2], with the addition of an index in the new Appendix L. MathML 2.0 itself was a revision of the earlier W3C Recommendation MathML 1.01 [MathML1]. It differed from it in that all chapters were updated and two new ones and some appendices were added.

Since MathML 1.01, Chapters 1 and 2, which are introductory material, have been revised to reflect the changes elsewhere in the document, and in the rapidly evolving Web environment. Chapters 3 and 4 have been extended to describe new functionalities added as well as smaller improvements of material already proposed. Chapter 5 has been newly written to reflect changes in the technology available. The major tables in Chapter 6 have been regenerated and reorganized to reflect an improved list of characters useful for mathematics, and the text revised to reflect the new situation in regard to Unicode. Chapter 7 has been completely revised since Web technology has changed. A new Chapter 8 on the DOM for MathML 2.0 has been added; the latter points to new appendices D and E for detailed listings.

The appendices have been reorganized into normative and non-normative groups. The material in Appendices D, E, G and L was not present in MathML 1.01.

Table of Contents

Appendices

Scalable Vector Graphics (SVG) 1.1 Specification

W3C Recommendation *14 January 2003*

This version:
http://www.w3.org/TR/2003/REC-SVG11-20030114/
Latest version:
http://www.w3.org/TR/SVG11/
Previous version:
http://www.w3.org/TR/2002/PR-SVG11-20021115/
Editors:
Jon Ferraiolo, Adobe Systems <jon.ferraiolo@adobe.com> (version 1.0)
藤沢 淳 (FUJISAWA Jun), Canon <fujisawa.jun@canon.co.jp> (modularization and DTD)
Dean Jackson, W3C/CSIRO <dean@w3.org> (version 1.1)
Authors:
See author list

Please refer to the **errata** for this document, which may include some normative corrections.

This document is also available in these non-normative packages: zip archive of HTML (without external dependencies) and PDF.

See also the translations of this document.

Abstract

This specification defines the features and syntax for Scalable Vector Graphics (SVG) Version 1.1, a modularized language for describing two-dimensional vector and mixed vector/raster graphics in XML.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document is the 14 January 2003 Recommendation of the SVG 1.1 specification. SVG 1.1 serves two purposes: to provide a modularization of SVG based on SVG 1.0 and to include the errata found so far in SVG 1.0. The SVG Working Group believes SVG 1.1 has been widely reviewed by the community, developers and other W3C groups. The list of changes made in this version of the document is available.

Public comments on this Recommendation are welcome. Please send them to www-svg@w3.org: the public email list for issues related to vector graphics on the Web. This list is archived and senders must agree to have their message publicly archived from their first posting. To subscribe send an email to www-svg-request@w3.org with the word `subscribe` in the subject line.

The W3C SVG Working Group have released a test suite for SVG 1.1 along with an implementation report.

The latest information regarding patent disclosures related to this document is available on the Web. As of this publication, the SVG Working Group are not aware of any royalty-bearing patents they believe to be essential to SVG.

This document has been produced by the W3C SVG Working Group as part of the Graphics Activity within the W3C Interaction Domain. The goals of the W3C SVG Working Group are discussed in the W3C SVG Charter (W3C Members only). The W3C SVG Working Group maintains a public Web page, http://www.w3.org/Graphics/SVG/, that contains further background information. The authors of this document are the SVG Working Group participants.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/. W3C publications may be updated, replaced, or obsoleted by other documents at any time.

Available languages

The English version of this specification is the only normative version. However, for translations in other languages see http://www.w3.org/Graphics/SVG/svg-updates/translations.html.

Table of Contents

The authors of the SVG 1.1 specification are the people who participated in the SVG Working Group as members or alternates.

Authors:

- Ola Andersson, ZOOMON AB
- Phil Armstrong, Corel Corporation
- Henric Axelsson, Ericsson AB
- Robin Berjon, Expway
- Benoît Bézaire, Corel Corporation
- John Bowler, Microsoft Corporation
- Craig Brown, Canon Information Systems Research Australia
- Mike Bultrowicz, Savage Software
- Tolga Capin, Nokia
- Milt Capsimalis, Autodesk Inc.
- Mathias Larsson Carlander, Ericsson AB
- Jakob Cederquist, ZOOMON AB
- Charilaos Christopoulos, Ericsson AB
- Richard Cohn, Adobe Systems Inc.
- Lee Cole, Quark
- Don Cone, America Online Inc.
- Alex Danilo, Canon Information Systems Research Australia
- Thomas DeWeese, Eastman Kodak
- David Dodds, Lexica
- Andrew Donoho, IBM
- David Duce, Oxford Brookes University
- Jerry Evans, Sun Microsystems
- Jon Ferraiolo, Adobe Systems Inc.
- Darryl Fuller, Schema Software
- 藤沢 淳 (FUJISAWA Jun), Canon
- Scott Furman, Netscape Communications Corporation

- Brent Getlin, Macromedia
- Peter Graffagnino, Apple
- Rick Graham, BitFlash
- Vincent Hardy, Sun Microsystems Inc.
- 端山 貴也 (HAYAMA Takanari), KDDI Research Labs
- Lofton Henderson, OASIS
- Jan Christian Herlitz, Excosoft
- Alan Hester, Xerox Corporation
- Bob Hopgood, RAL (CCLRC)
- 石川 雅康 (ISHIKAWA Masayasu), W3C
- Dean Jackson, W3C/CSIRO (*W3C Team Contact*)
- Christophe Jolif, ILOG S.A.
- Lee Klosterman, Hewlett-Packard
- 小林 亜令 (KOBAYASHI Arei), KDDI Research Labs
- Thierry Kormann, ILOG S.A.
- Yuri Khramov, Schema Software
- Kelvin Lawrence, IBM
- Håkon Lie, Opera
- Chris Lilley, W3C (*Working Group Chair*)
- Philip Mansfield, Schema Software
- Kevin McCluskey, Netscape Communications Corporation
- 水口 充 (MINAKUCHI Mitsuru), Sharp Corporation
- Luc Minnebo, Agfa-Gevaert N.V.
- Tuan Nguyen, Microsoft Corporation
- 小野 修一郎 (ONO Shuichiro), Sharp Corporation
- Antoine Quint, Fuchsia Design (formerly of ILOG)
- 相良 毅 (SAGARA Takeshi), KDDI Research Labs
- Troy Sandal, Visio Corporation
- Peter Santangeli, Macromedia
- Haroon Sheikh, Corel Corporation
- Brad Sipes, ZOOMON AB
- Peter Sorotokin, Adobe Systems Inc.
- Gavriel State, Corel Corporation
- Robert Stevahn, Hewlett-Packard
- Timothy Thompson, Eastman Kodak
- 上田 宏高 (UEDA Hirotaka), Sharp Corporation
- Rick Yardumian, Canon Development Americas
- Charles Ying, Openwave Systems Inc.
- Shenxue Zhou, Quark

Acknowledgments

The SVG Working Group would like to acknowledge the great many people outside of the SVG Working Group who help with the process of developing the SVG 1.1 specification. These people are too numerous to list individually. They include but are not limited to the early implementers of the SVG 1.0 and

1.1 languages (including viewers, authoring tools, and server-side transcoders), developers of SVG content, people who have contributed on the www-svg@w3.org and svg-developers@yahoogroups.com email lists, other Working Groups at the W3C, and the W3C Team. SVG 1.1 is truly a cooperative effort between the SVG Working Group, the rest of the W3C, and the public and benefits greatly from the pioneering work of early implementers and content developers, feedback from the public, and help from the W3C team.

---

previous next contents elements attributes properties index

Network Working Group
Request for Comments: 2732
Category: Standards Track

R. Hinden
Nokia
B. Carpenter
IBM
L. Masinter
AT&T

December 1999

Format for Literal IPv6 Addresses in URL's

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document defines the format for literal IPv6 Addresses in URL's for implementation in World Wide Web browsers. This format has been implemented in the IPv6 versions of several widely deployed browsers including Microsoft Internet Explorer, Mozilla, and Lynx. It is also intended to be used in the IPv6 version of the service location protocol.

This document incudes an update to the generic syntax for Uniform Resource Identifiers defined in RFC 2396 [URL]. It defines a syntax for IPv6 addresses and allows the use of "[" and "]" within a URI explicitly for this reserved purpose.

1. Introduction

The textual representation defined for literal IPv6 addresses in

[ARCH] is not directly compatible with URL's. Both use ":" and "." characters as delimiters. This document defines the format for literal IPv6 Addresses in URL's for implementation in World Wide Web browsers. The goal is to have a format that allows easy "cut" and "paste" operations with a minimum of editing of the literal address.

The format defined in this document has been implemented in the IPv6 versions of several widely deployed browsers including Microsoft Internet Explorer, Mozilla, and Lynx. It is also intended to be used in the IPv6 version of the service location protocol.

1.1 Requirements

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, if and where they appear in this document, are to be interpreted as described in [KEYWORDS].

World Wide Web browsers SHOULD implement the format of IPv6 literals in URL's defined in this document. Other types of applications and protocols that use URL's MAY use this format.

2. Literal IPv6 Address Format in URL's Syntax

To use a literal IPv6 address in a URL, the literal address should be enclosed in "[" and "]" characters. For example the following literal IPv6 addresses:

```
FEDC:BA98:7654:3210:FEDC:BA98:7654:3210
1080:0:0:0:8:800:200C:4171
3ffe:2a00:100:7031::1
1080::8:800:200C:417A
::192.9.5.5
::FFFF:129.144.52.38
2010:836B:4179::836B:4179
```

would be represented as in the following example URLs:

http://[FEDC:BA98:7654:3210:FEDC:BA98:7654:3210]:80/index.html
http://[1080:0:0:0:8:800:200C:417A]/index.html
http://[3ffe:2a00:100:7031::1]
http://[1080::8:800:200C:417A]/foo
http://[::192.9.5.5]/ipng
http://[::FFFF:129.144.52.38]:80/index.html
http://[2010:836B:4179::836B:4179]

3. Changes to RFC 2396

This document updates the generic syntax for Uniform Resource Identifiers defined in RFC 2396 [URL]. It defines a syntax for IPv6 addresses and allows the use of "[" and "]" within a URI explicitly for this reserved purpose.

The following changes to the syntax in RFC 2396 are made:
(1) change the 'host' non-terminal to add an IPv6 option:

```
host          = hostname | IPv4address | IPv6reference
ipv6reference = "[" IPv6address "]"
```

where IPv6address is defined as in RFC2373 [ARCH].

(2) Replace the definition of 'IPv4address' with that of RFC 2373, as it correctly defines an IPv4address as consisting of at most three decimal digits per segment.

(3) Add "[" and "]" to the set of 'reserved' characters:

```
reserved    = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
              "$" | "," | "[" | "]"
```

and remove them from the 'unwise' set:

```
unwise      = "{" | "}" | "|" | "\" | "^" | "`"
```

4. Security Considerations

The use of this approach to represent literal IPv6 addresses in URL's does not introduce any known new security concerns.

5. IANA Considerations

None.

6. Authors' Addresses

Robert M. Hinden
Nokia
313 Fairchild Drive
Mountain View, CA 94043
USA

Phone: +1 650 625 2004
EMail: hinden@iprg.nokia.com

Web: http://www.iprg.nokia.com/~hinden

Brian E. Carpenter
IBM
iCAIR, Suite 150
1890 Maple Avenue
Evanston IL 60201
USA

EMail: brian@icair.org

Larry Masinter
AT&T Labs
75 Willow Road
Menlo Park, CA 94025

EMail: LMM@acm.org
Web: http://larry.masinter.net

7. References

[ARCH] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[STD-PROC] Bradner, S., The Internet Standards Process -- Revision 3, BCP 9, RFC 2026, October 1996.

[URL] Fielding, R., Masinter, L. and T. Berners-Lee, "Uniform Resource Identifiers: Generic Syntax", RFC 2396, August 1998.

8. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

Network Working Group T. Berners-Lee
Request for Comments: 1738 CERN
Category: Standards Track L. Masinter
Xerox Corporation
M. McCahill
University of Minnesota
Editors
December 1994

Uniform Resource Locators (URL)

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document specifies a Uniform Resource Locator (URL), the syntax and semantics of formalized information for location and access of resources via the Internet.

1. Introduction

This document describes the syntax and semantics for a compact string representation for a resource available via the Internet. These strings are called "Uniform Resource Locators" (URLs).

The specification is derived from concepts introduced by the World-Wide Web global information initiative, whose use of such objects dates from 1990 and is described in "Universal Resource Identifiers in WWW", RFC 1630. The specification of URLs is designed to meet the requirements laid out in "Functional Requirements for Internet Resource Locators" [12].

This document was written by the URI working group of the Internet

Engineering Task Force. Comments may be addressed to the editors, or to the URI-WG <uri@bunyip.com>. Discussions of the group are archived at <URL:http://www.acl.lanl.gov/URI/archive/uri-archive.index.html>

2. General URL Syntax

Just as there are many different methods of access to resources, there are several schemes for describing the location of such resources.

The generic syntax for URLs provides a framework for new schemes to be established using protocols other than those defined in this document.

URLs are used to `locate' resources, by providing an abstract identification of the resource location. Having located a resource, a system may perform a variety of operations on the resource, as might be characterized by such words as `access', `update', `replace', `find attributes'. In general, only the `access' method needs to be specified for any URL scheme.

2.1. The main parts of URLs

A full BNF description of the URL syntax is given in Section 5.

In general, URLs are written as follows:

<scheme>:<scheme-specific-part>

A URL contains the name of the scheme being used (<scheme>) followed by a colon and then a string (the <scheme-specific-part>) whose interpretation depends on the scheme.

Scheme names consist of a sequence of characters. The lower case

letters "a"--"z", digits, and the characters plus ("+"), period (".") , and hyphen ("-") are allowed. For resiliency, programs interpreting URLs should treat upper case letters as equivalent to lower case in scheme names (e.g., allow "HTTP" as well as "http").

2.2. URL Character Encoding Issues

URLs are sequences of characters, i.e., letters, digits, and special characters. A URLs may be represented in a variety of ways: e.g., ink on paper, or a sequence of octets in a coded character set. The interpretation of a URL depends only on the identity of the characters used.

In most URL schemes, the sequences of characters in different parts of a URL are used to represent sequences of octets used in Internet protocols. For example, in the ftp scheme, the host name, directory name and file names are such sequences of octets, represented by parts of the URL. Within those parts, an octet may be represented by

the chararacter which has that octet as its code within the US-ASCII [20] coded character set.

In addition, octets may be encoded by a character triplet consisting of the character "%" followed by the two hexadecimal digits (from "0123456789ABCDEF") which forming the hexadecimal value of the octet. (The characters "abcdef" may also be used in hexadecimal encodings.)

Octets must be encoded if they have no corresponding graphic character within the US-ASCII coded character set, if the use of the corresponding character is unsafe, or if the corresponding character is reserved for some other interpretation within the particular URL scheme.

No corresponding graphic US-ASCII:

URLs are written only with the graphic printable characters of the US-ASCII coded character set. The octets 80-FF hexadecimal are not used in US-ASCII, and the octets 00-1F and 7F hexadecimal represent control characters; these must be encoded.

Unsafe:

Characters can be unsafe for a number of reasons. The space character is unsafe because significant spaces may disappear and insignificant spaces may be introduced when URLs are transcribed or typeset or subjected to the treatment of word-processing programs. The characters "<" and ">" are unsafe because they are used as the delimiters around URLs in free text; the quote mark (""") is used to delimit URLs in some systems. The character "#" is unsafe and should always be encoded because it is used in World Wide Web and in other systems to delimit a URL from a fragment/anchor identifier that might follow it. The character "%" is unsafe because it is used for encodings of other characters. Other characters are unsafe because gateways and other transport agents are known to sometimes modify such characters. These characters are "{", "}", "|", "\", "^", "~", "[", "]", and "`".

All unsafe characters must always be encoded within a URL. For example, the character "#" must be encoded within URLs even in systems that do not normally deal with fragment or anchor identifiers, so that if the URL is copied into another system that does use them, it will not be necessary to change the URL encoding.

Reserved:

Many URL schemes reserve certain characters for a special meaning: their appearance in the scheme-specific part of the URL has a designated semantics. If the character corresponding to an octet is reserved in a scheme, the octet must be encoded. The characters ";", "/", "?", ":", "@", "=" and "&" are the characters which may be reserved for special meaning within a scheme. No other characters may be reserved within a scheme.

Usually a URL has the same interpretation when an octet is represented by a character and when it encoded. However, this is not true for reserved characters: encoding a character reserved for a particular scheme may change the semantics of a URL.

Thus, only alphanumerics, the special characters "$-_.+!*'(),", and reserved characters used for their reserved purposes may be used unencoded within a URL.

On the other hand, characters that are not required to be encoded (including alphanumerics) may be encoded within the scheme-specific part of a URL, as long as they are not being used for a reserved purpose.

2.3 Hierarchical schemes and relative links

In some cases, URLs are used to locate resources that contain pointers to other resources. In some cases, those pointers are represented as relative links where the expression of the location of the second resource is in terms of "in the same place as this one except with the following relative path". Relative links are not described in this document. However, the use of relative links depends on the original URL containing a hierarchical structure against which the relative link is based.

Some URL schemes (such as the ftp, http, and file schemes) contain names that can be considered hierarchical; the components of the hierarchy are separated by "/".

3. Specific Schemes

The mapping for some existing standard and experimental protocols is outlined in the BNF syntax definition. Notes on particular protocols follow. The schemes covered are:

ftp	File Transfer protocol
http	Hypertext Transfer Protocol
gopher	The Gopher protocol
mailto	Electronic mail address
news	USENET news
nntp	USENET news using NNTP access
telnet	Reference to interactive sessions
wais	Wide Area Information Servers
file	Host-specific file names
prospero	Prospero Directory Service

Other schemes may be specified by future specifications. Section 4 of this document describes how new schemes may be registered, and lists some scheme names that are under development.

3.1. Common Internet Scheme Syntax

While the syntax for the rest of the URL may vary depending on the particular scheme selected, URL schemes that involve the direct use of an IP-based protocol to a specified host on the Internet use a common syntax for the scheme-specific data:

```
//<user>:<password>@<host>:<port>/<url-path>
```

Some or all of the parts "<user>:<password>@", ":<password>", ":<port>", and "/<url-path>" may be excluded. The scheme specific data start with a double slash "//" to indicate that it complies with the common Internet scheme syntax. The different components obey the following rules:

user
An optional user name. Some schemes (e.g., ftp) allow the specification of a user name.

password
An optional password. If present, it follows the user name separated from it by a colon.

The user name (and password), if present, are followed by a commercial at-sign "@". Within the user and password field, any ":",

"@", or "/" must be encoded.

Note that an empty user name or password is different than no user name or password; there is no way to specify a password without specifying a user name. E.g., <URL:ftp://@host.com/> has an empty user name and no password, <URL:ftp://host.com/> has no user name, while <URL:ftp://foo:@host.com/> has a user name of "foo" and an empty password.

host
The fully qualified domain name of a network host, or its IP address as a set of four decimal digit groups separated by ".". Fully qualified domain names take the form as described in Section 3.5 of RFC 1034 [13] and Section 2.1 of RFC 1123 [5]: a sequence of domain labels separated by ".", each domain label starting and ending with an alphanumerical character and possibly also containing "-" characters. The rightmost domain label will never start with a digit, though, which syntactically distinguishes all domain names from the IP addresses.

port
The port number to connect to. Most schemes designate protocols that have a default port number. Another port number may optionally be supplied, in decimal, separated from the host by a colon. If the port is omitted, the colon is as well.

url-path
The rest of the locator consists of data specific to the scheme, and is known as the "url-path". It supplies the details of how the specified resource can be accessed. Note that the "/" between the host (or port) and the url-path is NOT part of the url-path.

The url-path syntax depends on the scheme being used, as does the manner in which it is interpreted.

3.2. FTP

The FTP URL scheme is used to designate files and directories on Internet hosts accessible using the FTP protocol (RFC959).

A FTP URL follow the syntax described in Section 3.1. If :<port> is omitted, the port defaults to 21.

3.2.1. FTP Name and Password

A user name and password may be supplied; they are used in the ftp "USER" and "PASS" commands after first making the connection to the FTP server. If no user name or password is supplied and one is requested by the FTP server, the conventions for "anonymous" FTP are to be used, as follows:

> The user name "anonymous" is supplied.
>
> The password is supplied as the Internet e-mail address of the end user accessing the resource.

If the URL supplies a user name but no password, and the remote server requests a password, the program interpreting the FTP URL should request one from the user.

3.2.2. FTP url-path

The url-path of a FTP URL has the following syntax:

> <cwd1>/<cwd2>/.../<cwdN>/<name>;type=<typecode>

Where <cwd1> through <cwdN> and <name> are (possibly encoded) strings and <typecode> is one of the characters "a", "i", or "d". The part ";type=<typecode>" may be omitted. The <cwdx> and <name> parts may be

empty. The whole url-path may be omitted, including the "/" delimiting it from the prefix containing user, password, host, and port.

The url-path is interpreted as a series of FTP commands as follows:

Each of the <cwd> elements is to be supplied, sequentially, as the argument to a CWD (change working directory) command.

If the typecode is "d", perform a NLST (name list) command with <name> as the argument, and interpret the results as a file directory listing.

Otherwise, perform a TYPE command with <typecode> as the argument, and then access the file whose name is <name> (for example, using the RETR command.)

Within a name or CWD component, the characters "/" and ";" are reserved and must be encoded. The components are decoded prior to their use in the FTP protocol. In particular, if the appropriate FTP sequence to access a particular file requires supplying a string containing a "/" as an argument to a CWD or RETR command, it is

necessary to encode each "/".

For example, the URL <URL:ftp://myname@host.dom/%2Fetc/motd> is interpreted by FTP-ing to "host.dom", logging in as "myname" (prompting for a password if it is asked for), and then executing "CWD /etc" and then "RETR motd". This has a different meaning from <URL:ftp://myname@host.dom/etc/motd> which would "CWD etc" and then "RETR motd"; the initial "CWD" might be executed relative to the default directory for "myname". On the other hand, <URL:ftp://myname@host.dom//etc/motd>, would "CWD " with a null argument, then "CWD etc", and then "RETR motd".

FTP URLs may also be used for other operations; for example, it is possible to update a file on a remote file server, or infer information about it from the directory listings. The mechanism for doing so is not spelled out here.

3.2.3. FTP Typecode is Optional

The entire ;type=<typecode> part of a FTP URL is optional. If it is omitted, the client program interpreting the URL must guess the appropriate mode to use. In general, the data content type of a file can only be guessed from the name, e.g., from the suffix of the name; the appropriate type code to be used for transfer of the file can then be deduced from the data content of the file.

3.2.4 Hierarchy

For some file systems, the "/" used to denote the hierarchical structure of the URL corresponds to the delimiter used to construct a file name hierarchy, and thus, the filename will look similar to the URL path. This does NOT mean that the URL is a Unix filename.

3.2.5. Optimization

Clients accessing resources via FTP may employ additional heuristics to optimize the interaction. For some FTP servers, for example, it may be reasonable to keep the control connection open while accessing multiple URLs from the same server. However, there is no common hierarchical model to the FTP protocol, so if a directory change command has been given, it is impossible in general to deduce what sequence should be given to navigate to another directory for a second retrieval, if the paths are different. The only reliable algorithm is to disconnect and reestablish the control connection.

3.3. HTTP

The HTTP URL scheme is used to designate Internet resources accessible using HTTP (HyperText Transfer Protocol).

The HTTP protocol is specified elsewhere. This specification only

describes the syntax of HTTP URLs.

An HTTP URL takes the form:

http://<host>:<port>/<path>?<searchpart>

where <host> and <port> are as described in Section 3.1. If :<port> is omitted, the port defaults to 80. No user name or password is allowed. <path> is an HTTP selector, and <searchpart> is a query string. The <path> is optional, as is the <searchpart> and its preceding "?". If neither <path> nor <searchpart> is present, the "/" may also be omitted.

Within the <path> and <searchpart> components, "/", ";", "?" are reserved. The "/" character may be used within HTTP to designate a hierarchical structure.

3.4. GOPHER

The Gopher URL scheme is used to designate Internet resources accessible using the Gopher protocol.

The base Gopher protocol is described in RFC 1436 and supports items and collections of items (directories). The Gopher+ protocol is a set of upward compatible extensions to the base Gopher protocol and is described in [2]. Gopher+ supports associating arbitrary sets of attributes and alternate data representations with Gopher items. Gopher URLs accommodate both Gopher and Gopher+ items and item attributes.

3.4.1. Gopher URL syntax

A Gopher URL takes the form:

gopher://<host>:<port>/<gopher-path>

where <gopher-path> is one of

<gophertype><selector>
<gophertype><selector>%09<search>
<gophertype><selector>%09<search>%09<gopher+_string>

If :<port> is omitted, the port defaults to 70. <gophertype> is a single-character field to denote the Gopher type of the resource to which the URL refers. The entire <gopher-path> may also be empty, in which case the delimiting "/" is also optional and the <gophertype> defaults to "1".

<selector> is the Gopher selector string. In the Gopher protocol, Gopher selector strings are a sequence of octets which may contain any octets except 09 hexadecimal (US-ASCII HT or tab) 0A hexadecimal (US-ASCII character LF), and 0D (US-ASCII character CR).

Gopher clients specify which item to retrieve by sending the Gopher selector string to a Gopher server.

Within the <gopher-path>, no characters are reserved.

Note that some Gopher <selector> strings begin with a copy of the <gophertype> character, in which case that character will occur twice consecutively. The Gopher selector string may be an empty string; this is how Gopher clients refer to the top-level directory on a Gopher server.

3.4.2 Specifying URLs for Gopher Search Engines

If the URL refers to a search to be submitted to a Gopher search engine, the selector is followed by an encoded tab (%09) and the search string. To submit a search to a Gopher search engine, the Gopher client sends the <selector> string (after decoding), a tab, and the search string to the Gopher server.

3.4.3 URL syntax for Gopher+ items

URLs for Gopher+ items have a second encoded tab (%09) and a Gopher+ string. Note that in this case, the %09<search> string must be supplied, although the <search> element may be the empty string.

The <gopher+_string> is used to represent information required for retrieval of the Gopher+ item. Gopher+ items may have alternate views, arbitrary sets of attributes, and may have electronic forms associated with them.

To retrieve the data associated with a Gopher+ URL, a client will

connect to the server and send the Gopher selector, followed by a tab and the search string (which may be empty), followed by a tab and the Gopher+ commands.

3.4.4 Default Gopher+ data representation

When a Gopher server returns a directory listing to a client, the Gopher+ items are tagged with either a "+" (denoting Gopher+ items) or a "?" (denoting Gopher+ items which have a +ASK form associated with them). A Gopher URL with a Gopher+ string consisting of only a "+" refers to the default view (data representation) of the item while a Gopher+ string containing only a "?" refer to an item with a Gopher electronic form associated with it.

3.4.5 Gopher+ items with electronic forms

Gopher+ items which have a +ASK associated with them (i.e. Gopher+ items tagged with a "?") require the client to fetch the item's +ASK attribute to get the form definition, and then ask the user to fill out the form and return the user's responses along with the selector string to retrieve the item. Gopher+ clients know how to do this but depend on the "?" tag in the Gopher+ item description to know when to handle this case. The "?" is used in the Gopher+ string to be consistent with Gopher+ protocol's use of this symbol.

3.4.6 Gopher+ item attribute collections

To refer to the Gopher+ attributes of an item, the Gopher URL's Gopher+ string consists of "!" or "$". "!" refers to the all of a Gopher+ item's attributes. "$" refers to all the item attributes for all items in a Gopher directory.

3.4.7 Referring to specific Gopher+ attributes

To refer to specific attributes, the URL's gopher+_string is "!<attribute_name>" or "$<attribute_name>". For example, to refer to

the attribute containing the abstract of an item, the gopher+_string would be "!+ABSTRACT".

To refer to several attributes, the gopher+_string consists of the attribute names separated by coded spaces. For example, "!+ABSTRACT%20+SMELL" refers to the +ABSTRACT and +SMELL attributes of an item.

3.4.8 URL syntax for Gopher+ alternate views

Gopher+ allows for optional alternate data representations (alternate views) of items. To retrieve a Gopher+ alternate view, a Gopher+ client sends the appropriate view and language identifier (found in the item's +VIEW attribute). To refer to a specific Gopher+ alternate view, the URL's Gopher+ string would be in the form:


```
+<view_name>%20<language_name>
```

For example, a Gopher+ string of "+application/postscript%20Es_ES" refers to the Spanish language postscript alternate view of a Gopher+ item.

3.4.9 URL syntax for Gopher+ electronic forms

The gopher+_string for a URL that refers to an item referenced by a Gopher+ electronic form (an ASK block) filled out with specific values is a coded version of what the client sends to the server. The gopher+_string is of the form:

```
+%091%0D%0A+-1%0D%0A<ask_item1_value>%0D%0A<ask_item2_value>%0D%0A.%0D%0A
```

To retrieve this item, the Gopher client sends:

```
<a_gopher_selector><tab>+<tab>1<cr><lf>
+-1<cr><lf>
<ask_item1_value><cr><lf>
<ask_item2_value><cr><lf>
.<cr><lf>
```

to the Gopher server.

3.5. MAILTO

The mailto URL scheme is used to designate the Internet mailing address of an individual or service. No additional information other than an Internet mailing address is present or implied.

A mailto URL takes the form:

mailto:<rfc822-addr-spec>

where <rfc822-addr-spec> is (the encoding of an) addr-spec, as specified in RFC 822 [6]. Within mailto URLs, there are no reserved characters.

Note that the percent sign ("%") is commonly used within RFC 822 addresses and must be encoded.

Unlike many URLs, the mailto scheme does not represent a data object to be accessed directly; there is no sense in which it designates an object. It has a different use than the message/external-body type in MIME.

3.6. NEWS

The news URL scheme is used to refer to either news groups or individual articles of USENET news, as specified in RFC 1036.

A news URL takes one of two forms:

news:<newsgroup-name>
news:<message-id>

A <newsgroup-name> is a period-delimited hierarchical name, such as "comp.infosystems.www.misc". A <message-id> corresponds to the

Message-ID of section 2.1.5 of RFC 1036, without the enclosing "<" and ">"; it takes the form <unique>@<full_domain_name>. A message identifier may be distinguished from a news group name by the presence of the commercial at "@" character. No additional characters are reserved within the components of a news URL.

If <newsgroup-name> is "*" (as in <URL:news:*>), it is used to refer to "all available news groups".

The news URLs are unusual in that by themselves, they do not contain sufficient information to locate a single resource, but, rather, are location-independent.

3.7. NNTP

The nntp URL scheme is an alternative method of referencing news articles, useful for specifying news articles from NNTP servers (RFC 977).

A nntp URL take the form:

nntp://<host>:<port>/<newsgroup-name>/<article-number>

where <host> and <port> are as described in Section 3.1. If :<port> is omitted, the port defaults to 119.

The <newsgroup-name> is the name of the group, while the <article-number> is the numeric id of the article within that newsgroup.

Note that while nntp: URLs specify a unique location for the article resource, most NNTP servers currently on the Internet today are configured only to allow access from local clients, and thus nntp URLs do not designate globally accessible resources. Thus, the news: form of URL is preferred as a way of identifying news articles.

3.8. TELNET

The Telnet URL scheme is used to designate interactive services that may be accessed by the Telnet protocol.

A telnet URL takes the form:

telnet://<user>:<password>@<host>:<port>/

as specified in Section 3.1. The final "/" character may be omitted. If :<port> is omitted, the port defaults to 23. The :<password> can be omitted, as well as the whole <user>:<password> part.

This URL does not designate a data object, but rather an interactive service. Remote interactive services vary widely in the means by which they allow remote logins; in practice, the <user> and <password> supplied are advisory only: clients accessing a telnet URL merely advise the user of the suggested username and password.

3.9. WAIS

The WAIS URL scheme is used to designate WAIS databases, searches, or individual documents available from a WAIS database. WAIS is described in [7]. The WAIS protocol is described in RFC 1625 [17]; Although the WAIS protocol is based on Z39.50-1988, the WAIS URL scheme is not intended for use with arbitrary Z39.50 services.

A WAIS URL takes one of the following forms:

wais://<host>:<port>/<database>
wais://<host>:<port>/<database>?<search>
wais://<host>:<port>/<database>/<wtype>/<wpath>

where <host> and <port> are as described in Section 3.1. If :<port> is omitted, the port defaults to 210. The first form designates a WAIS database that is available for searching. The second form designates a particular search. <database> is the name of the WAIS database being queried.

The third form designates a particular document within a WAIS database to be retrieved. In this form <wtype> is the WAIS designation of the type of the object. Many WAIS implementations require that a client know the "type" of an object prior to retrieval, the type being returned along with the internal object identifier in the search response. The <wtype> is included in the URL in order to allow the client interpreting the URL adequate information to actually retrieve the document.

The <wpath> of a WAIS URL consists of the WAIS document-id, encoded as necessary using the method described in Section 2.2. The WAIS document-id should be treated opaquely; it may only be decomposed by the server that issued it.

3.10 FILES

The file URL scheme is used to designate files accessible on a particular host computer. This scheme, unlike most other URL schemes, does not designate a resource that is universally accessible over the Internet.

A file URL takes the form:

file://<host>/<path>

where <host> is the fully qualified domain name of the system on which the <path> is accessible, and <path> is a hierarchical directory path of the form <directory>/<directory>/.../<name>.

For example, a VMS file

DISK$USER:[MY.NOTES]NOTE123456.TXT

might become

<URL:file://vms.host.edu/disk$user/my/notes/note12345.txt>

As a special case, <host> can be the string "localhost" or the empty string; this is interpreted as `the machine from which the URL is being interpreted'.

The file URL scheme is unusual in that it does not specify an Internet protocol or access method for such files; as such, its utility in network protocols between hosts is limited.

3.11 PROSPERO

The Prospero URL scheme is used to designate resources that are accessed via the Prospero Directory Service. The Prospero protocol is described elsewhere [14].

A prospero URLs takes the form:

prospero://<host>:<port>/<hsoname>;<field>=<value>

where <host> and <port> are as described in Section 3.1. If :<port> is omitted, the port defaults to 1525. No username or password is

allowed.

The <hsoname> is the host-specific object name in the Prospero protocol, suitably encoded. This name is opaque and interpreted by the Prospero server. The semicolon ";" is reserved and may not appear without quoting in the <hsoname>.

Prospero URLs are interpreted by contacting a Prospero directory server on the specified host and port to determine appropriate access methods for a resource, which might themselves be represented as different URLs. External Prospero links are represented as URLs of the underlying access method and are not represented as Prospero URLs.

Note that a slash "/" may appear in the <hsoname> without quoting and no significance may be assumed by the application. Though slashes may indicate hierarchical structure on the server, such structure is not guaranteed. Note that many <hsoname>s begin with a slash, in which case the host or port will be followed by a double slash: the slash from the URL syntax, followed by the initial slash from the <hsoname>. (E.g., <URL:prospero://host.dom//pros/name> designates a <hsoname> of "/pros/name".)

In addition, after the <hsoname>, optional fields and values associated with a Prospero link may be specified as part of the URL. When present, each field/value pair is separated from each other and from the rest of the URL by a ";" (semicolon). The name of the field and its value are separated by a "=" (equal sign). If present, these

fields serve to identify the target of the URL. For example, the OBJECT-VERSION field can be specified to identify a specific version of an object.

4. REGISTRATION OF NEW SCHEMES

A new scheme may be introduced by defining a mapping onto a conforming URL syntax, using a new prefix. URLs for experimental schemes may be used by mutual agreement between parties. Scheme names starting with the characters "x-" are reserved for experimental purposes.

The Internet Assigned Numbers Authority (IANA) will maintain a registry of URL schemes. Any submission of a new URL scheme must include a definition of an algorithm for accessing of resources within that scheme and the syntax for representing such a scheme.

URL schemes must have demonstrable utility and operability. One way to provide such a demonstration is via a gateway which provides objects in the new scheme for clients using an existing protocol. If

the new scheme does not locate resources that are data objects, the properties of names in the new space must be clearly defined.

New schemes should try to follow the same syntactic conventions of existing schemes, where appropriate. It is likewise recommended that, where a protocol allows for retrieval by URL, that the client software have provision for being configured to use specific gateway locators for indirect access through new naming schemes.

The following scheme have been proposed at various times, but this document does not define their syntax or use at this time. It is suggested that IANA reserve their scheme names for future definition:

afs	Andrew File System global file names.
mid	Message identifiers for electronic mail.
cid	Content identifiers for MIME body parts.
nfs	Network File System (NFS) file names.
tn3270	Interactive 3270 emulation sessions.

mailserver Access to data available from mail servers.
z39.50 Access to ANSI Z39.50 services.

5. BNF for specific URL schemes

This is a BNF-like description of the Uniform Resource Locator syntax, using the conventions of RFC822, except that "|" is used to designate alternatives, and brackets [] are used around optional or repeated elements. Briefly, literals are quoted with "", optional elements are enclosed in [brackets], and elements may be preceded with <n>* to designate n or more repetitions of the following element; n defaults to 0.

```
; The generic form of a URL is:

genericurl     = scheme ":" schemepart

; Specific predefined schemes are defined here; new schemes
; may be registered with IANA

url            = httpurl | ftpurl | newsurl |
                 nntpurl | telneturl | gopherurl |
                 waisurl | mailtourl | fileurl |
                 prosperourl | otherurl

; new schemes follow the general syntax
otherurl       = genericurl

; the scheme is in lower case; interpreters should use case-ignore
scheme         = 1*[ lowalpha | digit | "+" | "-" | "." ]
```



```
schemepart     = *xchar | ip-schemepart


; URL schemeparts for ip based protocols:

ip-schemepart  = "//" login [ "/" urlpath ]

login          = [ user [ ":" password ] "@" ] hostport
```

```
hostport      = host [ ":" port ]
host          = hostname | hostnumber
hostname      = *[ domainlabel "." ] toplabel
domainlabel   = alphadigit | alphadigit *[ alphadigit | "-" ] alphadigit
toplabel      = alpha | alpha *[ alphadigit | "-" ] alphadigit
alphadigit    = alpha | digit
hostnumber    = digits "." digits "." digits "." digits
port          = digits
user          = *[ uchar | ";" | "?" | "&" | "=" ]
password      = *[ uchar | ";" | "?" | "&" | "=" ]
urlpath       = *xchar    ; depends on protocol see section 3.1

; The predefined schemes:

; FTP (see also RFC959)

ftpurl        = "ftp://" login [ "/" fpath [ ";type=" ftptype ]]
fpath         = fsegment *[ "/" fsegment ]
fsegment      = *[ uchar | "?" | ":" | "@" | "&" | "=" ]
ftptype       = "A" | "I" | "D" | "a" | "i" | "d"

; FILE

fileurl       = "file://" [ host | "localhost" ] "/" fpath

; HTTP

httpurl       = "http://" hostport [ "/" hpath [ "?" search ]]
hpath         = hsegment *[ "/" hsegment ]
hsegment      = *[ uchar | ";" | ":" | "@" | "&" | "=" ]
search        = *[ uchar | ";" | ":" | "@" | "&" | "=" ]

; GOPHER (see also RFC1436)

gopherurl     = "gopher://" hostport [ / [ gtype [ selector
              [ "%09" search [ "%09" gopher+_string ] ] ] ] ]
gtype         = xchar
selector      = *xchar
gopher+_string = *xchar
```

```
; MAILTO (see also RFC822)

mailtourl      = "mailto:" encoded822addr
encoded822addr = 1*xchar              ; further defined in RFC822

; NEWS (see also RFC1036)

newsurl        = "news:" grouppart
grouppart      = "*" | group | article
group          = alpha *[ alpha | digit | "-" | "." | "+" | "_" ]
article        = 1*[ uchar | ";" | "/" | "?" | ":" | "&" | "=" ] "@" host

; NNTP (see also RFC977)

nntpurl        = "nntp://" hostport "/" group [ "/" digits ]

; TELNET

telneturl      = "telnet://" login [ "/" ]

; WAIS (see also RFC1625)

waisurl        = waisdatabase | waisindex | waisdoc
waisdatabase   = "wais://" hostport "/" database
waisindex      = "wais://" hostport "/" database "?" search
waisdoc        = "wais://" hostport "/" database "/" wtype "/" wpath
database       = *uchar
wtype          = *uchar
wpath          = *uchar

; PROSPERO

prosperourl    = "prospero://" hostport "/" ppath *[ fieldspec ]
ppath          = psegment *[ "/" psegment ]
psegment       = *[ uchar | "?" | ":" | "@" | "&" | "=" ]
fieldspec      = ";" fieldname "=" fieldvalue
fieldname      = *[ uchar | "?" | ":" | "@" | "&" ]
fieldvalue     = *[ uchar | "?" | ":" | "@" | "&" ]

; Miscellaneous definitions

lowalpha       = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" |
                 "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" |
                 "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" |
```

```
            "y" | "z"
hialpha       = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
            "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
            "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
```



```
alpha         = lowalpha | hialpha
digit         = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
            "8" | "9"
safe          = "$" | "-" | "_" | "." | "+"
extra         = "!" | "*" | "'" | "(" | ")" | ","
national      = "{" | "}" | "|" | "\" | "^" | "~" | "[" | "]" | "`"
punctuation   = "<" | ">" | "#" | "%" | <">


reserved      = ";" | "/" | "?" | ":" | "@" | "&" | "="
hex           = digit | "A" | "B" | "C" | "D" | "E" | "F" |
            "a" | "b" | "c" | "d" | "e" | "f"
escape        = "%" hex hex

unreserved    = alpha | digit | safe | extra
uchar         = unreserved | escape
xchar         = unreserved | reserved | escape
digits        = 1*digit
```

6. Security Considerations

The URL scheme does not in itself pose a security threat. Users should beware that there is no general guarantee that a URL which at one time points to a given object continues to do so, and does not even at some later time point to a different object due to the movement of objects on servers.

A URL-related security threat is that it is sometimes possible to construct a URL such that an attempt to perform a harmless idempotent operation such as the retrieval of the object will in fact cause a possibly damaging remote operation to occur. The unsafe URL is typically constructed by specifying a port number other than that reserved for the network protocol in question. The client unwittingly contacts a server which is in fact running a different

protocol. The content of the URL contains instructions which when interpreted according to this other protocol cause an unexpected operation. An example has been the use of gopher URLs to cause a rude message to be sent via a SMTP server. Caution should be used when using any URL which specifies a port number other than the default for the protocol, especially when it is a number within the reserved space.

Care should be taken when URLs contain embedded encoded delimiters for a given protocol (for example, CR and LF characters for telnet protocols) that these are not unencoded before transmission. This would violate the protocol but could be used to simulate an extra operation or parameter, again causing an unexpected and possible harmful remote operation to be performed.

The use of URLs containing passwords that should be secret is clearly unwise.

7. Acknowledgements

This paper builds on the basic WWW design (RFC 1630) and much discussion of these issues by many people on the network. The discussion was particularly stimulated by articles by Clifford Lynch, Brewster Kahle [10] and Wengyik Yeong [18]. Contributions from John Curran, Clifford Neuman, Ed Vielmetti and later the IETF URL BOF and URI working group were incorporated.

Most recently, careful readings and comments by Dan Connolly, Ned Freed, Roy Fielding, Guido van Rossum, Michael Dolan, Bert Bos, John Kunze, Olle Jarnefors, Peter Svanberg and many others have helped refine this RFC.

APPENDIX: Recommendations for URLs in Context

URIs, including URLs, are intended to be transmitted through protocols which provide a context for their interpretation.

In some cases, it will be necessary to distinguish URLs from other possible data structures in a syntactic structure. In this case, is recommended that URLs be preceeded with a prefix consisting of the characters "URL:". For example, this prefix may be used to distinguish URLs from other kinds of URIs.

In addition, there are many occasions when URLs are included in other kinds of text; examples include electronic mail, USENET news messages, or printed on paper. In such cases, it is convenient to

have a separate syntactic wrapper that delimits the URL and separates it from the rest of the text, and in particular from punctuation marks that might be mistaken for part of the URL. For this purpose, is recommended that angle brackets ("<" and ">"), along with the prefix "URL:", be used to delimit the boundaries of the URL. This wrapper does not form part of the URL and should not be used in contexts in which delimiters are already specified.

In the case where a fragment/anchor identifier is associated with a URL (following a "#"), the identifier would be placed within the brackets as well.

In some cases, extra whitespace (spaces, linebreaks, tabs, etc.) may need to be added to break long URLs across lines. The whitespace should be ignored when extracting the URL.

No whitespace should be introduced after a hyphen ("-") character. Because some typesetters and printers may (erroneously) introduce a hyphen at the end of line when breaking a line, the interpreter of a URL containing a line break immediately after a hyphen should ignore all unencoded whitespace around the line break, and should be aware that the hyphen may or may not actually be part of the URL.

Examples:

> Yes, Jim, I found it under <URL:ftp://info.cern.ch/pub/www/doc; type=d> but you can probably pick it up from <URL:ftp://ds.in ternic.net/rfc>. Note the warning in <URL:http://ds.internic. net/instructions/overview.html#WARNING>.

References

[1] Anklesaria, F., McCahill, M., Lindner, P., Johnson, D., Torrey, D., and B. Alberti, "The Internet Gopher Protocol

(a distributed document search and retrieval protocol)",
RFC 1436, University of Minnesota, March 1993.
<URL:ftp://ds.internic.net/rfc/rfc1436.txt;type=a>

[2] Anklesaria, F., Lindner, P., McCahill, M., Torrey, D.,
Johnson, D., and B. Alberti, "Gopher+: Upward compatible
enhancements to the Internet Gopher protocol",
University of Minnesota, July 1993.
<URL:ftp://boombox.micro.umn.edu/pub/gopher/gopher_protocol
/Gopher+/Gopher+.txt>

[3] Berners-Lee, T., "Universal Resource Identifiers in WWW: A
Unifying Syntax for the Expression of Names and Addresses of
Objects on the Network as used in the World-Wide Web", RFC
1630, CERN, June 1994.
<URL:ftp://ds.internic.net/rfc/rfc1630.txt>

[4] Berners-Lee, T., "Hypertext Transfer Protocol (HTTP)",
CERN, November 1993.
<URL:ftp://info.cern.ch/pub/www/doc/http-spec.txt.Z>

[5] Braden, R., Editor, "Requirements for Internet Hosts --
Application and Support", STD 3, RFC 1123, IETF, October 1989.
<URL:ftp://ds.internic.net/rfc/rfc1123.txt>

[6] Crocker, D. "Standard for the Format of ARPA Internet Text
Messages", STD 11, RFC 822, UDEL, April 1982.
<URL:ftp://ds.internic.net/rfc/rfc822.txt>

[7] Davis, F., Kahle, B., Morris, H., Salem, J., Shen, T., Wang, R.,
Sui, J., and M. Grinbaum, "WAIS Interface Protocol Prototype
Functional Specification", (v1.5), Thinking Machines
Corporation, April 1990.
<URL:ftp://quake.think.com/pub/wais/doc/protspec.txt>

[8] Horton, M. and R. Adams, "Standard For Interchange of USENET
Messages", RFC 1036, AT&T Bell Laboratories, Center for Seismic
Studies, December 1987.
<URL:ftp://ds.internic.net/rfc/rfc1036.txt>

[9] Huitema, C., "Naming: Strategies and Techniques", Computer
Networks and ISDN Systems 23 (1991) 107-110.

[10] Kahle, B., "Document Identifiers, or International Standard Book Numbers for the Electronic Age", 1991. <URL:ftp://quake.think.com/pub/wais/doc/doc-ids.txt>

[11] Kantor, B. and P. Lapsley, "Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News", RFC 977, UC San Diego & UC Berkeley, February 1986. <URL:ftp://ds.internic.net/rfc/rfc977.txt>

[12] Kunze, J., "Functional Requirements for Internet Resource Locators", Work in Progress, December 1994. <URL:ftp://ds.internic.net/internet-drafts /draft-ietf-uri-irl-fun-req-02.txt>

[13] Mockapetris, P., "Domain Names - Concepts and Facilities", STD 13, RFC 1034, USC/Information Sciences Institute, November 1987. <URL:ftp://ds.internic.net/rfc/rfc1034.txt>

[14] Neuman, B., and S. Augart, "The Prospero Protocol", USC/Information Sciences Institute, June 1993. <URL:ftp://prospero.isi.edu/pub/prospero/doc /prospero-protocol.PS.Z>

[15] Postel, J. and J. Reynolds, "File Transfer Protocol (FTP)", STD 9, RFC 959, USC/Information Sciences Institute, October 1985. <URL:ftp://ds.internic.net/rfc/rfc959.txt>

[16] Sollins, K. and L. Masinter, "Functional Requirements for Uniform Resource Names", RFC 1737, MIT/LCS, Xerox Corporation, December 1994. <URL:ftp://ds.internic.net/rfc/rfc1737.txt>

[17] St. Pierre, M, Fullton, J., Gamiel, K., Goldman, J., Kahle, B., Kunze, J., Morris, H., and F. Schiettecatte, "WAIS over Z39.50-1988", RFC 1625, WAIS, Inc., CNIDR, Thinking Machines Corp., UC Berkeley, FS Consulting, June 1994. <URL:ftp://ds.internic.net/rfc/rfc1625.txt>

[18] Yeong, W. "Towards Networked Information Retrieval", Technical report 91-06-25-01, Performance Systems International, Inc. <URL:ftp://uu.psi.com/wp/nir.txt>, June 1991.

[19] Yeong, W., "Representing Public Archives in the Directory", Work in Progress, November 1991.

[20] "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4-1986.

Editors' Addresses

Tim Berners-Lee
World-Wide Web project
CERN,
1211 Geneva 23,
Switzerland

Phone: +41 (22)767 3755
Fax: +41 (22)767 7155
EMail: timbl@info.cern.ch

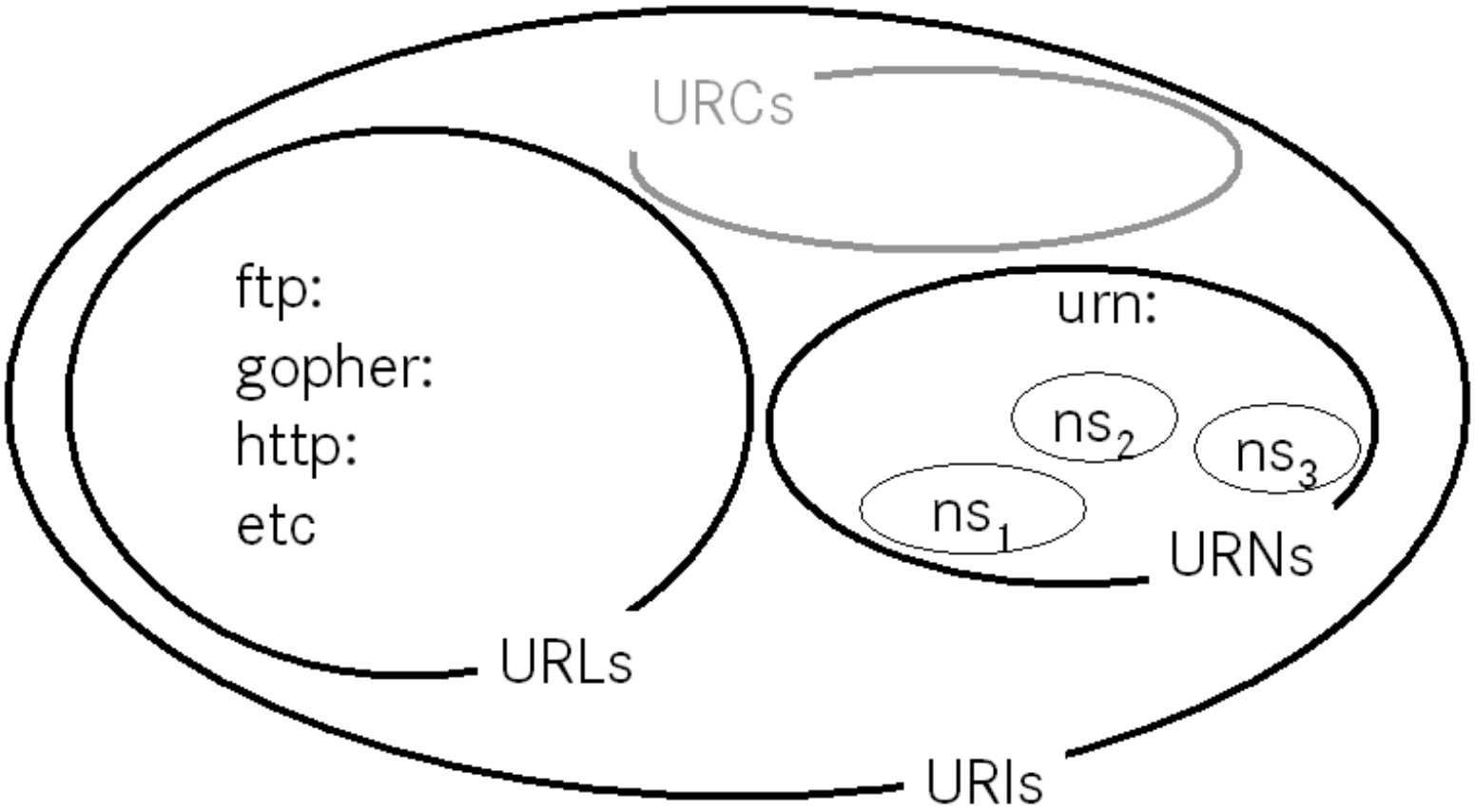
Larry Masinter
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94034

Phone: (415) 812-4365
Fax: (415) 812-4333
EMail: masinter@parc.xerox.com

Mark McCahill
Computer and Information Services,
University of Minnesota
Room 152 Shepherd Labs

100 Union Street SE
Minneapolis, MN 55455

Phone: (612) 625 1300
EMail: mpm@boombox.micro.umn.edu


URCs
ftp:
gopher:
http:
etc
urn:
ns2
ns3
ns1
URNs
URLs
URIs

URIs, URLs, and URNs: Clarifications and Recommendations 1.0

Report from the joint W3C/IETF URI Planning Interest Group

W3C Note 21 September 2001

This version:
http://www.w3.org/TR/2001/NOTE-uri-clarification-20010921/
Latest version:
http://www.w3.org/TR/uri-clarification

by:
URI Planning Interest Group, W3C/IETF
(see Acknowledgements)

Abstract

This paper addresses and attempts to clarify two issues pertaining to URIs, and presents recommendations. Section 1 addresses how URI space is partitioned and the relationship between URIs, URLs, and URNs. Section 2 describes how URI schemes and URN namespace ids are registered. Section 3 mentions additional unresolved issues not considered by this paper and section 4 presents recommendations.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C in a list of current W3C technical reports at http://www.w3.org/TR/ .

This is a report from the W3C/IETF URI Planning Interest Group, for review by W3C members, the IETF community, and other interested parties. We invite review and discussion of our recommendations for future work in the IETF and/or W3C. Please address your feedback to uri@w3.org, a mailing list with public archive.

This document has been produced as part of the W3C URI Activity.

Table of Contents

Appendixes

---

1 URI Partitioning

There is some confusion in the web community over the partitioning of URI space, specifically, the relationship among the concepts of URL, URN, and URI. The confusion owes to the incompatibility between two different views of URI partitioning, which we call the "classical" and "contemporary" views.

1.1 Classical View

During the early years of discussion of web identifiers (early to mid 90s), people assumed that an identifer type would be cast into one of two (or possibly more) classes. An identifier might specify the location of a resource (a URL) or its name (a URN) independent of location. Thus a URI was either a URL or a URN. There was discussion about generalizing this by addition of a discrete number of additional classes; for example, a URI might point to metadata rather than the resource itself, in which case the URI would be a URC (citation). URI space was thus viewed as partitioned into subspaces: URL and URN, and additional subspaces, to be defined. The only such additional space ever proposed was URC and there never was any buy-in; so without loss of generality it's reasonable to say that URI space was thought to be partitioned into two classes: URL and URN. Thus for example, "`http:`" was a URL scheme, and "`isbn:`" would (someday) be a URN scheme. Any new scheme would be cast into one or the other of these two classes.

1.2 Contemporary View

Over time, the importance of this additional level of hierarchy seemed to lessen; the view became that an individual scheme does not need to be cast into one of a discrete set of URI types such as "URL", "URN", "URC", etc. Web-identifer schemes are in general URI schemes; a given URI scheme may define subspaces. Thus "`http:`" is a URI scheme. "`urn:`" is also a URI scheme; it defines subspaces, called "namespaces". For example, the set of URNs of the form "`urn:isbn:n-nn-nnnnnn-n`" is a URN namespace. ("`isbn`" is an URN namespace identifier. It is not a "URN scheme" nor a "URI scheme").

Further according to the contemporary view, the term "URL" does not refer to a formal partition of URI space; rather, URL is a useful but informal concept: a URL is a type of URI that identifies a resource via a representation of its primary access mechanism (e.g., its network "location"), rather than by some other attributes it may have. Thus as we noted, "`http:`" is a URI scheme. An http URI is a URL. The phrase "URL scheme" is now used infrequently, usually to refer to some subclass of URI schemes which exclude URNs.

1.3 Confusion

The body of documents (RFCs, etc) covering URI architecture, syntax, registration, etc., spans both the classical and contemporary periods. People who are well-versed in URI matters tend to use "URL" and "URI" in ways that seem to be interchangable. Among these experts, this isn't a problem. But among the Internet community at large, it is. People are not convinced that URI and URL mean the same thing, in documents where they (apparently) do. When one sees an RFC that talks about URI schemes (e.g. [RFC 2396]), another that talks about URL schemes (e.g. [RFC 2717]), and yet another that talks of URN schemes ([RFC 2276]) it is natural to wonder what's the difference, and how they relate to one another. While RFC 2396 1.2 attempts to address the distinction between URIs, URLs and URNs, it has not been successful in clearing up the confusion.

2 Registration

This section examines the state of registration of URI schemes and URN namespaces and the mechanisms by which registration currently occurs.

2.1 URI Schemes

2.1.1 Registered URI schemes

The official register of URI scheme names is maintained by IANA, at http://www.iana.org/assignments/uri-schemes . For each scheme, the RFC that defines the scheme is listed, for example "`http:`" is defined by [RFC 2616]. The table currently lists 30 schemes. In addition, there are a few "reserved" scheme names; at one point in time these were intended to become registered schemes but have since been dropped.

2.1.2 Unregistered URI Schemes

We distinguish between public (unregistered) and private schemes. A public scheme (registered or not), is one for which there is some public document describing it.

2.1.2.1 Public Unregistered Schemes

Dan Connolly maintains a list of known, public URI schemes, both registered and un-registered, a total of 84 schemes. 50 or so of these are unregistered (not listed in the IANA register). Some may be obsolete (for example, it appears that "`phone`", is obsolete, superceded by "`tel`"). Some have an

RFC, but are not included in the IANA list.

2.1.2.2 Private Schemes

It's probably impossible to determine all of these, and it's not clear that it's worthwhile to try, except perhaps to get some idea of their number. In the minutes of the August 1997 IETF meeting is the observation that there may be 20-40 in use at Microsoft, with 2-3 being added a day, and that WebTV has 24, with 6 added per year.

2.1.3 Registration of URI Schemes

[RFC 2717] specifies procedures for registering scheme names, and points to [RFC 2718] which supplies guidelines. RFC 2717 describes an organization of schemes into "trees".

2.1.3.1 IETF Tree

The IETF tree is intended for schemes of general interest to the Internet community, and which require a substantive review and approval process. Registration in the IETF tree requires publication of the scheme syntax and semantics in an RFC.

2.1.3.2 Other Trees

Although RFC 2717 describes "alternative trees", no alternative trees have been registered to date, although a vendor-supplied tree ("vnd") is pending. URI schemes in alternative trees will be distinguished because they will have a "." in the scheme name.

2.2 URN Namespaces

A URN namespace is identified by a "Namespace ID", NID, which is registered with IANA (see **2.2.4 Registration Procedures for URN NIDs**).

2.2.1 Registered URN NIDs

There are two categories of registered URN NIDs:

- Informal: These are of the form "urn-<number>" where <number> is assigned by IANA. There are three registered in this category (urn-1, urn-2, and urn-3).
- Formal: The official list of registered NIDs is kept by IANA at http://www.iana.org/assignments/urn-namespaces. Currently it lists eight

registered NIDs:

- 'ietf', defined by [RFC 2648], *URN Namespace for IETF Documents*
- 'pin', defined by [RFC 3043], *The Network Solutions Personal Internet Name (PIN): A URN Namespace for People and Organizations*
- 'issn' defined by [RFC 3043], *Using The ISSN as URN within an ISSN-URN Namespace*
- 'oid' defined by [RFC 3061], *A URN Namespace of Object Identifiers*
- 'newsml' defined by [RFC 3085], *URN Namespace for NewsML Resources*
- 'oasis' defined by [RFC 3121], *A URN Namespace for OASIS*
- 'xmlorg' defined by [RFC 3120], *A URN Namespace for XML.org*
- 'publicid' defined by [RFC 3151], *A URN Namespace for Public Identifiers*

2.2.2 Pending URN NIDs

There are a number of pending URN NID registration requests but there is no reliable way to discover them, or their status. For example, 'isbn' and 'nbn' have been approved by the IESG and are in the RFC Editor's queue. 'isbn', as a potential URN namespace (or URI scheme), in particular has been a source of much speculation and confusion over several years. It would be helpful if there were some formal means to track the status of NID requests such as 'isbn'.

2.2.3 Unregistered NIDs

In the "unregistered" category (besides the experimental case, not described in this paper) there are bonafide NIDs that just haven't bothered to even explore the process of registration.The most prominent that comes to mind is 'hdl'. In the case of 'hdl', it has been speculated that this scheme has not been registered because it is not clear to the owners whether it should be registered as a URI scheme or as a URN namespace.

2.2.4 Registration Procedures for URN NIDs

[RFC 2611] describes the mechanism to obtain an NID for a URN namespace, which is registered with IANA.

A request for an NID should describe features including: structural characteristic of identifiers (for example, features relevant to caching/ shortcuts approaches); specific character encoding rules (e.g., which character should be used for single-quotes); RFCs, standards, etc, that explains the namespace structure; identifier uniqueness considerations; delegation of assignment authority, including how to become an assigner of identifiers; identifier persistence considerations; quality of service considerations; process for identifier resolution; rules for lexical equivalence; any special considerations required for conforming with the URN syntax (particularly applicable in the case of legacy naming systems); validation mechanisms (determining whether a given string is currently a validly-assigned URN; and scope (for example,"United States social security numbers").

3 Additional URI Issues

There are additional unresolved URI issues, not considered by this paper, which we hope will be addressed by a follow-on effort. We have not attempted to completely enumerate these issues, however, they include (but are not limited to) the following:

- The use of URIs as identifiers that don't actually identify network resources (for example they identify an abstract object such as an XML schema, or a physical object such as a book or even a person).
- IRIs (International Resource Identifiers): the extension of URI syntax to non-ASCII.

4 Recommendations

We recommend the following:

1. The W3C and IETF should jointly develop and endorse a model for URIs, URLs and URNs consistent with the '"Contemporary View" described in section 1, and which considers the additional URI issues listed or alluded to in section 3.
2. RFCs such as 2717 ("Registration Procedures for URL Scheme Names") and 2718 ("Guidelines for new URL Schemes") should both be generalized to refer to "URI schemes" rather that "URL schemes" and, after refinement, moved forward as Best Current Practice in IETF.
3. The registration procedures for alternative trees should be clarified in RFC 2717.
4. Public but unregistered schemes should become registered, where possible. Obsolete schemes should be purged or clearly marked as

obsolete.

5. IANA registration information should be updated:
    - Add 'urn' to the list of registered URI schemes with a pointer to the URN namespace registry.
    - Maintain status information about pending registrations (URI schemes and URN NID requests).
    - Insure that it is clear that the page is the official registry, e.g., by adding a heading to the effect "This is the Official IANA Registry of URI Schemes".

A Acknowledgements

The participants in the URI Planning Interest Group are:

- Tony Coates
- Dan Connolly
- Diana Dack
- Leslie Daigle
- Ray Denenberg
- Martin Dürst
- Paul Grosso
- Sandro Hawke
- Renato Iannella
- Graham Klyne
- Larry Masinter
- Michael Mealling
- Mark Needleman
- Norman Walsh

B References

RFC 2717
IETF (Internet Engineering Task Force). *Registration Procedures for URL Scheme Names*, ed. R. Petke and I. King. 1999.

RFC 2718
IETF (Internet Engineering Task Force). *Guidelines for new URL Schemes*, ed. L. Masinter, H. Alvestrand, D. Zigmond, and R. Petke. 1999.

RFC 2648
IETF (Internet Engineering Task Force). *A URN Namespace for IETF Documents* ed. R. Moats. 1999.

RFC 3043
IETF (Internet Engineering Task Force). *The Network Solutions*

Personal Internet Name (PIN): A URN Namespace for People and Organizations ed. M. Mealling. 2001.

RFC 3044

IETF (Internet Engineering Task Force). *Using The ISSN (International Serial Standard Number) as URN (Uniform Resource Names) within an ISSN-URN Namespace* ed. S. Rozenfeld. 2001.

RFC 3061

IETF (Internet Engineering Task Force). *A URN Namespace of Object Identifiers* ed. M. Mealling, 2001.

RFC 3085

IETF (Internet Engineering Task Force). *URN Namespace for NewsML Resources* ed. A. Coates, D. Allen, and D. Rivers-Moore. 2001.

RFC 3121

IETF (Internet Engineering Task Force). *A URN Namespace for OASIS* ed. K. Best and N. Walsh. 2001.

RFC 3120

IETF (Internet Engineering Task Force). *A URN Namespace for XML.org* ed. K. Best and N. Walsh. 2001.

RFC 3151

IETF (Internet Engineering Task Force). *A URN Namespace for Public Identifiers*, ed. N. Walsh, J. Cowan, and P. Grosso. 2001.

RFC 2611

IETF (Internet Engineering Task Force). *URN Namespace Definition Mechanisms*, ed. L. Daigle, R. Iannella, and P. Faltstrom. 1999.

RFC 2396

IETF (Internet Engineering Task Force). *Uniform Resource Identifiers (URI): Generic Syntax*, ed. T. Berners-Lee, R. Fielding, and L. Masinter. 1998.

RFC 2276

IETF (Internet Engineering Task Force). *Architectural Principles of Uniform Resource Name Resolution*, ed. K. Sollins. 1998.

RFC 2616

IETF (Internet Engineering Task Force). *Hypertext Transfer Protocol -- HTTP/1.1*, ed. R. Fielding, J. Gettys, J. Mogul, et. al. 1999.

Seventh Heaven

The Anatomy of an URL

Internet-Scale Namespaces, Part I

Rohit Khare * 4K Associates * September 1, 1999

In 1977 Charles and Ray Eames produced a classic short that opened with a shot of a couple lounging on a picnic blanket in a lakeside park in Chicago. Over the next eight minutes they stepped out one order of magnitude at a time to depict the park, the city, the planet, all the way up to galactic clusters -- only to zoom all the way back into his hand into its cells, proteins, and even subatomic particles.

Powers of Ten introduces two complementary points. First, that we have different scientific vocabularies for phenomena at various scales, ranging up from meteorology to geology to astronomy to cosmology; and ranging down from biology to chemistry to quantum mechanics. Second, the exact same physical laws apply at every scale. There are still scale-invariant laws underlying it all.

At UC Irvine's recent Workshop on Internet Scale Technologies (TWIST99) on Namespaces, I stole the same rhetorical framework to zoom in on a single Web

page, decomposing one Internet-scale name into a host of more specialized names; and to zoom out to the larger social 'web of trust' such transactions are embedded in. Like the Eames', our whirlwind tour will attempt to illustrate unique features of each namespaces' domain of discourse, while simultaneously identifying cross-cutting, uniquely Internet-scale management issues.

Anatomy of an URL

Let's say I want to buy an airline ticket today. I've already made a reservation, so I fire up my Web browser and type the Uniform Resource Identifier http://www.united.com/Itinerary/NQSS5A into its 'Address' field. To me, that's just an opaque address which is directly resolved into a web page. That's why this format is also known as a Uniform Resource Locator: it can be directly interpreted as the process to find that web page as much as its destination. Contrast it with the difference between a postal address and directions to the same building.

On the other hand, it's less of a locator than the schemes it succeeded. Directing folks to Internet information resources before URIs required writing out several lines of instructions, as documented by the MIME message/external-body access types in RFC 1521. When Tim Berners-Lee set out to define a format suitable even for "cocktail napkins," he was able to reduce the recipe to one line, as scheme://user:password@host:port/path. Ease of transcription came at the expense of character repertoire, though. While URLs look great in English, ad-

hoc internationalization efforts must squeeze into the few legal alphanumeric US-ASCII characters permitted by the URI specification (originally RFC 1630, in June 1994, and standardized in August 1998 as RFC 2396).

It's certainly been a successful namespace. The big search engine projects claim there are over a billion web pages out there, each with its place in a site's hierarchy (URLs and URL paths are read left-to-right). We know from experience that some URIs are ephemeral to the point of expiring in seconds, while some have been around for a decade already. And as user interface goes, well, you'd never see anonymous FTP instructions on a toothpick wrapper...

Scheme

To the Web browser, though, the URI in the 'Address' field isn't. It has to resolve each of the namespaces embedded within that URI. The URI Scheme is the very first part, since it selects the namespace and syntax (and typically, access protocol) for the rest of the URI.

Typical entries in this namespace are telnet: and ftp:, but there are also less protocol-specific schemes such as mailto: and even data: (which is followed by an arbitrary block of base-64 encoded bytes!) Security flags have also been crammed into the scheme, using a single, unreliable 's' to indicate a separate port for https (443) or ftps (990), for example.

At the same time, if URIs are expected to be stable for decades there can't be too many schemes. The requirement to publish an IETF RFC to become an IANA-

registered scheme enforces scarcity. It's also reflected in the limited ASCII token space, as well as the preclusion of internationalized variants.

Domain Name

Moving left-to-right in the http: URI syntax, we encounter the string "www.united.com." The Domain Name System interprets it in turn as a hierarchical right-to-left identifier within a global, singly-rooted tree. That is to say, the new Internet Corporation for Assigned Names and Numbers ultimately stands behind the (literally, anonymous) "." domain, delegating .com, .int, .mil and other national codes authority to "top-level domain" registrars. Physically, though, there are 13 "root servers" (the most that can fit in a single UDP packet) which mirror each others' copies of the entire DNS database.

Sitting at the edge of the network, the browser isn't concerned with any of these details. It passes that address to the local caching DNS resolver, which in turn hooks into the global pyramid scheme to find .com (today, from IANA), then united.com (today, from Network Solutions), and then www.united.com (from United).

Of course, figuring out whether it's United Air Lines, United Moving Lines, or United Parcel Service is the original sin inherent in DNS. The 1982-3 vision of replacing manual site-local "hosts.txt" lookup files with a unified distributed database has succeeded only too well, entrenching it as the only possible human-friendly IP address directory scheme. The trademark, regulatory, and

commercial morass of the mid-90's will ultimately set the stage for parallel resolution services such as RealNames or, arguably, Yahoo! categories.

DNS namespaces have a few more interesting constraints. There is little range for internationalization, since only the letters A-Z, digits, and '-' are permitted. The protocol limits fully-qualified domain names (FQDN's) to 255 characters maximum and components to 63 (while .com entries are further restricted to only 20 characters). While names were expected to remain valid for years or decades, dialup and mobile access have motivated proposals for Dynamic DNS as well. Since DNS namespace also contains its inverse -- by looking up an IP address under the in-addr.arpa "domain" -- dynamism is even more difficult to accomodate.

IP Address

Once the browser reduces a domain name to a 32-bit IPv4 address, it is combined with a 16-bit port number (usually determined by the URI scheme, such as HTTP on port 80) and passed down one further layer. The TCP/IP transport stack has to resolve this "name" to a next-hop (or destination) machine.

Unbeknownst to the application layer, the underlying routing algorithms rely on the internal division of the IP address into network and local numbers. Routers can summarize the world's network topology because IANA delegates large blocks of numbers to regional IP registries, who in turn allocate variable-size classes of network-numbers to service providers (per policy articulated in RFC

2050). Other blocks are reserved for multicasting (224.0.0.0 and up), broadcasting (set the local part to all 1's), and loopback testing (127.0.0.1, also reserved as the domain name "localhost")

Network Address Translators (NATs) can complicate the picture. Renumbering a whole organization when it outgrows its network class can be a tedious and expensive proposition, so it may 'hide' its entire network behind one or a few gateway machines. Since IP addresses are assumed to be end-to-end unique (one IP interface number per host), NATs can't catch every place in every packet every application protocol might cite an IP address. In practice, applications can 'leak' private IP addresses onto the public Internet, risking routing havoc and other errors.

MAC Address

At the next layer down, the network interface has to translate an IP address to a Multiple Access Control identifier, such as a 48-bit Ethernet station identifier. The MAC namespace is permanently world-unique, since manufacturers must buy 4K blocks from the IEEE Registration Authority for $500 each, plus a $1,250 initiation fee.

On any particular network medium, though, this minature namespace is resolved in the packet driver using the Address Resolution Protocol (ARP) and its complement, Reverse ARP. Rather than flooding the network by periodically announcing one's own IP:MAC binding, there is a reserved query frame type in

Ethernet for this purpose.

Phone Number

Or perhaps this airline ticket's actually being purchased over a dialup connection. Once more, the link layer's address becomes the next lower one's resolvable name. In this case, the Point-to-Point protocol driver needs a phone number to setup a connection over the PSTN. As initially designed by the Bell System in 1947, telephone numbers are another world-unique hierarchical left-to-right namespace, split into country, city, exchange, and subscriber codes.

Past seven digits, their inherent human interface isn't all that great, relying on yellow and white pages at any appreciable scale. Even the written form varies, from ITU standard +1-(626)-806.7574 to the domain name form 4.7.5.7.6.0.8.6.2.6.1.tpc.int. Furthermore, voice is only one of many modern applications, so phone numbers are additionally disambiguated by use (fax, mobile, data, etc).

The North American Numbering Plan Administration (www.nanpa.net) is the authority responsible for renumbering new area codes and allocating the 792 valid exchanges to local carriers. Issuing every new entrant a minimum block of 10,000 lines only aggravates the problem, similar to the IP registrars' fears before the advent of Classless Inter-Domain Routing (CIDR) allowed for network prefixes between 16 and 24 bits wide.

NANPA also reserves application-specific numberspaces, such as the 555 exchange for information services or 800-855 for toll-free teletype access.

Pathname

Our trip down the network stack has only moved us halfway across the original URI. Now we have a connection to an HTTP server, which can resolve the remaining pathname into a renderable Web page. Only the server knows what aliases and file-type extensions and pathname rewriting rules apply to the URI path to yield a filename. The server's operating system, in turn, can resolve that filename to an inode. The filesystem uses inodes to insulate the bits on disk from user-level renaming, moving, linking, and so on. Inodes, in turn, are resolved to physical track and sector addresses by the disk driver's directory (or "map").

Of course, a Web server does more than blindly transfer files; it can also execute processes or interpret scripts. In our airline case, the first component of the URI path specifies a reservation database at the server, but the second is a database key, which names a single record within that database. In the lingo of the airlines' Global Distribution Systems (GDS), a Passenger Name Record (PNR) is created for every reservation enquiry and ticket.

A PNR is a short, opaque alphanumeric string that's particular to that GDS -- but a PNR alone isn't enough to tell if it comes from the Apollo or Sabre namespace, for example (much less which airline it's on!). Furthermore, a PNR name is

needed up until the day of flight, to plan revenue yield, catering, and so on; but for archival post-flight access, a more permanent International Air Transport Association (IATA) 16-digit ticket number names the journey.

Names in HTTP Messages

Once the Web server begins transmitting an HTML page, we can zoom in one last time, down to the actual "bytes on the wire," as we say. Now the HTTP response message format itself looms large. First comes "HTTP/1.1 200 OK", sporting an IANA-registered version number and reply code. Later, we see "Content-Type: text/html", plucked from the Internet Media Type namespace. Also known as MIME types, these entries must be documented with IANA to some degree, or designated private with "x-" or "vnd." prefixes. "Content-Language: en-us" also comes from an IANA list, but it's ultimately dependent on ISO country and language tokens. Similarly with character sets, referenced by IANA-registered atoms like "iso-8859-1" and encodings like "UTF-8", though the actual character sets and encoding algorithms are defined by other bodies.

After the blank line, we're in the midst of the actual hypertext markup, with tags like <HEAD> and <B> flying by from the various HTML tag namespaces defined by W3C over the years. Within a <META> tag, we come across a Platform for Internet Content Selection (PICS) label rating this page. Suppose J.D. Power and Associates defined a ratings schema for airlines: price, meals, timeliness, each on a 1-5 scale. That itself is yet another namespace, of course, but let's dig even

further into the PICS label's fine print.

We find that at the end of the label is J.D. Powers' digital signature verifying that this rating is indeed accurate. Within that data structure we can make out URIs describing which particular cryptographic algorithms were selected, from a space defined by W3C's DSig 1.0 specification.

Ultimately we arrive at Ground Truth: the actual mathematical facts, including the public key.

Zooming Out

But what do we really have here? Is a prime number an address, just one entry in the infinite possibilities for public keys? Or is it a name, a legally binding entry resolvable to J.D. Powers & Associates? All of a sudden, we'll have to zoom all the way out from "bytes on the wire" to "people in organizations and society." Is identity certified through hierarchy or peer-to-peer introductions? Is United's XML tag for <FARE> comprable to Delta's, which happens to include all taxes, too? What about that URI linking to Hertz's rental deal -- will it still be around tomorrow? And what about the names of airports, the syntax of GPS coordinates, or passport numbers?

In the first half of our tour, zooming in on the anatomy of URL has certainly exposed a plethora of namespaces. In the next issue, we'll try to distill which of these are uniquely Internet-scale challenges: namespaces that scale across time, into the future; across space, into uncountable digital nooks and crannies; and

across organizational boundaries, negotiating the meaning of each name in lieu of "universal" enforced standards.

- - - -

For more detail on the namespaces introduced here (and a sneak preview), companion slides for this article are available at http://www.ics.uci.edu/IRUS/twist/twist99/presentations/khare/ -- as well as a dozen other speakers' presentations and detailed minutes from the workshop.

Part II, November/December 99

Naming is a Choice

"Any problem in computer science can be solved with another layer of indirection" -- but I couldn't tell you the name of this design proverb's author! I heard it from Butler Lampson, whose own Turing Award lecture cites David Wheeler of EDSAC fame; but another MIT professor pointed to Alan Perlis, founder of CMU's computer science program. Searching the Web yields many more occurrences of the aphorism -- all unattributed! So I am left to make a trust decision for my binding of "author's name."

Resolving that quote to "David Wheeler" seems nothing like resolving "w3.org" to 18.29.0.27; the former process weighs human relationships, history, and judgment, while the latter mechanically queries a Domain Name System (DNS)

database. Dig deeper, though, I believe they're seem more similar than not. Every decision to name something is a trust decision, resolvable only in the context of some community that agrees to that namespace.

Perhaps the comparison will make more sense in reverse: why do we name objects in the first place? We use names to abstract away details: of location, of authorization, of human-readability. Every time we indirect through another layer of naming, we interpose a new fulcrum for administrative leverage, to redirect the binding, extract rents, and implement other policies.

In particular, namespace management at Internet scale requires more than scalable algorithms alone. Internet scale is additionally about scaling across time, space, and organizations -- raising unique issues of longevity, latency, and liability, respectively.

Social Namespaces

X.500 Directory Hierarchy

PGP Web of Trust

XML Tag Namespaces

Internet-Scale Properties

Segue on xml namespaces

Misc table

Across Time

Across Space

Across Organizations

<table: even more isns>

Postmodernism Perspective

Link to other workshop speakers

URI history, etc.

 About Amaya

- Home page
- Overview
- Release history
- Amaya in the Press

 Download Amaya

- Distributions
- Source
- RPMs
- CVS
- Latest CVS changes
- Bugs and Wishlist

 Mailing lists

- www-amaya
- www-amaya-dev
- www-amaya-doc
- amaya@ml.free.fr

 Documentation

- Users and Developers
- FAQ

 Project Contributors

- Contributors
- How to contribute
- Open projects

Welcome to Amaya

W3C's Editor/Browser

Amaya is a Web editor, i.e. a tool used to create and update documents directly on the Web. Browsing features are seamlessly integrated with the editing and remote access features in a uniform environment. This follows the original vision of the Web as a space for collaboration and not just a one-way publishing medium.

Work on Amaya started at W3C in 1996 to showcase Web technologies in a fully-featured Web client. The main motivation for developing Amaya was to provide a framework that can integrate as many W3C technologies as possible. It is used to demonstrate these technologies in action while taking advantage of their combination in a single, consistent environment.

Amaya started as an HTML + CSS style sheets editor. Since that time it was extended to support XML and an increasing number of XML applications such as the XHTML family, MathML, and SVG. It allows all those vocabularies to be edited simultaneously in compound documents.

Amaya includes a collaborative annotation application based on Resource Description Framework (RDF), XLink, and XPointer. Visit the Annotea project home page.

Amaya - Open Source

Amaya is an open source software project hosted by W3C. You are invited to contribute in many forms (documentation, translation, writing code, fixing bugs, porting to other platforms...).

The Amaya software is written in C and is available for Windows, Unix platforms and MacOSX.

Amaya Team

The application is jointly developed by W3C and the WAM (Web, Adaptation and Multimedia) project at INRIA. The core team includes: Irène Vatton (Project lead, INRIA), Laurent Carcone (W3C), Stéphane Gully (INRIA), Vincent Quint (INRIA). José Kahan (W3C Semantic Web Advanced Development project) develops an Annotea and Annotea Bookmarks support in Amaya.

 **Amaya 8.5 is now available**

Current Release

The current release, Amaya 8.5, supports HTML 4.01, XHTML 1.0, XHTML Basic, XHTML

1.1, HTTP 1.1, MathML 2.0, many CSS 2 features, and includes SVG support (transformation, transparency, and SMIL animation on OpenGL platforms). You can display and partially edit XML documents. It's an internationalized application.

See the Overview page for more details.

- Download Amaya 8.5 public release (27 April 2004)
- Download the source code of Amaya 8.5 (27 April 2004)
- Checkout from the CVS base
- RPM distributions are also available at http://rpmfind.net/linux/rpm2html/search.php?query=amaya
- Debian package distribution: http://packages.debian.org/stable/web/amaya.html
- Installing or compiling Amaya MacOSX

Future Releases

A next release should be available current July.

Amaya is covered by the W3C Software Notice and License. The icon 

can be inserted in your Web pages when they are created and edited by Amaya.

Irène Vatton
$Date: 2004/05/03 15:17:51 $

XHTML Dokument, mit MathML- und SVG-Inhalten

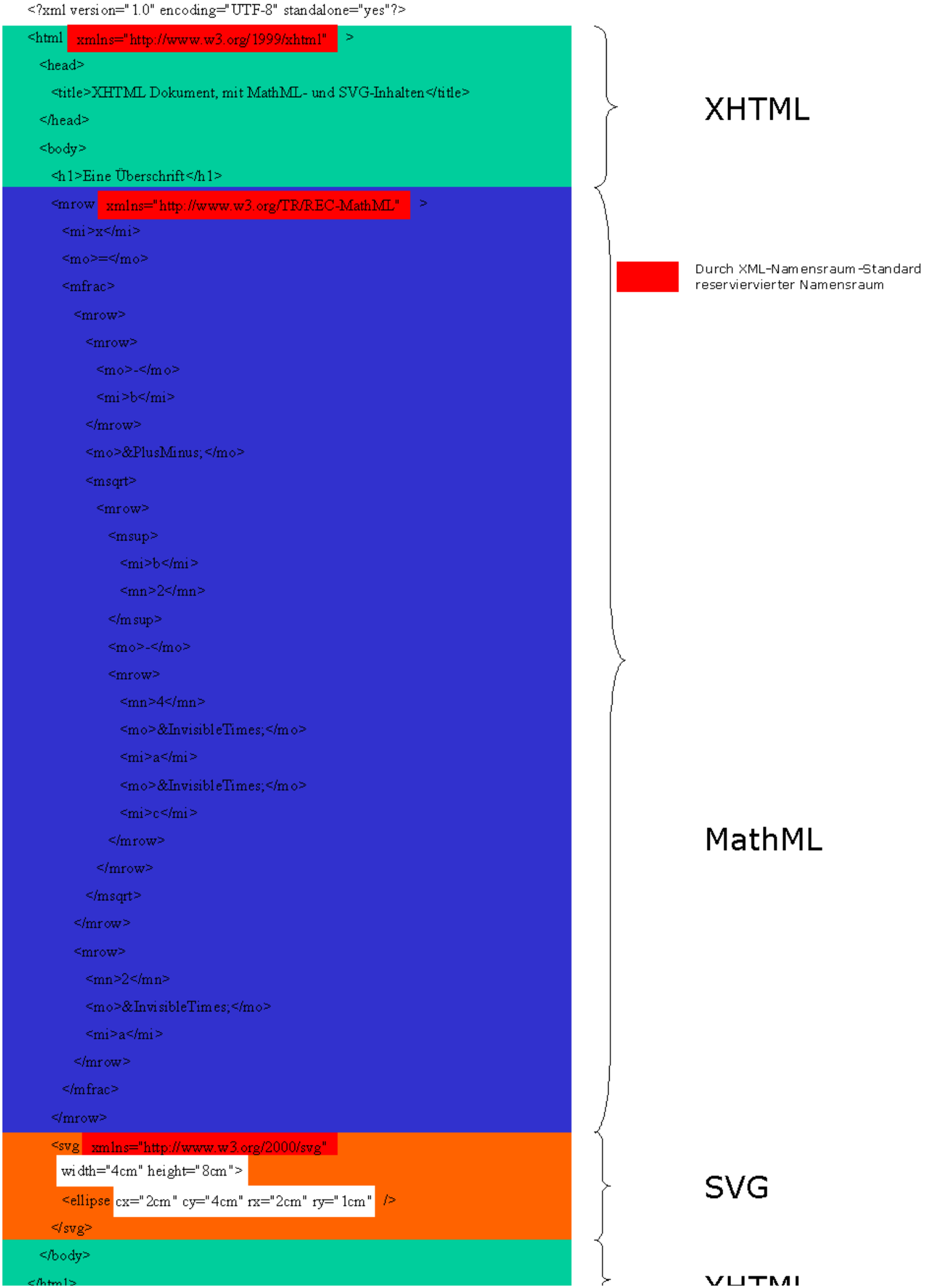
File Edit Types Links Views Style Special Attributes Annotations Help

Open http://www.jeckle.de/vorlesung/xml/examples/namespace14.xml

Eine Überschrift

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Text \ h1 \ body \ html \ Document


<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<title>XHTML Dokument, mit MathML- und SVG-Inhalten</title>
</head>
<body>
<h1>Eine Überschrift</h1>
<mrow xmlns="http://www.w3.org/TR/REC-MathML" >
<mi>x</mi>
<mo>=</mo>
<mfrac>
<mrow>
<mrow>
<mo>-</mo>
<mi>b</mi>
</mrow>
<mo>±</mo>
<msqrt>
<mrow>
<msup>
<mi>b</mi>
<mn>2</mn>
</msup>
<mo>-</mo>
<mrow>
<mn>4</mn>
<mo>⁢</mo>
<mi>a</mi>
<mo>⁢</mo>
<mi>c</mi>
</mrow>
</mrow>
</msqrt>
</mrow>
<mrow>
<mn>2</mn>
<mo>⁢</mo>
<mi>a</mi>
</mrow>
</mfrac>
</mrow>
<svg xmlns="http://www.w3.org/2000/svg"
width="4cm" height="8cm">
<ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm" />
</svg>
</body>
</html>
XHTML
Durch XML-Namensraum-Standard
reserviervierter Namensraum
MathML
SVG
XHTML

```
</html>
```

XHTML

Extensible Markup Language (XML) 1.1

W3C Recommendation 04 February 2004, edited in place 15 April 2004

This version:
http://www.w3.org/TR/2004/REC-xml11-20040204/
Latest version:
http://www.w3.org/TR/xml11
Previous version:
http://www.w3.org/TR/2003/PR-xml11-20031105/
Editors:
Tim Bray, Textuality and Netscape <tbray@textuality.com>
Jean Paoli, Microsoft <jeanpa@microsoft.com>
C. M. Sperberg-McQueen, W3C <cmsmcq@w3.org>
Eve Maler, Sun Microsystems, Inc. <eve.maler@east.sun.com>
François Yergeau <fyergeau@alis.com>
John Cowan <cowan@ccil.org>

Please refer to the **errata** for this document, which may include some normative corrections.

This document is also available in these non-normative formats: XML and XHTML with color-coded revision indicators.

See also **translations**.

Abstract

The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This document is a Recommendation of the W3C. It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document specifies a syntax created by subsetting an existing, widely used international text processing standard (Standard Generalized Markup Language, ISO 8879:1986(E) as amended and corrected) for use on the World Wide Web. It is a product of the W3C XML Activity.

On 15 April 2004, this document was edited in place to add two missing spaces to production [1] in section 2.1

The English version of this specification is the only normative version. However, for translations of this document, see http://www.w3.org/2003/03/Translations/byTechnology?technology=xml11.

Documentation of intellectual property possibly relevant to this recommendation may be found at the Working Group's public IPR disclosure page.

An implementation report for XML 1.1 is available at http://www.w3.org/XML/2002/09/xml11-implementation.html.

Please report errors in this document to xml-editor@w3.org; archives are available. The errata list for this edition is available at http://www.w3.org/XML/xml-V11-1e-errata.

A Test Suite is maintained to help assessing conformance to this specification.

Table of Contents

Appendices

1 Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

[Definition: A software module called an **XML processor** is used to read XML documents and provide access to their content and structure.] [Definition: It is assumed that an XML processor is doing its work on behalf of another module, called the **application**.] This specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C. The membership of the XML Working Group is given in an appendix. Dan Connolly served as the Working Group's contact with the W3C.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

This specification, together with associated standards (Unicode [Unicode] and ISO/IEC 10646 [ISO/IEC 10646] for characters, Internet RFC 3066 [IETF RFC 3066] for language identification tags, ISO 639 [ISO 639] for language name codes, and ISO 3166 [ISO 3166] for country name codes), provides all the information necessary to understand XML Version 1.1 and construct computer programs to process it.

This version of the XML specification may be distributed freely, as long as all text and legal notices remain intact.

1.2 Terminology

The terminology used to describe XML documents is defined in the body of this specification. The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when EMPHASIZED, are to be interpreted as described in [IETF RFC 2119]. In addition, the terms defined in the following list are used in building those definitions and in describing the actions of an XML processor:

error

> [Definition: A violation of the rules of this specification; results are undefined. Unless otherwise specified, failure to observe a prescription of this specification indicated by one of the keywords MUST, REQUIRED, MUST NOT, SHALL and SHALL NOT is an error. Conforming software MAY detect and report an error and MAY recover from it.]

fatal error

> [Definition: An error which a conforming XML processor MUST detect and report to the application. After encountering a fatal error, the processor MAY continue processing the data to search for further errors and MAY report such errors to the application. In order to support correction of errors, the processor MAY make unprocessed data from the document (with intermingled character data and markup) available to the application. Once a fatal error is detected, however, the processor MUST NOT continue normal processing (i.e., it MUST NOT continue to pass character data and information about the document's logical structure to the application in the normal way).]

at user option

> [Definition: Conforming software MAY or MUST (depending on the modal verb in the sentence) behave as described; if it does, it MUST provide users a means to enable or disable the behavior described.]

validity constraint

> [Definition: A rule which applies to all valid XML documents. Violations of validity constraints are errors; they MUST, at user option, be reported by validating XML processors.]

well-formedness constraint

> [Definition: A rule which applies to all well-formed XML documents. Violations of well-formedness constraints are fatal errors.]

match

[Definition: (Of strings or names:) Two strings or names being compared MUST be identical. Characters with multiple possible representations in Unicode (e.g. characters with both precomposed and base+diacritic forms) match only if they have the same representation in both strings. No case folding is performed. (Of strings and rules in the grammar:) A string matches a grammatical production if it belongs to the language generated by that production. (Of content and content models:) An element matches its declaration when it conforms in the fashion described in the constraint **[VC: Element Valid]**.]

for compatibility

[Definition: Marks a sentence describing a feature of XML included solely to ensure that XML remains compatible with SGML.]

for interoperability

[Definition: Marks a sentence describing a non-binding recommendation included to increase the chances that XML documents can be processed by the existing installed base of SGML processors which predate the WebSGML Adaptations Annex to ISO 8879.]

1.3 Rationale and list of changes for XML 1.1

The W3C's XML 1.0 Recommendation was first issued in 1998, and despite the issuance of many errata culminating in a Third Edition of 2004, has remained (by intention) unchanged with respect to what is well-formed XML and what is not. This stability has been extremely useful for interoperability. However, the Unicode Standard on which XML 1.0 relies for character specifications has not remained static, evolving from version 2.0 to version 4.0 and beyond. Characters not present in Unicode 2.0 may already be used in XML 1.0 character data. However, they are not allowed in XML names such as element type names, attribute names, enumerated attribute values, processing instruction targets, and so on. In addition, some characters that should have been permitted in XML names were not, due to oversights and inconsistencies in Unicode 2.0.

The overall philosophy of names has changed since XML 1.0. Whereas XML 1.0 provided a rigid definition of names, wherein everything that was not permitted was forbidden, XML 1.1 names are designed so that everything that is not forbidden (for a specific reason) is permitted. Since Unicode will continue to grow past version 4.0, further changes to XML can be avoided by allowing almost any character, including those not yet assigned, in names.

In addition, XML 1.0 attempts to adapt to the line-end conventions of various modern operating systems, but discriminates against the conventions used on IBM and IBM-compatible mainframes. As a result, XML documents on mainframes are not plain text files according to the local conventions. XML 1.0 documents generated on mainframes must either violate the local line-end conventions, or employ otherwise unnecessary translation phases before parsing and after generation. Allowing straightforward interoperability is particularly important when data stores are shared between mainframe and non-mainframe systems (as opposed to being copied from one to the other). Therefore XML 1.1 adds NEL (#x85) to the list of line-end characters. For completeness, the Unicode line separator character, #x2028, is also supported.

Finally, there is considerable demand to define a standard representation of arbitrary Unicode characters in XML documents. Therefore, XML 1.1 allows the use of character references to the control characters #x1 through #x1F, most of which are forbidden in XML 1.0. For reasons of robustness, however, these characters still cannot be used directly in documents. In order to improve the robustness of character encoding detection, the additional control characters #x7F through #x9F, which were freely allowed in XML 1.0 documents, now must also appear only as character references. (Whitespace characters are of course exempt.) The minor sacrifice of backward compatibility is considered not significant. Due to potential problems with APIs, #x0 is still forbidden both directly and as a character reference.

Finally, XML 1.1 defines a set of constraints called "full normalization" on XML documents, which document creators SHOULD adhere to, and document processors SHOULD verify. Using fully normalized documents ensures that identity comparisons of names, attribute values, and character content can be made correctly by simple binary comparison of Unicode strings.

A new XML version, rather than a set of errata to XML 1.0, is being created because the changes affect the definition of well-formed documents. XML 1.0 processors must continue to reject documents that contain new characters in XML names, new line-end conventions, and references to control characters. The distinction between XML 1.0 and XML 1.1 documents is indicated by the version number information in the XML declaration at the start of each document.

2 Documents

[Definition: A data object is an **XML document** if it is well-formed, as defined in this specification. A well-formed XML document MAY in addition be valid if it meets certain further constraints.]

Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity MAY refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The logical and physical structures MUST nest properly, as described in **4.3.2 Well-Formed Parsed Entities**.

2.1 Well-Formed XML Documents

[Definition: A textual object is a **well-formed** XML document if:]

1. Taken as a whole, it matches the production labeled document.
2. It meets all the well-formedness constraints given in this specification.
3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed.

Document

```
[1]  document  ::=  prolog element Misc* - Char* RestrictedChar Char*
```

Matching the document production implies that:

1. It contains one or more elements.
2. [Definition: There is exactly one element, called the **root**, or document element, no part of which appears in the content of any other element.] For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

[Definition: As a consequence of this, for each non-root element `C` in the document, there is one other element `P` in the document such that `C` is in the content of `P`, but is not in the content of any other element that is in the content of `P`. `P` is referred to as the **parent** of `C`, and `C` as a **child** of `P`.]

2.2 Characters

[Definition: A parsed entity contains **text**, a sequence of characters, which may represent markup or character data.] [Definition: A **character** is an atomic unit of text as specified by ISO/IEC 10646 [ISO/IEC 10646]. Legal characters are tab, carriage return, line feed, and the legal characters of Unicode and ISO/IEC 10646. The versions of these standards cited in **A.1 Normative References** were current at the time this document was prepared. New characters may be added to these standards by amendments or new editions. Consequently, XML processors MUST accept any character in the range specified for Char.]

Character Range

```
[2]   Char            ::=  [#x1-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]   /* any Unicode character, excluding the surrogate blocks, FFFE, and FFFF. */
[2a]  RestrictedChar  ::=  [#x1-#x8] | [#xB-#xC] | [#xE-#x1F] | [#x7F-#x84] | [#x86-#x9F]
```

The mechanism for encoding character code points into bit patterns MAY vary from entity to entity. All XML processors MUST accept the UTF-8 and UTF-16 encodings of Unicode [Unicode]; the mechanisms for signaling which of the two is in use, or for bringing other encodings into play, are discussed later, in **4.3.3 Character Encoding in Entities**.

Note:

Document authors are encouraged to avoid "compatibility characters", as defined in Unicode [Unicode]. The characters defined in the following ranges are also discouraged. They are either control characters or permanently undefined Unicode characters:

```
[#x7F-#x84], [#x86-#x9F], [#xFDD0-#xFDDF],
[#1FFFE-#x1FFFF], [#2FFFE-#x2FFFF], [#3FFFE-#x3FFFF],
[#4FFFE-#x4FFFF], [#5FFFE-#x5FFFF], [#6FFFE-#x6FFFF],
[#7FFFE-#x7FFFF], [#8FFFE-#x8FFFF], [#9FFFE-#x9FFFF],
[#AFFFE-#xAFFFF], [#BFFFE-#xBFFFF], [#CFFFE-#xCFFFF],
[#DFFFE-#xDFFFF], [#EFFFE-#xEFFFF], [#FFFFE-#xFFFFF],
[#10FFFE-#x10FFFF].
```

2.3 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.

S (white space) consists of one or more space (#x20) characters, carriage returns, line feeds, or tabs.

White Space

```
[3]  S  ::=  (#x20 | #x9 | #xD | #xA)+
```

Note:

The presence of #xD in the above production is maintained purely for backward compatibility with the First Edition. As explained in **2.11 End-of-Line Handling**, all #xD characters literally present in an XML document are either removed or replaced by #xA characters before any other processing is done. The only way to get a #xD character to match this production is to use a character reference in an entity value literal.

[Definition: A **Name** is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters.] Names beginning with the string "`xml`", or with any string which would match `(('X'|'x') ('M'|'m') ('L'|'l'))`, are reserved for standardization in this or future versions of this specification.

Note:

The Namespaces in XML Recommendation [XML Names] assigns a meaning to names containing colon characters. Therefore, authors should not use the colon in XML names except for namespace purposes, but XML processors must accept the colon as a name character.

An Nmtoken (name token) is any mixture of name characters.

The first character of a Name MUST be a NameStartChar, and any other characters MUST be NameChars; this mechanism is used to prevent names from beginning with European (ASCII) digits or with basic combining characters. Almost all characters are permitted in names, except those which either are or reasonably could be used as delimiters. The intention is to be inclusive rather than exclusive, so that writing systems not yet encoded in Unicode can be used in XML names. See **I Suggestions for XML Names** for suggestions on the creation of names.

Document authors are encouraged to use names which are meaningful words or combinations of words in natural languages, and to avoid symbolic or white space characters in names. Note that COLON, HYPHEN-MINUS, FULL STOP (period), LOW LINE (underscore), and MIDDLE DOT are explicitly permitted.

The ASCII symbols and punctuation marks, along with a fairly large group of Unicode symbol characters, are excluded from names because they are more useful as delimiters in contexts where XML names are used outside XML documents; providing this group gives those contexts hard guarantees about what *cannot* be part of an XML name. The character #x037E, GREEK QUESTION MARK, is excluded because when normalized it becomes a semicolon, which could change the meaning of entity references.

Names and Tokens

```
[4]   NameStartChar  ::=  ":" | [A-Z] | "_" | [a-z] | [#xC0-#xD6] | [#xD8-#xF6] | [#xF8-#x2FF] | [#x370-
                          #x37D] | [#x37F-#x1FFF] | [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF] |
                          [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]
[4a]  NameChar       ::=  NameStartChar | "-" | "." | [0-9] | #xB7 | [#x0300-#x036F] | [#x203F-#x2040]
[5]   Name           ::=  NameStartChar (NameChar)*
[6]   Names          ::=  Name (#x20 Name)*
[7]   Nmtoken        ::=  (NameChar)+
[8]   Nmtokens       ::=  Nmtoken (#x20 Nmtoken)*
```

Note:

The Names and Nmtokens productions are used to define the validity of tokenized attribute values after normalization (see **3.3.1 Attribute Types**).

Literal data is any quoted string not containing the quotation mark used as a delimiter for that string. Literals are used for specifying the content of internal entities (EntityValue), the values of attributes (AttValue), and external identifiers (SystemLiteral). Note that a SystemLiteral can be parsed without scanning for markup.

Literals

```
[9]   EntityValue    ::=  '"' ([^%&"] | PEReference | Reference)* '"'
                          |  "'" ([^%&'] | PEReference | Reference)* "'"
[10]  AttValue       ::=  '"' ([^<&"] | Reference)* '"'
                          |  "'" ([^<&'] | Reference)* "'"
[11]  SystemLiteral  ::=  ('"' [^"]* '"') | ("'" [^']* "'")
[12]  PubidLiteral   ::=  '"' PubidChar* '"' | "'" (PubidChar - "'")* "'"
[13]  PubidChar      ::=  #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]
```

Note:

Although the EntityValue production allows the definition of a general entity consisting of a single explicit < in the literal (e.g., `<!ENTITY mylt "<">`), it is strongly advised to avoid this practice since any reference to that entity will cause a well-formedness error.

2.4 Character Data and Markup

Text consists of intermingled character data and markup. [Definition: **Markup** takes the form of start-tags, end-tags, empty-element tags, entity references, character references, comments, CDATA section delimiters, document type declarations, processing instructions, XML declarations, text declarations, and any white space that is at the top level of the document entity (that is, outside the document element and not inside any other markup).]

[Definition: All text that is not markup constitutes the **character data** of the document.]

The ampersand character (&) and the left angle bracket (<) MUST NOT appear in their literal form, except when used as markup delimiters, or within a comment, a processing instruction, or a CDATA section. If they are needed elsewhere, they MUST be escaped using either numeric character references or the strings "`&`" and "`<`" respectively. The right angle bracket (>) MAY be represented using the string "`>`", and MUST, for compatibility, be escaped using either "`>`" or a character reference when it appears in the string "`]]>`" in content, when that string is not marking the end of a CDATA section.

In the content of elements, character data is any string of characters which does not contain the start-delimiter of any markup or the CDATA-section-close delimiter, "`]]>`". In a CDATA section, character data is any string of characters not including the CDATA-section-close delimiter.

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') MAY be represented as "`&apos;`", and the double-quote character (") as "`&quot;`".

Character Data

```
[14]  CharData  ::=  [^<&]* - ([^<&]* ']]>' [^<&]*)
```

2.5 Comments

[Definition: **Comments** MAY appear anywhere in a document outside other markup; in addition, they MAY appear within the document type declaration at places allowed by the grammar. They are not part of the document's character data; an XML processor MAY, but need not, make it possible for an application to retrieve the text of comments. For compatibility, the string "`--`" (double-hyphen) MUST NOT occur within comments.] Parameter entity references MUST NOT be recognized within comments.

Comments

```
[15]  Comment  ::=  '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

An example of a comment:

```
<!-- declarations for <head> & <body> -->
```

Note that the grammar does not allow a comment ending in `--->`. The following example is *not* well-formed.

```
<!-- B+, B, or B--->
```

2.6 Processing Instructions

[Definition: **Processing instructions** (PIs) allow documents to contain instructions for applications.]

Processing Instructions

```
[16]  PI        ::=  '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17]  PITarget  ::=  Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

PIs are not part of the document's character data, but MUST be passed through to the application. The PI begins with a target (PITarget) used to identify the application to which the instruction is directed. The target names "`XML`", "`xml`", and so on are reserved for standardization in this or future versions of this specification. The XML Notation mechanism MAY be used for formal declaration of PI targets. Parameter entity references MUST NOT be recognized within processing instructions.

2.7 CDATA Sections

[Definition: **CDATA sections** MAY occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup. CDATA sections begin with the string "`<![CDATA[`" and end with the string "`]]>`":]

CDATA Sections

```
[18]  CDSect   ::=  CDStart CData CDEnd
[19]  CDStart  ::=  '<![CDATA['
[20]  CData    ::=  (Char* - (Char* ']]>'
                    Char*))
[21]  CDEnd    ::=  ']]>'
```

Within a CDATA section, only the CDEnd string is recognized as markup, so that left angle brackets and ampersands may occur in their literal form; they need not (and cannot) be escaped using "`<`" and "`&`". CDATA sections cannot nest.

An example of a CDATA section, in which "`<greeting>`" and "`</greeting>`" are recognized as character data, not markup:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

2.8 Prolog and Document Type Declaration

[Definition: XML 1.1 documents MUST begin with an **XML declaration** which specifies the version of XML being used.] For example, the following is a complete XML 1.1 document, well-formed but not valid:

```
<?xml version="1.1"?>
<greeting>Hello, world!</greeting>
```

but the following is an XML 1.0 document because it does not have an XML declaration:

```
<greeting>Hello, world!</greeting>
```

The function of the markup in an XML document is to describe its storage and logical structure and to associate attribute name-value pairs with its logical structures. XML provides a mechanism, the document type declaration, to define constraints on the logical structure and to support the use of predefined storage units. [Definition: An XML document is **valid** if it has an associated document type declaration and if the document complies with the constraints expressed in it.]

The document type declaration MUST appear before the first element in the document.

Prolog

```
[22]  prolog        ::=  XMLDecl Misc* (doctypedecl Misc*)?
[23]  XMLDecl       ::=  '<?xml' VersionInfo EncodingDecl? SDDecl? S?'?>'
[24]  VersionInfo   ::=  S 'version' Eq ("'" VersionNum "'" | '"' VersionNum '"')
[25]  Eq            ::=  S? '=' S?
[26]  VersionNum    ::=  '1.1'
[27]  Misc          ::=  Comment | PI | S
```

[Definition: The XML **document type declaration** contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or **DTD**. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. The DTD for a document consists of both subsets taken together.]

[Definition: A **markup declaration** is an element type declaration, an attribute-list declaration, an entity declaration, or a notation declaration.] These declarations MAY be contained in whole or in part within parameter entities, as described in the well-formedness and validity constraints below. For further information, see **4 Physical Structures**.

Document Type Definition

```
[28]   doctypedecl  ::=  '<!DOCTYPE' S Name (S ExternalID)? S? ('[' intSubset ']'
                         S?)? '>'                                  [VC: Root Element Type]
                                                                   [WFC: External Subset]
[28a]  DeclSep      ::=  PEReference | S                          [WFC: PE Between Declarations]
[28b]  intSubset    ::=  (markupdecl | DeclSep)*
[29]   markupdecl   ::=  elementdecl | AttlistDecl | EntityDecl | NotationDecl |
                         PI | Comment                            [VC: Proper Declaration/PE Nesting]
                                                                   [WFC: PEs in Internal Subset]
```

Note that it is possible to construct a well-formed document containing a doctypedecl that neither points to an external subset nor contains an internal subset.

The markup declarations MAY be made up in whole or in part of the replacement text of parameter entities. The productions later in this specification for individual nonterminals (elementdecl, AttlistDecl, and so on) describe the declarations *after* all the parameter entities have been included.

Parameter entity references are recognized anywhere in the DTD (internal and external subsets and external parameter entities), except in literals, processing instructions, comments, and the contents of ignored conditional sections (see **3.4 Conditional Sections**). They are also recognized in entity value literals. The use of parameter entities in the internal subset is restricted as described below.

Validity constraint: Root Element Type

The Name in the document type declaration MUST match the element type of the root element.

Validity constraint: Proper Declaration/PE Nesting

Parameter-entity replacement text MUST be properly nested with markup declarations. That is to say, if either the first character or the last character of a markup declaration (markupdecl above) is contained in the replacement text for a parameter-entity reference, both MUST be contained in the same replacement text.

Well-formedness constraint: PEs in Internal Subset

In the internal DTD subset, parameter-entity references MUST NOT occur within markup declarations; they MAY occur where markup declarations can occur. (This does not apply to references that occur in external parameter entities or to the external subset.)

Well-formedness constraint: External Subset

The external subset, if any, MUST match the production for extSubset.

Well-formedness constraint: PE Between Declarations

The replacement text of a parameter entity reference in a DeclSep MUST match the production extSubsetDecl.

Like the internal subset, the external subset and any external parameter entities referenced in a DeclSep MUST consist of a series of complete markup declarations of the types allowed by the non-terminal symbol markupdecl, interspersed with white space or parameter-entity references. However, portions of the contents of the external subset or of these external parameter entities MAY conditionally be ignored by using the conditional section construct; this is not allowed in the internal subset but is allowed in external parameter entities referenced in the internal subset.

External Subset

```
[30]  extSubset      ::=  TextDecl? extSubsetDecl
[31]  extSubsetDecl  ::=  ( markupdecl | conditionalSect | DeclSep)*
```

The external subset and external parameter entities also differ from the internal subset in that in them, parameter-entity references are permitted *within* markup declarations, not only *between* markup declarations.

An example of an XML document with a document type declaration:

```
<?xml version="1.1"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

The system identifier "`hello.dtd`" gives the address (a URI reference) of a DTD for the document.

The declarations can also be given locally, as in this example:

```
<?xml version="1.1" encoding="UTF-8" ?>
<!DOCTYPE greeting [
<!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

If both the external and internal subsets are used, the internal subset MUST be considered to occur before the external subset. This has the effect that entity and attribute-list declarations in the internal subset take precedence over those in the external subset.

XML 1.1 processors SHOULD accept XML 1.0 documents as well. If a document is well-formed or valid XML 1.0, and provided it does not contain any control characters in the range [#x7F-#x9F] other than as character escapes, it may be made well-formed or valid XML 1.1 respectively simply by changing the version number.

2.9 Standalone Document Declaration

Markup declarations can affect the content of the document, as passed from an XML processor to an application; examples are attribute defaults and entity declarations. The standalone document declaration, which MAY appear as a component of the XML declaration, signals whether or not there are such declarations which appear external to the document entity or in parameter entities. [Definition: An **external markup declaration** is defined as a markup declaration occurring in the external subset or in a parameter entity (external or internal, the latter being included because non-validating processors are not required to read them).]

Standalone Document Declaration

```
[32]  SDDecl  ::=  #x20+ 'standalone' Eq (("'" ('yes' | 'no') "'") |
                   ('"' ('yes' | 'no') '"'))
```

[VC: Standalone Document Declaration]

In a standalone document declaration, the value "yes" indicates that there are no external markup declarations which affect the information passed from the XML processor to the application. The value "no" indicates that there are or may be such external markup declarations. Note that the standalone document declaration only denotes the presence of external *declarations*; the presence, in a document, of references to external *entities*, when those entities are internally declared, does not change its standalone status.

If there are no external markup declarations, the standalone document declaration has no meaning. If there are external markup declarations but there is no standalone document declaration, the value "no" is assumed.

Any XML document for which `standalone="no"` holds can be converted algorithmically to a standalone document, which may be desirable for some network delivery applications.

Validity constraint: Standalone Document Declaration

The standalone document declaration MUST have the value "no" if any external markup declarations contain declarations of:

- attributes with default values, if elements to which these attributes apply appear in the document without specifications of values for these attributes, or
- entities (other than `amp`, `lt`, `gt`, `apos`, `quot`), if references to those entities appear in the document, or
- attributes with tokenized types, where the attribute appears in the document with a value such that *normalization* will produce a different value from that which would be produced in the absence of the declaration, or
- element types with element content, if white space occurs directly within any instance of those types.

An example XML declaration with a standalone document declaration:

```
<?xml version="1.1" standalone='yes'?>
```

2.10 White Space Handling

In editing XML documents, it is often convenient to use "white space" (spaces, tabs, and blank lines) to set apart the markup for greater readability. Such white space is typically not intended for inclusion in the delivered version of the document. On the other hand, "significant" white space that should be preserved in the delivered version is common, for example in poetry and source code.

An XML processor MUST always pass all characters in a document that are not markup through to the application. A validating XML processor MUST also inform the application which of these characters constitute white space appearing in element content.

A special attribute named `xml:space` MAY be attached to an element to signal an intention that in that element, white space should be preserved by applications. In valid documents, this attribute, like any other, MUST be declared if it is used. When declared, it MUST be given as an enumerated type whose values are one or both of "default" and "preserve". For example:

```
<!ATTLIST poem  xml:space (default|preserve) 'preserve'>
<!ATTLIST pre xml:space (preserve) #FIXED 'preserve'>
```

The value "default" signals that applications' default white-space processing modes are acceptable for this element; the value "preserve" indicates the intent that applications preserve all the white space. This declared intent is considered to apply to all elements within the content of the element where it is specified, unless overridden with another instance of the `xml:space` attribute. This specification does not give meaning to any value of `xml:space` other than "default" and "preserve". It is an error for other values to be specified; the XML processor MAY report the error or MAY recover by ignoring the attribute specification or by reporting the (erroneous) value to the application. Applications may ignore or reject erroneous values.

The root element of any document is considered to have signaled no intentions as regards application space handling, unless it provides a value for this attribute or the attribute is declared with a default value.

2.11 End-of-Line Handling

XML parsed entities are often stored in computer files which, for editing convenience, are organized into lines. These lines are typically separated by some combination of the characters CARRIAGE RETURN (#xD) and LINE FEED (#xA).

To simplify the tasks of applications, the XML processor MUST behave as if it normalized all line breaks in external parsed entities (including the document entity) on input, before parsing, by translating all of the following to a single #xA character:

1. the two-character sequence #xD #xA
2. the two-character sequence #xD #x85
3. the single character #x85
4. the single character #x2028
5. any #xD character that is not immediately followed by #xA or #x85.

The characters #x85 and #x2028 cannot be reliably recognized and translated until an entity's encoding declaration (if present) has been read. Therefore, it is a fatal error to use them within the XML declaration or text declaration.

2.12 Language Identification

In document processing, it is often useful to identify the natural or formal language in which the content is written. A special attribute named `xml:lang` MAY be inserted in documents to specify the language used in the contents and attribute values of any element in an XML document. In valid documents, this attribute, like any other, MUST be declared if it is used. The values of the attribute are language identifiers as defined by [IETF RFC 3066], *Tags for the Identification of Languages*, or its successor; in addition, the empty string MAY be specified.

(Productions 33 through 38 have been removed.)

For example:

```
<p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
<l>Habe nun, ach! Philosophie,</l>
<l>Juristerei, und Medizin</l>
<l>und leider auch Theologie</l>
<l>durchaus studiert mit hei&#xDF;em Bem&#xFC;h'n.</l>
</sp>
```

The intent declared with `xml:lang` is considered to apply to all attributes and content of the element where it is specified, unless overridden with an instance of `xml:lang` on another element within that content. In particular, the empty value of `xml:lang` is used on an element B to override a specification of `xml:lang` on an enclosing element A, without specifying another language. Within B, it is considered that there is no language information available, just as if `xml:lang` had not been specified on B or any of its ancestors.

Note:

Language information may also be provided by external transport protocols (e.g. HTTP or MIME). When available, this information may be used by XML applications, but the more local information provided by `xml:lang` should be considered to override it.

A simple declaration for `xml:lang` might take the form

```
xml:lang CDATA #IMPLIED
```

but specific default values MAY also be given, if appropriate. In a collection of French poems for English students, with glosses and notes in English, the `xml:lang` attribute might be declared this way:

```
<!ATTLIST poem   xml:lang CDATA 'fr'>
<!ATTLIST gloss  xml:lang CDATA 'en'>
<!ATTLIST note   xml:lang CDATA 'en'>
```

2.13 Normalization Checking

All XML parsed entities (including document entities) SHOULD be fully normalized as per the definition of **B Definitions for Character Normalization** supplemented by the following definitions of *relevant constructs* for XML:

1. The replacement text of all parsed entities
2. All text matching, in context, one of the following productions:
    a. CData
    b. CharData
    c. content
    d. Name
    e. Nmtoken

However, a document is still well-formed even if it is not fully normalized. XML processors SHOULD provide a user option to verify that the document being processed is in fully normalized form, and report to the application whether it is or not. The option to not verify SHOULD be chosen only when the input text is certified, as defined by **B Definitions for Character Normalization**.

The verification of full normalization MUST be carried out as if by first verifying that the entity is in include-normalized form as defined by **B Definitions for Character Normalization** and by then verifying that none of the relevant constructs listed above begins (after character references are expanded) with a composing character as defined by **B Definitions for Character Normalization**. Non-validating processors MUST ignore possible denormalizations that would be caused by inclusion of external entities that they do not read.

> **Note:**
>
> The composing character are all Unicode characters of non-zero combining class, plus a small number of class-zero characters that nevertheless take part as a non-initial character in certain Unicode canonical decompositions. Since these characters are meant to follow base characters, restricting relevant constructs (including content) from beginning with a composing character does not meaningfully diminish the expressiveness of XML.

If, while verifying full normalization, a processor encounters characters for which it cannot determine the normalization properties (i.e., characters introduced in a version of Unicode [Unicode] later than the one used in the implementation of the processor), then the processor MAY, at user option, ignore any possible denormalizations caused by these characters. The option to ignore those denormalizations SHOULD NOT be chosen by applications when reliability or security are critical.

XML processors MUST NOT transform the input to be in fully normalized form. XML applications that create XML 1.1 output from either XML 1.1 or XML 1.0 input SHOULD ensure that the output is fully normalized; it is not necessary for internal processing forms to be fully normalized.

The purpose of this section is to strongly encourage XML processors to ensure that the creators of XML documents have properly normalized them, so that XML applications can make tests such as identity comparisons of strings without having to worry about the different possible "spellings" of strings which Unicode allows.

When entities are in a non-Unicode encoding, if the processor transcodes them to Unicode, it SHOULD use a normalizing transcoder.

3 Logical Structures

[Definition: Each XML document contains one or more **elements**, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, sometimes called its "generic identifier" (GI), and MAY have a set of attribute specifications.] Each attribute specification has a name and a value.

Element

```
[39]  element  ::=  EmptyElemTag
                  | STag content ETag [WFC: Element Type Match]
                                      [VC: Element Valid]
```

This specification does not constrain the semantics, use, or (beyond syntax) names of the element types and attributes, except that names beginning with a match to `(('X'|'x')('M'|'m')('L'|'l'))` are reserved for standardization in this or future versions of this specification.

> **Well-formedness constraint: Element Type Match**
>
> The Name in an element's end-tag MUST match the element type in the start-tag.

Validity constraint: Element Valid

An element is valid if there is a declaration matching elementdecl where the Name matches the element type, and one of the following holds:

1. The declaration matches **EMPTY** and the element has no content (not even entity references, comments, PIs or white space).
2. The declaration matches children and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space, comments and PIs (i.e. markup matching production [27] Misc) between the start-tag and the first child element, between child elements, or between the last child element and the end-tag. Note that a CDATA section containing only white space or a reference to an entity whose replacement text is character references expanding to white space do not match the nonterminal S, and hence cannot appear in these positions; however, a reference to an internal entity with a literal value consisting of character references expanding to white space does match S, since its replacement text is the white space resulting from expansion of the character references.
3. The declaration matches Mixed and the content (after replacing any entity references with their replacement text) consists of character data, comments, PIs and child elements whose types match names in the content model.
4. The declaration matches **ANY**, and the content (after replacing any entity references with their replacement text) consists of character data and child elements whose types have been declared.

3.1 Start-Tags, End-Tags, and Empty-Element Tags

[Definition: The beginning of every non-empty XML element is marked by a **start-tag**.]

Start-tag

```
[40]  STag       ::=  '<' Name (S Attribute)* S? '>'  [WFC: Unique Att Spec]
[41]  Attribute  ::=  Name Eq AttValue                [VC: Attribute Value Type]
                                                      [WFC: No External Entity References]
                                                      [WFC: No < in Attribute Values]
```

The Name in the start- and end-tags gives the element's **type**. [Definition: The Name-AttValue pairs are referred to as the **attribute specifications** of the element], [Definition: with the Name in each pair referred to as the **attribute name**] and [Definition: the content of the AttValue (the text between the ' or " delimiters) as the **attribute value**.] Note that the order of attribute specifications in a start-tag or empty-element tag is not significant.

Well-formedness constraint: Unique Att Spec

An attribute name MUST NOT appear more than once in the same start-tag or empty-element tag.

Validity constraint: Attribute Value Type

The attribute MUST have been declared; the value MUST be of the type declared for it. (For attribute types, see **3.3 Attribute-List Declarations**.)

Well-formedness constraint: No External Entity References

Attribute values MUST NOT contain direct or indirect entity references to external entities.

Well-formedness constraint: No < in Attribute Values

The replacement text of any entity referred to directly or indirectly in an attribute value MUST NOT contain a <.

An example of a start-tag:

```
<termdef id="dt-dog" term="dog">
```

[Definition: The end of every element that begins with a start-tag MUST be marked by an **end-tag** containing a name that echoes the element's type as given in the start-tag:]

End-tag

```
[42]  ETag  ::=  '</' Name S? '>'
```

An example of an end-tag:

```
</termdef>
```

[Definition: The text between the start-tag and end-tag is called the element's **content**:]

Content of Elements

```
[43]  content  ::=  CharData? ((element | Reference | CDSect | PI | Comment) CharData?)*
```

[Definition: An element with no content is said to be **empty**.] The representation of an empty element is either a start-tag immediately followed by an end-tag, or an empty-element tag. [Definition: An **empty-element tag** takes a special form:]

Tags for Empty Elements

```
[44]  EmptyElemTag  ::=  '<' Name (S Attribute)* S? '/>' [WFC: Unique Att Spec]
```

Empty-element tags MAY be used for any element which has no content, whether or not it is declared using the keyword **EMPTY**. For interoperability, the empty-element tag SHOULD be used, and SHOULD only be used, for elements which are declared EMPTY.

Examples of empty elements:

```
<IMG align="left"
src="http://www.w3.org/Icons/WWW/w3c_home" />
<br></br>
<br/>
```

3.2 Element Type Declarations

The element structure of an XML document MAY, for validation purposes, be constrained using element type and attribute-list declarations. An element type declaration constrains the element's content.

Element type declarations often constrain which element types can appear as children of the element. At user option, an XML processor MAY issue a warning when a declaration mentions an element type for which no declaration is provided, but this is not an error.

[Definition: An **element type declaration** takes the form:]

Element Type Declaration

```
[45]  elementdecl  ::=  '<!ELEMENT' S Name S contentspec S? '>' [VC: Unique Element Type Declaration]
[46]  contentspec  ::=  'EMPTY' | 'ANY' | Mixed | children
```

where the Name gives the element type being declared.

> **Validity constraint: Unique Element Type Declaration**
>
> An element type MUST NOT be declared more than once.

Examples of element type declarations:

```
<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA|emph)* >
<!ELEMENT %name.para; %content.para; >
<!ELEMENT container ANY>
```

3.2.1 Element Content

[Definition: An element type has **element content** when elements of that type MUST contain only child elements (no character data), optionally separated by white space (characters matching the nonterminal S).] [Definition: In this case, the constraint includes a **content model**, a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear.] The grammar is built on content particles (cps), which consist of names, choice lists of content particles, or sequence lists of content particles:

Element-content Models

```
[47]  children  ::=  (choice | seq) ('?' | '*' | '+')?
[48]  cp        ::=  (Name | choice | seq) ('?' | '*' | '+')?
[49]  choice    ::=  '(' S? cp ( S? '|' S? cp )+ S? ')'      [VC: Proper Group/PE Nesting]
[50]  seq       ::=  '(' S? cp ( S? ',' S? cp )* S? ')'      [VC: Proper Group/PE Nesting]
```

where each Name is the type of an element which MAY appear as a child. Any content particle in a choice list MAY appear in the element content at the location where the choice list appears in the grammar; content particles occurring in a sequence list MUST each appear in the element content in the order given in the list. The optional character following a name or list governs whether the element or the content particles in the list may occur one or more (+), zero or more (*), or zero or one times (?). The absence of such an operator means that the element or content particle MUST appear exactly once. This syntax and meaning are identical to those used in the productions in this specification.

The content of an element matches a content model if and only if it is possible to trace out a path through the content model, obeying the sequence, choice, and repetition operators and matching each element in the content against an element type in the content model. For compatibility, it is an error if the content model allows an element to match more than one occurrence of an element type in the content model. For more information, see **D Deterministic Content Models**.

Validity constraint: Proper Group/PE Nesting

Parameter-entity replacement text MUST be properly nested with parenthesized groups. That is to say, if either of the opening or closing parentheses in a choice, seq, or Mixed construct is contained in the replacement text for a parameter entity, both MUST be contained in the same replacement text.

For interoperability, if a parameter-entity reference appears in a choice, seq, or Mixed construct, its replacement text SHOULD contain at least one non-blank character, and neither the first nor last non-blank character of the replacement text SHOULD be a connector (`|` or `,`).

Examples of element-content models:

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
<!ELEMENT dictionary-body (%div.mix; | %dict.mix;)*>
```

3.2.2 Mixed Content

[Definition: An element type has **mixed content** when elements of that type MAY contain character data, optionally interspersed with child elements.] In this case, the types of the child elements MAY be constrained, but not their order or their number of occurrences:

Mixed-content Declaration

```
[51]  Mixed  ::=  '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*'
                  | '(' S? '#PCDATA' S? ')'                    [VC: Proper Group/PE Nesting]
                                                               [VC: No Duplicate Types]
```

where the Names give the types of elements that may appear as children. The keyword **#PCDATA** derives historically from the term "parsed character data."

Validity constraint: No Duplicate Types

The same name MUST NOT appear more than once in a single mixed-content declaration.

Examples of mixed content declarations:

```
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >
<!ELEMENT b (#PCDATA)>
```

3.3 Attribute-List Declarations

Attributes are used to associate name-value pairs with elements. Attribute specifications MUST NOT appear outside of start-tags and empty-element tags; thus, the productions used to recognize them appear in **3.1 Start-Tags, End-Tags, and Empty-Element Tags**. Attribute-list declarations MAY be used:

- To define the set of attributes pertaining to a given element type.
- To establish type constraints for these attributes.
- To provide default values for attributes.

[Definition: **Attribute-list declarations** specify the name, data type, and default value (if any) of each attribute associated with a given element type:]

Attribute-list Declaration

```
[52]  AttlistDecl  ::=  '<!ATTLIST' S Name AttDef* S? '>'
[53]  AttDef       ::=  S Name S AttType S DefaultDecl
```

The Name in the AttlistDecl rule is the type of an element. At user option, an XML processor MAY issue a warning if attributes are declared for an element type not itself declared, but this is not an error. The Name in the AttDef rule is the name of the attribute.

When more than one AttlistDecl is provided for a given element type, the contents of all those provided are merged. When more than one definition is provided for the same attribute of a given element type, the first declaration is binding and later declarations are ignored. For interoperability, writers of DTDs MAY choose to provide at most one attribute-list declaration for a given element type, at most one attribute definition for a given attribute name in an attribute-list declaration, and at least one attribute definition in each attribute-list declaration. For interoperability, an XML processor MAY at user option issue a warning when more than one attribute-list declaration is provided for a given element type, or more than one attribute definition is provided for a given attribute, but this is not an error.

3.3.1 Attribute Types

XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The string type may take any literal string as a value; the tokenized types have varying lexical and semantic constraints. The validity constraints noted in the grammar are applied after the

attribute value has been normalized as described in **3.3.3 Attribute-Value Normalization**.

Attribute Types

```
[54]  AttType        ::=  StringType | TokenizedType | EnumeratedType
[55]  StringType     ::=  'CDATA'
[56]  TokenizedType  ::=  'ID'                                    [VC: ID]
                                                                  [VC: One ID per Element Type]
                                                                  [VC: ID Attribute Default]
                          | 'IDREF'                               [VC: IDREF]
                          | 'IDREFS'                              [VC: IDREF]
                          | 'ENTITY'                              [VC: Entity Name]
                          | 'ENTITIES'                            [VC: Entity Name]
                          | 'NMTOKEN'                             [VC: Name Token]
                          | 'NMTOKENS'                            [VC: Name Token]
```

Validity constraint: ID

Values of type **ID** MUST match the Name production. A name MUST NOT appear more than once in an XML document as a value of this type; i.e., ID values MUST uniquely identify the elements which bear them.

Validity constraint: One ID per Element Type

An element type MUST NOT have more than one ID attribute specified.

Validity constraint: ID Attribute Default

An ID attribute MUST have a declared default of **#IMPLIED** or **#REQUIRED**.

Validity constraint: IDREF

Values of type **IDREF** MUST match the Name production, and values of type **IDREFS** MUST match Names; each Name MUST match the value of an ID attribute on some element in the XML document; i.e. **IDREF** values MUST match the value of some ID attribute.

Validity constraint: Entity Name

Values of type **ENTITY** MUST match the Name production, values of type **ENTITIES** MUST match Names; each Name MUST match the name of an unparsed entity declared in the DTD.

Validity constraint: Name Token

Values of type **NMTOKEN** MUST match the Nmtoken production; values of type **NMTOKENS** MUST match Nmtokens.

[Definition: **Enumerated attributes** MUST take one of a list of values provided in the declaration]. There are two kinds of enumerated types:

Enumerated Attribute Types

```
[57]  EnumeratedType  ::=  NotationType | Enumeration
[58]  NotationType    ::=  'NOTATION' S '(' S? Name (S? '|' S? Name)* S?     [VC: Notation Attributes]
                           ')'
                                                                            [VC: One Notation Per Element Type]
                                                                            [VC: No Notation on Empty Element]
                                                                            [VC: No Duplicate Tokens]
[59]  Enumeration     ::=  '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'       [VC: Enumeration]
                                                                            [VC: No Duplicate Tokens]
```

A **NOTATION** attribute identifies a notation, declared in the DTD with associated system and/or public identifiers, to be used in interpreting the element to which the attribute is attached.

Validity constraint: Notation Attributes

Values of this type MUST match one of the *notation* names included in the declaration; all notation names in the declaration MUST be declared.

Validity constraint: One Notation Per Element Type

An element type MUST NOT have more than one **NOTATION** attribute specified.

Validity constraint: No Notation on Empty Element

For compatibility, an attribute of type **NOTATION** MUST NOT be declared on an element declared **EMPTY**.

Validity constraint: No Duplicate Tokens

The notation names in a single NotationType attribute declaration, as well as the NmTokens in a single Enumeration attribute declaration, MUST all be distinct.

Validity constraint: Enumeration

Values of this type MUST match one of the Nmtoken tokens in the declaration.

For interoperability, the same Nmtoken SHOULD NOT occur more than once in the enumerated attribute types of a single element type.

3.3.2 Attribute Defaults

An attribute declaration provides information on whether the attribute's presence is REQUIRED, and if not, how an XML processor is to react if a declared attribute is absent in a document.

Attribute Defaults

```
[60]  DefaultDecl  ::=  '#REQUIRED' | '#IMPLIED'
                       | (('#FIXED' S)? AttValue) [VC: Required Attribute]
                                                  [VC: Attribute Default Value Syntactically Correct]
                                                  [WFC: No < in Attribute Values]
                                                  [VC: Fixed Attribute Default]
```

In an attribute declaration, **#REQUIRED** means that the attribute MUST always be provided, **#IMPLIED** that no default value is provided. [Definition: If the declaration is neither **#REQUIRED** nor **#IMPLIED**, then the AttValue value contains the declared **default** value; the **#FIXED** keyword states that the attribute MUST always have the default value. When an XML processor encounters an element without a specification for an attribute for which it has read a default value declaration, it MUST report the attribute with the declared default value to the application.]

Validity constraint: Required Attribute

If the default declaration is the keyword **#REQUIRED**, then the attribute MUST be specified for all elements of the type in the attribute-list declaration.

Validity constraint: Attribute Default Value Syntactically Correct

The declared default value MUST meet the syntactic constraints of the declared attribute type.

Note that only the syntactic constraints of the type are required here; other constraints (e.g. that the value be the name of a declared unparsed entity, for an attribute of type ENTITY) may come into play if the declared default value is actually used (an element without a specification for this attribute occurs).

Validity constraint: Fixed Attribute Default

If an attribute has a default value declared with the **#FIXED** keyword, instances of that attribute MUST match the default value.

Examples of attribute-list declarations:

```
<!ATTLIST termdef
id      ID      #REQUIRED
name    CDATA   #IMPLIED>
<!ATTLIST list
type    (bullets|ordered|glossary)  "ordered">
<!ATTLIST form
method  CDATA   #FIXED "POST">
```

3.3.3 Attribute-Value Normalization

Before the value of an attribute is passed to the application or checked for validity, the XML processor MUST normalize the attribute value by applying the algorithm below, or by using some other method such that the value passed to the application is the same as that produced by the algorithm.

1. All line breaks MUST have been normalized on input to #xA as described in **2.11 End-of-Line Handling**, so the rest of this algorithm operates on text normalized in this way.
2. Begin with a normalized value consisting of the empty string.
3. For each character, entity reference, or character reference in the unnormalized attribute value, beginning with the first and continuing to the last, do the following:
   - For a character reference, append the referenced character to the normalized value.
   - For an entity reference, recursively apply step 3 of this algorithm to the replacement text of the entity.
   - For a white space character (#x20, #xD, #xA, #x9), append a space character (#x20) to the normalized value.
   - For another character, append the character to the normalized value.

If the attribute type is not CDATA, then the XML processor MUST further process the normalized attribute value by discarding any leading and trailing space (#x20) characters, and by replacing sequences of space (#x20) characters by a single space (#x20) character.

Note that if the unnormalized attribute value contains a character reference to a white space character other than space (#x20), the normalized value contains the referenced character itself (#xD, #xA or #x9). This contrasts with the case where the unnormalized value contains a white space character (not a reference), which is replaced with a space character (#x20) in the normalized value and also contrasts with the case where the unnormalized value contains an entity reference whose replacement text contains a white space character; being recursively processed, the white space character is replaced with a space character (#x20) in the normalized value.

All attributes for which no declaration has been read SHOULD be treated by a non-validating processor as if declared **CDATA**.

It is an error if an attribute value contains a reference to an entity for which no declaration has been read.

Following are examples of attribute normalization. Given the following declarations:

```
<!ENTITY d "&#xD;">
<!ENTITY a "&#xA;">
<!ENTITY da "&#xD;&#xA;">
```

the attribute specifications in the left column below would be normalized to the character sequences of the middle column if the attribute a is declared **NMTOKENS** and to those of the right columns if a is declared **CDATA**.

Attribute specification	a is NMTOKENS	a is CDATA
a=" xyz"	x y z	#x20 #x20 x y z
a="&d;&d;A&a; &a;B&da;"	A #x20 B	#x20 #x20 A #x20 #x20 #x20 B #x20 #x20
a= "

A

B
" | #xD #xD A #xA #xA B #xD #xA | #xD #xD A #xA #xA B #xD #xA |

Note that the last example is invalid (but well-formed) if a is declared to be of type **NMTOKENS**.

3.4 Conditional Sections

[Definition: **Conditional sections** are portions of the document type declaration external subset or of external parameter entities which are included in, or excluded from, the logical structure of the DTD based on the keyword which governs them.]

Conditional Section

```
[61]  conditionalSect     ::=  includeSect | ignoreSect
[62]  includeSect         ::=  '<!['  S?  'INCLUDE'  S?  '['  extSubsetDecl  ']]>'      [VC: Proper Conditional Section/PE Nesting]
[63]  ignoreSect          ::=  '<!['  S?  'IGNORE'  S?  '['  ignoreSectContents*  ']]>' [VC: Proper Conditional Section/PE Nesting]
[64]  ignoreSectContents  ::=  Ignore  ('<!['  ignoreSectContents  ']]>'  Ignore)*
[65]  Ignore              ::=  Char* - (Char* ('<!['  |  ']]>')  Char*)
```

Validity constraint: Proper Conditional Section/PE Nesting

If any of the "<![", "[", or "]]>" of a conditional section is contained in the replacement text for a parameter-entity reference, all of them MUST be contained in the same replacement text.

Like the internal and external DTD subsets, a conditional section may contain one or more complete declarations, comments, processing instructions, or nested conditional sections, intermingled with white space.

If the keyword of the conditional section is **INCLUDE**, then the contents of the conditional section MUST be considered part of the DTD. If the keyword of the conditional section is **IGNORE**, then the contents of the conditional section MUST be considered as not logically part of the DTD. If a conditional section with a keyword of **INCLUDE** occurs within a larger conditional section with a keyword of **IGNORE**, both the outer and the inner conditional sections MUST be ignored. The contents of an ignored conditional section MUST be parsed by ignoring all characters after the "[" following the keyword, except conditional section starts "<![" and ends "]]>", until the matching conditional section end is found. Parameter entity references MUST NOT be recognized in this process.

If the keyword of the conditional section is a parameter-entity reference, the parameter entity MUST be replaced by its content before the processor decides whether to include or ignore the conditional section.

An example:

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >
<![%draft;[
```

```
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

4 Physical Structures

[Definition: An XML document may consist of one or many storage units. These are called **entities**; they all have **content** and are all (except for the document entity and the external DTD subset) identified by entity **name**.] Each XML document has one entity called the document entity, which serves as the starting point for the XML processor and may contain the whole document.

Entities may be either parsed or unparsed. [Definition: The contents of a **parsed entity** are referred to as its replacement text; this text is considered an integral part of the document.]

[Definition: An **unparsed entity** is a resource whose contents may or may not be text, and if text, may be other than XML. Each unparsed entity has an associated notation, identified by name. Beyond a requirement that an XML processor make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.]

Parsed entities are invoked by name using entity references; unparsed entities by name, given in the value of **ENTITY** or **ENTITIES** attributes.

[Definition: **General entities** are entities for use within the document content. In this specification, general entities are sometimes referred to with the unqualified term *entity* when this leads to no ambiguity.] [Definition: **Parameter entities** are parsed entities for use within the DTD.] These two types of entities use different forms of reference and are recognized in different contexts. Furthermore, they occupy different namespaces; a parameter entity and a general entity with the same name are two distinct entities.

4.1 Character and Entity References

[Definition: A **character reference** refers to a specific character in the ISO/IEC 10646 character set, for example one not directly accessible from available input devices.]

Character Reference

```
[66]  CharRef  ::=  '&#' [0-9]+ ';'
                  | '&#x' [0-9a-fA-F]+ ';' [WFC: Legal Character]
```

Well-formedness constraint: Legal Character

Characters referred to using character references MUST match the production for Char.

If the character reference begins with "`&#x`", the digits and letters up to the terminating `;` provide a hexadecimal representation of the character's code point in ISO/IEC 10646. If it begins just with "`&#`", the digits up to the terminating `;` provide a decimal representation of the character's code point.

[Definition: An **entity reference** refers to the content of a named entity.] [Definition: References to parsed general entities use ampersand (`&`) and semicolon (`;`) as delimiters.] [Definition: **Parameter-entity references** use percent-sign (`%`) and semicolon (`;`) as delimiters.]

Entity Reference

```
[67]  Reference     ::=  EntityRef | CharRef
[68]  EntityRef     ::=  '&' Name ';'        [WFC: Entity Declared]
                                             [VC: Entity Declared]
                                             [WFC: Parsed Entity]
                                             [WFC: No Recursion]
[69]  PEReference   ::=  '%' Name ';'        [VC: Entity Declared]
                                             [WFC: No Recursion]
                                             [WFC: In DTD]
```

Well-formedness constraint: Entity Declared

In a document without any DTD, a document with only an internal DTD subset which contains no parameter entity references, or a document with "`standalone='yes'`", for an entity reference that does not occur within the external subset or a parameter entity, the Name given in the entity reference MUST match that in an *entity declaration* that does not occur within the external subset or a parameter entity, except that well-formed documents need not declare any of the following entities: `amp`, `lt`, `gt`, `apos`, `quot`. The declaration of a general entity MUST precede any reference to it which appears in a default value in an attribute-list declaration.

Note that non-validating processors are *not obligated to* to read and process entity declarations occurring in parameter entities or in the external subset; for such documents, the rule that an entity must be declared is a well-formedness constraint only if *standalone='yes'*.

Validity constraint: Entity Declared

In a document with an external subset or external parameter entities with "`standalone='no'`", the Name given in the entity reference MUST match that in an *entity declaration*. For interoperability, valid documents SHOULD declare the entities `amp`, `lt`, `gt`, `apos`, `quot`, in the form specified in **4.6 Predefined Entities**. The declaration of a parameter entity MUST precede any reference to it. Similarly, the declaration of a general entity MUST precede any attribute-list declaration containing a default value with a direct or indirect reference to that general entity.

Well-formedness constraint: Parsed Entity

An entity reference MUST NOT contain the name of an unparsed entity. Unparsed entities may be referred to only in attribute values declared to be of type **ENTITY** or **ENTITIES**.

Well-formedness constraint: No Recursion

A parsed entity MUST NOT contain a recursive reference to itself, either directly or indirectly.

Well-formedness constraint: In DTD

Parameter-entity references MUST NOT appear outside the DTD.

Examples of character and entity references:

```
Type <key>less-than</key> (&#x3C;) to save options.
This document was prepared on &docdate; and
is classified &security-level;.
```

Example of a parameter-entity reference:

```
<!-- declare the parameter entity "ISOLat2"... -->
<!ENTITY % ISOLat2
SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ... now reference it. -->
%ISOLat2;
```

4.2 Entity Declarations

[Definition: Entities are declared thus:]

Entity Declaration

```
[70]  EntityDecl  ::=  GEDecl | PEDecl
[71]  GEDecl      ::=  '<!ENTITY' S Name S EntityDef S? '>'
[72]  PEDecl      ::=  '<!ENTITY' S '%' S Name S PEDef S? '>'
[73]  EntityDef   ::=  EntityValue| (ExternalID NDataDecl?)
[74]  PEDef       ::=  EntityValue | ExternalID
```

The Name identifies the entity in an entity reference or, in the case of an unparsed entity, in the value of an **ENTITY** or **ENTITIES** attribute. If the same entity is declared more than once, the first declaration encountered is binding; at user option, an XML processor MAY issue a warning if entities are declared multiple times.

4.2.1 Internal Entities

[Definition: If the entity definition is an EntityValue, the defined entity is called an **internal entity**. There is no separate physical storage object, and the content of the entity is given in the declaration.] Note that some processing of entity and character references in the literal entity value may be required to produce the correct replacement text: see **4.5 Construction of Entity Replacement Text**.

An internal entity is a parsed entity.

Example of an internal entity declaration:

```
<!ENTITY Pub-Status "This is a pre-release of the
specification.">
```

4.2.2 External Entities

[Definition: If the entity is not internal, it is an **external entity**, declared as follows:]

External Entity Declaration

```
[75]  ExternalID  ::=  'SYSTEM' S SystemLiteral
                     | 'PUBLIC' S PubidLiteral S SystemLiteral
```

[76] NDataDecl ::= S 'NDATA' S Name [VC: Notation Declared]

If the NDataDecl is present, this is a general unparsed entity; otherwise it is a parsed entity.

Validity constraint: Notation Declared

The Name MUST match the declared name of a notation.

[Definition: The SystemLiteral is called the entity's **system identifier**. It is meant to be converted to a URI reference (as defined in [IETF RFC 2396], updated by [IETF RFC 2732]), as part of the process of dereferencing it to obtain input for the XML processor to construct the entity's replacement text.] It is an error for a fragment identifier (beginning with a # character) to be part of a system identifier. Unless otherwise provided by information outside the scope of this specification (e.g. a special XML element type defined by a particular DTD, or a processing instruction defined by a particular application specification), relative URIs are relative to the location of the resource within which the entity declaration occurs. This is defined to be the external entity containing the '<' which starts the declaration, at the point when it is parsed as a declaration. A URI might thus be relative to the document entity, to the entity containing the external DTD subset, or to some other external parameter entity. Attempts to retrieve the resource identified by a URI MAY be redirected at the parser level (for example, in an entity resolver) or below (at the protocol level, for example, via an HTTP `Location:` header). In the absence of additional information outside the scope of this specification within the resource, the base URI of a resource is always the URI of the actual resource returned. In other words, it is the URI of the resource retrieved after all redirection has occurred.

System identifiers (and other XML strings meant to be used as URI references) MAY contain characters that, according to [IETF RFC 2396] and [IETF RFC 2732], must be escaped before a URI can be used to retrieve the referenced resource. The characters to be escaped are the control characters #x0 to #x1F and #x7F (most of which cannot appear in XML), space #x20, the delimiters '<' #x3C, '>' #x3E and "" #x22, the *unwise* characters '{' #x7B, '}' #x7D, '|' #x7C, '\' #x5C, '^' #x5E and '`' #x60, as well as all characters above #x7F. Since escaping is not always a fully reversible process, it MUST be performed only when absolutely necessary and as late as possible in a processing chain. In particular, neither the process of converting a relative URI to an absolute one nor the process of passing a URI reference to a process or software component responsible for dereferencing it SHOULD trigger escaping. When escaping does occur, it MUST be performed as follows:

1. Each character to be escaped is represented in UTF-8 [Unicode] as one or more bytes.
2. The resulting bytes are escaped with the URI escaping mechanism (that is, converted to %*HH*, where HH is the hexadecimal notation of the byte value).
3. The original character is replaced by the resulting character sequence.

[Definition: In addition to a system identifier, an external identifier MAY include a **public identifier**.] An XML processor attempting to retrieve the entity's content MAY use any combination of the public and system identifiers as well as additional information outside the scope of this specification to try to generate an alternative URI reference. If the processor is unable to do so, it MUST use the URI reference specified in the system literal. Before a match is attempted, all strings of white space in the public identifier MUST be normalized to single space characters (#x20), and leading and trailing white space MUST be removed.

Examples of external entity declarations:

```
<!ENTITY open-hatch
SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
"http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
SYSTEM "../grafix/OpenHatch.gif"
NDATA gif >
```

4.3 Parsed Entities

4.3.1 The Text Declaration

External parsed entities SHOULD each begin with a **text declaration**.

Text Declaration

[77] TextDecl ::= '<?xml' VersionInfo? EncodingDecl S? '?>'

The text declaration MUST be provided literally, not by reference to a parsed entity. The text declaration MUST NOT appear at any position other than the beginning of an external parsed entity. The text declaration in an external parsed entity is not considered part of its replacement text.

4.3.2 Well-Formed Parsed Entities

The document entity is well-formed if it matches the production labeled document. An external general parsed entity is well-formed if it matches the production labeled extParsedEnt. All external parameter entities are well-formed by definition.

Well-Formed External Parsed Entity

[78] extParsedEnt ::= TextDecl? content - Char* RestrictedChar Char*

An internal general parsed entity is well-formed if its replacement text matches the production labeled content. All internal parameter entities are well-formed by definition.

A consequence of well-formedness in general entities is that the logical and physical structures in an XML document are properly nested; no start-tag, end-tag, empty-element tag, element, comment, processing instruction, character reference, or entity reference can begin in one entity and end in another.

4.3.3 Character Encoding in Entities

Each external parsed entity in an XML document MAY use a different encoding for its characters. All XML processors MUST be able to read entities in both the UTF-8 and UTF-16 encodings. The terms "UTF-8" and "UTF-16" in this specification do not apply to character encodings with any other labels, even if the encodings or labels are very similar to UTF-8 or UTF-16.

Entities encoded in UTF-16 MUST and entities encoded in UTF-8 MAY begin with the Byte Order Mark described in ISO/IEC 10646 [ISO/IEC 10646] or Unicode [Unicode] (the ZERO WIDTH NO-BREAK SPACE character, #xFEFF). This is an encoding signature, not part of either the markup or the character data of the XML document. XML processors MUST be able to use this character to differentiate between UTF-8 and UTF-16 encoded documents.

Although an XML processor is required to read only entities in the UTF-8 and UTF-16 encodings, it is recognized that other encodings are used around the world, and it may be desired for XML processors to read entities that use them. In the absence of external character encoding information (such as MIME headers), parsed entities which are stored in an encoding other than UTF-8 or UTF-16 MUST begin with a text declaration (see **4.3.1 The Text Declaration**) containing an encoding declaration:

Encoding Declaration

```
[80]  EncodingDecl  ::=  S 'encoding' Eq ('"' EncName '"' | "'" EncName
                         "'" )
[81]  EncName       ::=  [A-Za-z] ([A-Za-z0-9._] | '-')*                  /* Encoding name contains only Latin characters
                                                                          */
```

In the document entity, the encoding declaration is part of the XML declaration. The EncName is the name of the encoding used.

In an encoding declaration, the values "`UTF-8`", "`UTF-16`", "`ISO-10646-UCS-2`", and "`ISO-10646-UCS-4`" SHOULD be used for the various encodings and transformations of Unicode / ISO/IEC 10646, the values "`ISO-8859-1`", "`ISO-8859-2`", ... "`ISO-8859-`*n*" (where *n* is the part number) SHOULD be used for the parts of ISO 8859, and the values "`ISO-2022-JP`", "`Shift_JIS`", and "`EUC-JP`" SHOULD be used for the various encoded forms of JIS X-0208-1997. It is RECOMMENDED that character encodings registered (as *charset*s) with the Internet Assigned Numbers Authority [IANA-CHARSETS], other than those just listed, be referred to using their registered names; other encodings SHOULD use names starting with an "x-" prefix. XML processors SHOULD match character encoding names in a case-insensitive way and SHOULD either interpret an IANA-registered name as the encoding registered at IANA for that name or treat it as unknown (processors are, of course, not required to support all IANA-registered encodings).

In the absence of information provided by an external transport protocol (e.g. HTTP or MIME), it is a fatal error for an entity including an encoding declaration to be presented to the XML processor in an encoding other than that named in the declaration, or for an entity which begins with neither a Byte Order Mark nor an encoding declaration to use an encoding other than UTF-8. Note that since ASCII is a subset of UTF-8, ordinary ASCII entities do not strictly need an encoding declaration.

It is a fatal error for a TextDecl to occur other than at the beginning of an external entity.

It is a fatal error when an XML processor encounters an entity with an encoding that it is unable to process. It is a fatal error if an XML entity is determined (via default, encoding declaration, or higher-level protocol) to be in a certain encoding but contains byte sequences that are not legal in that encoding. Specifically, it is a fatal error if an entity encoded in UTF-8 contains any irregular code unit sequences, as defined in Unicode [Unicode]. Unless an encoding is determined by a higher-level protocol, it is also a fatal error if an XML entity contains no encoding declaration and its content is not legal UTF-8 or UTF-16.

Examples of text declarations containing encoding declarations:

```
<?xml encoding='UTF-8'?>
<?xml encoding='EUC-JP'?>
```

4.3.4 Version Information in Entities

Each entity, including the document entity, can be separately declared as XML 1.0 or XML 1.1. The version declaration appearing in the document entity determines the version of the document as a whole. An XML 1.1 document may invoke XML 1.0 external entities, so that otherwise duplicated versions of external entities, particularly DTD external subsets, need not be maintained. However, in such a case the rules of XML 1.1 are applied to the entire document.

If an entity (including the document entity) is not labeled with a version number, it is treated as if labeled as version 1.0.

4.4 XML Processor Treatment of Entities and References

The table below summarizes the contexts in which character references, entity references, and invocations of unparsed entities might appear and the REQUIRED behavior of an XML processor in each case. The labels in the leftmost column describe the recognition context:

Reference in Content

as a reference anywhere after the start-tag and before the end-tag of an element; corresponds to the nonterminal content.

Reference in Attribute Value

as a reference within either the value of an attribute in a start-tag, or a default value in an attribute declaration; corresponds to the nonterminal AttValue.

Occurs as Attribute Value

as a Name, not a reference, appearing either as the value of an attribute which has been declared as type **ENTITY**, or as one of the space-separated tokens in the value of an attribute which has been declared as type **ENTITIES**.

Reference in Entity Value

as a reference within a parameter or internal entity's literal entity value in the entity's declaration; corresponds to the nonterminal EntityValue.

Reference in DTD

as a reference within either the internal or external subsets of the DTD, but outside of an EntityValue, AttValue, PI, Comment, SystemLiteral, PubidLiteral, or the contents of an ignored conditional section (see **3.4 Conditional Sections**).

.

	Entity Type				Character
	Parameter	Internal General	External Parsed General	Unparsed	
Reference in Content	*Not recognized*	*Included*	*Included if validating*	*Forbidden*	*Included*
Reference in Attribute Value	*Not recognized*	*Included in literal*	*Forbidden*	*Forbidden*	*Included*
Occurs as Attribute Value	*Not recognized*	*Forbidden*	*Forbidden*	*Notify*	*Not recognized*
Reference in EntityValue	*Included in literal*	*Bypassed*	*Bypassed*	*Error*	*Included*
Reference in DTD	*Included as PE*	*Forbidden*	*Forbidden*	*Forbidden*	*Forbidden*

4.4.1 Not Recognized

Outside the DTD, the % character has no special significance; thus, what would be parameter entity references in the DTD are not recognized as markup in content. Similarly, the names of unparsed entities are not recognized except when they appear in the value of an appropriately declared attribute.

4.4.2 Included

[Definition: An entity is **included** when its replacement text is retrieved and processed, in place of the reference itself, as though it were part of the document at the location the reference was recognized.] The replacement text MAY contain both character data and (except for parameter entities) markup, which MUST be recognized in the usual way. (The string "`AT&T;`" expands to "`AT&T;`" and the remaining ampersand is not recognized as an entity-reference delimiter.) A character reference is **included** when the indicated character is processed in place of the reference itself.

4.4.3 Included If Validating

When an XML processor recognizes a reference to a parsed entity, in order to validate the document, the processor MUST include its replacement text. If the entity is external, and the processor is not attempting to validate the XML document, the processor MAY, but need not, include the entity's replacement text. If a non-validating processor does not include the replacement text, it MUST inform the application that it recognized, but did not read, the entity.

This rule is based on the recognition that the automatic inclusion provided by the SGML and XML entity mechanism, primarily designed to support modularity in authoring, is not necessarily appropriate for other applications, in particular document browsing. Browsers, for example, when encountering an external parsed entity reference, might choose to provide a visual indication of the entity's presence and retrieve it for display only on demand.

4.4.4 Forbidden

The following are forbidden, and constitute fatal errors:

- the appearance of a reference to an unparsed entity, except in the EntityValue in an entity declaration.
- the appearance of any character or general-entity reference in the DTD except within an EntityValue or AttValue.
- a reference to an external entity in an attribute value.

4.4.5 Included in Literal

When an entity reference appears in an attribute value, or a parameter entity reference appears in a literal entity value, its replacement text MUST be processed in place of the reference itself as though it were part of the document at the location the reference was recognized, except that a single or double quote character in the replacement text MUST always be treated as a normal data character and MUST NOT terminate the literal. For example, this is well-formed:

```
<!ENTITY % YN '"Yes"' >
<!ENTITY WhatHeSaid "He said %YN;" >
```

while this is not:

```
<!ENTITY EndAttr "27'" >
<element attribute='a-&EndAttr;>
```

4.4.6 Notify

When the name of an unparsed entity appears as a token in the value of an attribute of declared type **ENTITY** or **ENTITIES**, a validating processor MUST inform the application of the system and public (if any) identifiers for both the entity and its associated notation.

4.4.7 Bypassed

When a general entity reference appears in the EntityValue in an entity declaration, it MUST be bypassed and left as is.

4.4.8 Included as PE

Just as with external parsed entities, parameter entities need only be *included if validating*. When a parameter-entity reference is recognized in the DTD and included, its replacement text MUST be enlarged by the attachment of one leading and one following space (#x20) character; the intent is to constrain the replacement text of parameter entities to contain an integral number of grammatical tokens in the DTD. This behavior MUST NOT apply to parameter entity references within entity values; these are described in **4.4.5 Included in Literal**.

4.4.9 Error

It is an error for a reference to an unparsed entity to appear in the EntityValue in an entity declaration.

4.5 Construction of Entity Replacement Text

In discussing the treatment of entities, it is useful to distinguish two forms of the entity's value. [Definition: For an internal entity, the **literal entity value** is the quoted string actually present in the entity declaration, corresponding to the non-terminal EntityValue.] [Definition: For an external entity, the **literal entity value** is the exact text contained in the entity.] [Definition: For an internal entity, the **replacement text** is the content of the entity, after replacement of character references and parameter-entity references.] [Definition: For an external entity, the **replacement text** is the content of the entity, after stripping the text declaration (leaving any surrounding white space) if there is one but without any replacement of character references or parameter-entity references.]

The literal entity value as given in an internal entity declaration (EntityValue) MAY contain character, parameter-entity, and general-entity references. Such references MUST be contained entirely within the literal entity value. The actual replacement text that is included (or included in literal) as described above MUST contain the *replacement text* of any parameter entities referred to, and MUST contain the character referred to, in place of any character references in the literal entity value; however, general-entity references MUST be left as-is, unexpanded. For example, given the following declarations:

```
<!ENTITY % pub    "&#xc9;ditions Gallimard" >
<!ENTITY   rights "All rights reserved" >
<!ENTITY   book   "La Peste: Albert Camus,
&#xA9; 1947 %pub;. &rights;" >
```

then the replacement text for the entity "`book`" is:

```
La Peste: Albert Camus,
© 1947 Éditions Gallimard. &rights;
```

The general-entity reference "`&rights;`" would be expanded should the reference "`&book;`" appear in the document's content or an attribute value.

These simple rules may have complex interactions; for a detailed discussion of a difficult example, see **C Expansion of Entity and Character References**.

4.6 Predefined Entities

[Definition: Entity and character references MAY both be used to **escape** the left angle bracket, ampersand, and other delimiters. A set of general entities (`amp`, `lt`, `gt`, `apos`, `quot`) is specified for this purpose. Numeric character references MAY also be used; they are expanded immediately when recognized and MUST be treated as character data, so the numeric character references "`&#60;`" and "`&#38;`" MAY be used to escape < and & when they occur in character data.]

All XML processors MUST recognize these entities whether they are declared or not. For interoperability, valid XML documents SHOULD declare these entities, like any others, before using them. If the entities `lt` or `amp` are declared, they MUST be declared as internal entities whose replacement text is a character reference to the respective character (less-than sign or ampersand) being escaped; the double escaping is REQUIRED for these entities so that references to them produce a well-formed result. If the entities `gt`, `apos`, or `quot` are declared, they MUST be declared as internal entities whose replacement text is the single character being escaped (or a character reference to that character; the double escaping here is OPTIONAL but harmless). For example:

```
<!ENTITY lt     "&#38;#60;">
<!ENTITY gt     "&#62;">
<!ENTITY amp    "&#38;#38;">
<!ENTITY apos   "&#39;">
<!ENTITY quot   "&#34;">
```

4.7 Notation Declarations

[Definition: **Notations** identify by name the format of unparsed entities, the format of elements which bear a notation attribute, or the application to which a processing instruction is addressed.]

[Definition: **Notation declarations** provide a name for the notation, for use in entity and attribute-list declarations and in attribute specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.]

Notation Declarations

```
[82]  NotationDecl  ::=  '<!NOTATION' S Name S (ExternalID | PublicID) S? '>' [VC: Unique Notation Name]
[83]  PublicID      ::=  'PUBLIC' S PubidLiteral
```

Validity constraint: Unique Notation Name

A given Name MUST NOT be declared in more than one notation declaration.

XML processors MUST provide applications with the name and external identifier(s) of any notation declared and referred to in an attribute value, attribute definition, or entity declaration. They MAY additionally resolve the external identifier into the system identifier, file name, or other information needed to allow the application to call a processor for data in the notation described. (It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.)

4.8 Document Entity

[Definition: The **document entity** serves as the root of the entity tree and a starting-point for an XML processor.] This specification does not specify how the document entity is to be located by an XML processor; unlike other entities, the document entity has no name and might well appear on a processor input stream without any identification at all.

5 Conformance

5.1 Validating and Non-Validating Processors

Conforming XML processors fall into two classes: validating and non-validating.

Validating and non-validating processors alike MUST report violations of this specification's well-formedness constraints in the content of the document entity and any other parsed entities that they read.

[Definition: **Validating processors** MUST, at user option, report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification.] To accomplish this, validating XML processors MUST read and process the entire DTD and all external parsed entities referenced in the document.

Non-validating processors are REQUIRED to check only the document entity, including the entire internal DTD subset, for well-formedness. [Definition: While they are not required to check the document for validity, they are REQUIRED to **process** all the declarations they read in the internal DTD subset and in any parameter entity that they read, up to the first reference to a parameter entity that they do *not* read; that is to say, they MUST use the information in those declarations to *normalize* attribute values, *include* the replacement text of internal entities, and supply *default attribute values*.] Except when `standalone="yes"`, they MUST NOT process entity declarations or attribute-list declarations encountered after a reference to a parameter entity that is not read, since the entity may have contained overriding declarations; when `standalone="yes"`, processors MUST process these declarations.

Note that when processing invalid documents with a non-validating processor the application may not be presented with consistent information. For example, several requirements for uniqueness within the document may not be met, including more than one element with the same id, duplicate declarations of elements or notations with the same name, etc. In these cases the behavior of the parser with respect to reporting such information to the application is undefined.

XML 1.1 processors MUST be able to process both XML 1.0 and XML 1.1 documents. Programs which generate XML SHOULD generate XML 1.0, unless one of the specific features of XML 1.1 is required.

5.2 Using XML Processors

The behavior of a validating XML processor is highly predictable; it must read every piece of a document and report all well-formedness and validity violations. Less is required of a non-validating processor; it need not read any part of the document other than the document entity. This has two effects that may be important to users of XML processors:

- Certain well-formedness errors, specifically those that require reading external entities, may fail to be detected by a non-validating processor. Examples include the constraints entitled *Entity Declared*, *Parsed Entity*, and *No Recursion*, as well as some of the cases

described as *forbidden* in **4.4 XML Processor Treatment of Entities and References**.

- The information passed from the processor to the application may vary, depending on whether the processor reads parameter and external entities. For example, a non-validating processor may fail to *normalize* attribute values, *include* the replacement text of internal entities, or supply *default attribute values*, where doing so depends on having read declarations in external or parameter entities.

For maximum reliability in interoperating between different XML processors, applications which use non-validating processors SHOULD NOT rely on any behaviors not required of such processors. Applications which require DTD facilities not related to validation (such as the declaration of default attributes and internal entities that are or may be specified in external entities SHOULD use validating XML processors.

6 Notation

The formal grammar of XML is given in this specification using a simple Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one symbol, in the form

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are the start symbol of a regular language, otherwise with an initial lowercase letter. Literal strings are quoted.

Within the expression on the right-hand side of a rule, the following expressions are used to match strings of one or more characters:

#xN

where N is a hexadecimal integer, the expression matches the character whose number (code point) in ISO/IEC 10646 is N. The number of leading zeros in the #xN form is insignificant.

[a-zA-Z], [#xN-#xN]

matches any Char with a value in the range(s) indicated (inclusive).

[abc], [#xN#xN#xN]

matches any Char with a value among the characters enumerated. Enumerations and ranges can be mixed in one set of brackets.

[^a-z], [^#xN-#xN]

matches any Char with a value *outside* the range indicated.

[^abc], [^#xN#xN#xN]

matches any Char with a value not among the characters given. Enumerations and ranges of forbidden values can be mixed in one set of brackets.

"string"

matches a literal string matching that given inside the double quotes.

'string'

matches a literal string matching that given inside the single quotes.

These symbols may be combined to match more complex patterns as follows, where A and B represent simple expressions:

(expression)

expression is treated as a unit and may be combined as described in this list.

A?

matches A or nothing; optional A.

A B

matches A followed by B. This operator has higher precedence than alternation; thus A B | C D is identical to (A B) | (C D).

A | B

matches A or B.

A - B

matches any string that matches A but does not match B.

A+

matches one or more occurrences of A. Concatenation has higher precedence than alternation; thus A+ | B+ is identical to (A+) | (B+).

A*

matches zero or more occurrences of A. Concatenation has higher precedence than alternation; thus A* | B* is identical to (A*) | (B*).

Other notations used in the productions are:

/* ... */

comment.

[wfc: ...]

well-formedness constraint; this identifies by name a constraint on well-formed documents associated with a production.

[vc: ...]

validity constraint; this identifies by name a constraint on valid documents associated with a production.

A References

A.1 Normative References

IANA-CHARSETS
(Internet Assigned Numbers Authority) *Official Names for Character Sets*, ed. Keld Simonsen et al. (See http://www.iana.org/assignments/character-sets.)

IETF RFC 2119
IETF (Internet Engineering Task Force). *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*. Scott Bradner, 1997. (See http://www.ietf.org/rfc/rfc2119.txt.)

IETF RFC 2396
IETF (Internet Engineering Task Force). *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*. T. Berners-Lee, R. Fielding, L. Masinter. 1998. (See http://www.ietf.org/rfc/rfc2396.txt.)

IETF RFC 2732
IETF (Internet Engineering Task Force). *RFC 2732: Format for Literal IPv6 Addresses in URL's*. R. Hinden, B. Carpenter, L. Masinter. 1999. (See http://www.ietf.org/rfc/rfc2732.txt.)

IETF RFC 3066
IETF (Internet Engineering Task Force). *RFC 3066: Tags for the Identification of Languages*, ed. H. Alvestrand. 2001. (See http://www.ietf.org/rfc/rfc3066.txt.)

ISO/IEC 10646
ISO (International Organization for Standardization). *ISO/IEC 10646-1:2000. Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane* and *ISO/IEC 10646-2:2001. Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 2: Supplementary Planes*, as, from time to time, amended, replaced by a new edition or expanded by the addition of new parts. [Geneva]: International Organization for Standardization. (See http://www.iso.ch for the latest version.)

Unicode
The Unicode Consortium. *The Unicode Standard, Version 4.0.* Reading, Mass.: Addison-Wesley, 2003, as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions for the latest version and additional information on versions of the standard and of the Unicode Character Database).

XML-1.0
W3C. *Extensible Markup Language (XML) 1.0 (Third Edition)*. Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler, François Yergeau (editors) (See http://www.w3.org/TR/REC-xml.)

A.2 Other References

Aho/Ullman
Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Reading: Addison-Wesley, 1986, rpt. corr. 1988.

Brüggemann-Klein
Brüggemann-Klein, Anne. *Formal Models in Document Processing*. Habilitationsschrift. Faculty of Mathematics at the University of Freiburg, 1993. (See ftp://ftp.informatik.uni-freiburg.de/documents/papers/brueggem/habil.ps.)

Brüggemann-Klein and Wood
Brüggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages*. Universität Freiburg, Institut für Informatik, Bericht 38, Oktober 1991. Extended abstract in A. Finkel, M. Jantzen, Hrsg., STACS 1992, S. 173-184. Springer-Verlag, Berlin 1992. Lecture Notes in Computer Science 577. Full version titled *One-Unambiguous Regular Languages* in Information and Computation 140 (2): 229-253, February 1998.

Charmod
W3C Working Draft. *Character Model for the World Wide Web 1.0*. Martin J. Dürst, François Yergeau, Richard Ishida, Misha Wolf, Tex Texin. (See http://www.w3.org/TR/2003/WD-charmod-20030822/.)

Clark
James Clark. *Comparison of SGML and XML*. (See http://www.w3.org/TR/NOTE-sgml-xml-971215.)

IANA-LANGCODES
(Internet Assigned Numbers Authority) *Registry of Language Tags*, ed. Keld Simonsen et al. (See http://www.iana.org/assignments/language-tags.)

IETF RFC 2141
IETF (Internet Engineering Task Force). *RFC 2141: URN Syntax*, ed. R. Moats. 1997. (See http://www.ietf.org/rfc/rfc2141.txt.)

IETF RFC 3023
IETF (Internet Engineering Task Force). *RFC 3023: XML Media Types*. eds. M. Murata, S. St.Laurent, D. Kohn. 2001. (See http://www.ietf.org/rfc/rfc3023.txt.)

IETF RFC 2781
IETF (Internet Engineering Task Force). *RFC 2781: UTF-16, an encoding of ISO 10646*, ed. P. Hoffman, F. Yergeau. 2000. (See http://www.ietf.org/rfc/rfc2781.txt.)

ISO 639
(International Organization for Standardization). *ISO 639:1988 (E). Code for the representation of names of languages.* [Geneva]: International Organization for Standardization, 1988.

ISO 3166
(International Organization for Standardization). *ISO 3166-1:1997 (E). Codes for the representation of names of countries and their subdivisions — Part 1: Country codes* [Geneva]: International Organization for Standardization, 1997.

ISO 8879

ISO (International Organization for Standardization). *ISO 8879:1986(E). Information processing — Text and Office Systems — Standard Generalized Markup Language (SGML).* First edition — 1986-10-15. [Geneva]: International Organization for Standardization, 1986.

ISO/IEC 10744

ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology — Hypermedia/Time-based Structuring Language (HyTime).* [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe.* [Geneva]: International Organization for Standardization, 1996.

WEBSGML

ISO (International Organization for Standardization). *ISO 8879:1986 TC2. Information technology — Document Description and Processing Languages*. [Geneva]: International Organization for Standardization, 1998. (See http://www.sgmlsource.com/8879/n0029.htm.)

XML Names

Tim Bray, Dave Hollander, and Andrew Layman, editors. *Namespaces in XML*. Textuality, Hewlett-Packard, and Microsoft. World Wide Web Consortium, 1999. (See http://www.w3.org/TR/REC-xml-names/.)

B Definitions for Character Normalization

This appendix contains the necessary definitions for character normalization. For additional background information and examples, see [Charmod].

[Definition: Text is said to be in a **Unicode encoding form** if it is encoded in UTF-8, UTF-16 or UTF-32.]

[Definition: **Legacy encoding** is taken to mean any character encoding not based on Unicode.]

[Definition: A **normalizing transcoder** is a transcoder that converts from a legacy encoding to a Unicode encoding form and ensures that the result is in Unicode Normalization Form C (see UAX #15 [Unicode]).]

[Definition: A **character escape** is a syntactic device defined in a markup or programming language that allows one or more of:]

1. expressing syntax-significant characters while disregarding their significance in the syntax of the language, or
2. expressing characters not representable in the character encoding chosen for an instance of the language, or
3. expressing characters in general, without use of the corresponding character codes.

[Definition: **Certified** text is text which satisfies at least one of the following conditions:]

1. it has been confirmed through inspection that the text is in normalized form
2. the source text-processing component is identified and is known to produce only normalized text.

[Definition: Text is, for the purposes of this specification, **Unicode-normalized** if it is in a Unicode encoding form and is in Unicode Normalization Form C, according to a version of Unicode Standard Annex #15: Unicode Normalization Forms [Unicode] at least as recent as the oldest version of the Unicode Standard that contains all the characters actually present in the text, but no earlier than version 3.2.]

[Definition: Text is **include-normalized** if:]

1. the text is Unicode-normalized and does not contain any character escapes or includes whose expansion would cause the text to become no longer Unicode-normalized; or
2. the text is in a legacy encoding and, if it were transcoded to a Unicode encoding form by a normalizing transcoder, the resulting text would satisfy clause 1 above.

[Definition: A **composing character** is a character that is one or both of the following:]

1. the second character in the canonical decomposition mapping of some primary composite (as defined in D3 of UAX #15 [Unicode]), or
2. of non-zero canonical combining class (as defined in Unicode [Unicode]).

[Definition: Text is **fully-normalized** if:]

1. the text is in a Unicode encoding form, is include-normalized and none of the relevant constructs comprising the text begin with a composing character or a character escape representing a composing character; or
2. the text is in a legacy encoding and, if it were transcoded to a Unicode encoding form by a normalizing transcoder, the resulting text would satisfy clause 1 above.

C Expansion of Entity and Character References (Non-Normative)

This appendix contains some examples illustrating the sequence of entity- and character-reference recognition and expansion, as specified in **4.4 XML Processor Treatment of Entities and References**.

If the DTD contains the declaration

```
<!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;#38;) or with a general entity
(&amp;amp;).</p>" >
```

then the XML processor will recognize the character references when it parses the entity declaration, and resolve them before storing the following string as the value of the entity "`example`":

```
<p>An ampersand (&#38;) may be escaped
```

```
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>
```

A reference in the document to "`&example;`" will cause the text to be reparsed, at which time the start- and end-tags of the p element will be recognized and the three references will be recognized and expanded, resulting in a p element with the following content (all data, no delimiters or markup):

```
An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;).
```

A more complex example will illustrate the rules and their effects fully. In the following example, the line numbers are solely for reference.

```
1 <?xml version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60;!ENTITY tricky "error-prone" >' >
6 %xx;
7 ]>
8 <test>This sample shows a &tricky; method.</test>
```

This produces the following:

- in line 4, the reference to character 37 is expanded immediately, and the parameter entity "`xx`" is stored in the symbol table with the value "`%zz;`". Since the replacement text is not rescanned, the reference to parameter entity "`zz`" is not recognized. (And it would be an error if it were, since "`zz`" is not yet declared.)
- in line 5, the character reference "`&#60;`" is expanded immediately and the parameter entity "`zz`" is stored with the replacement text "`<!ENTITY tricky "error-prone" >`", which is a well-formed entity declaration.
- in line 6, the reference to "`xx`" is recognized, and the replacement text of "`xx`" (namely "`%zz;`") is parsed. The reference to "`zz`" is recognized in its turn, and its replacement text ("`<!ENTITY tricky "error-prone" >`") is parsed. The general entity "`tricky`" has now been declared, with the replacement text "`error-prone`".
- in line 8, the reference to the general entity "`tricky`" is recognized, and it is expanded, so the full content of the `test` element is the self-describing (and ungrammatical) string *This sample shows a error-prone method.*

D Deterministic Content Models (Non-Normative)

As noted in **3.2.1 Element Content**, it is required that content models in element type declarations be deterministic. This requirement is for compatibility with SGML (which calls deterministic content models "unambiguous"); XML processors built using SGML systems may flag non-deterministic content models as errors.

For example, the content model `((b, c) | (b, d))` is non-deterministic, because given an initial b the XML processor cannot know which b in the model is being matched without looking ahead to see which element follows the b. In this case, the two references to b can be collapsed into a single reference, making the model read `(b, (c | d))`. An initial b now clearly matches only a single name in the content model. The processor doesn't need to look ahead to see what follows; either c or d would be accepted.

More formally: a finite state automaton may be constructed from the content model using the standard algorithms, e.g. algorithm 3.5 in section 3.9 of Aho, Sethi, and Ullman [Aho/Ullman]. In many such algorithms, a follow set is constructed for each position in the regular expression (i.e., each leaf node in the syntax tree for the regular expression); if any position has a follow set in which more than one following position is labeled with the same element type name, then the content model is in error and may be reported as an error.

Algorithms exist which allow many but not all non-deterministic content models to be reduced automatically to equivalent deterministic models; see Brüggemann-Klein 1991 [Brüggemann-Klein].

E Autodetection of Character Encodings (Non-Normative)

The XML encoding declaration functions as an internal label on each entity, indicating which character encoding is in use. Before an XML processor can read the internal label, however, it apparently has to know what character encoding is in use — which is what the internal label is trying to indicate. In the general case, this is a hopeless situation. It is not entirely hopeless in XML, however, because XML limits the general case in two ways: each implementation is assumed to support only a finite set of character encodings, and the XML encoding declaration is restricted in position and content in order to make it feasible to autodetect the character encoding in use in each entity in normal cases. Also, in many cases other sources of information are available in addition to the XML data stream itself. Two cases may be distinguished, depending on whether the XML entity is presented to the processor without, or with, any accompanying (external) information. We consider the first case first.

E.1 Detection Without External Encoding Information

Because each XML entity not accompanied by external encoding information and not in UTF-8 or UTF-16 encoding must begin with an XML encoding declaration, in which the first characters must be '`<?xml`', any conforming processor can detect, after two to four octets of input, which of the following cases apply. In reading this list, it may help to know that in UCS-4, '<' is "`#x0000003C`" and '?' is "`#x0000003F`", and the Byte Order Mark required of UTF-16 data streams is "`#xFEFF`". The notation ## is used to denote any byte value except that two consecutive ##s cannot be both 00.

With a Byte Order Mark:

00 00 FE FF	UCS-4, big-endian machine (1234 order)
FF FE 00 00	UCS-4, little-endian machine (4321 order)
00 00 FF FE	UCS-4, unusual octet order (2143)
FE FF 00 00	UCS-4, unusual octet order (3412)
FE FF ## ##	UTF-16, big-endian
FF FE ## ##	UTF-16, little-endian
EF BB BF	UTF-8

Without a Byte Order Mark:

00 00 00 3C 3C 00 00 00 00 00 3C 00 00 3C 00 00	UCS-4 or other encoding with a 32-bit code unit and ASCII characters encoded as ASCII values, in respectively big-endian (1234), little-endian (4321) and two unusual byte orders (2143 and 3412). The encoding declaration must be read to determine which of UCS-4 or other supported 32-bit encodings applies.
00 3C 00 3F	UTF-16BE or big-endian ISO-10646-UCS-2 or other encoding with a 16-bit code unit in big-endian order and ASCII characters encoded as ASCII values (the encoding declaration must be read to determine which)
3C 00 3F 00	UTF-16LE or little-endian ISO-10646-UCS-2 or other encoding with a 16-bit code unit in little-endian order and ASCII characters encoded as ASCII values (the encoding declaration must be read to determine which)
3C 3F 78 6D	UTF-8, ISO 646, ASCII, some part of ISO 8859, Shift-JIS, EUC, or any other 7-bit, 8-bit, or mixed-width encoding which ensures that the characters of ASCII have their normal positions, width, and values; the actual encoding declaration must be read to detect which of these applies, but since all of these encodings use the same bit patterns for the relevant ASCII characters, the encoding declaration itself may be read reliably
4C 6F A7 94	EBCDIC (in some flavor; the full encoding declaration must be read to tell which code page is in use)
Other	UTF-8 without an encoding declaration, or else the data stream is mislabeled (lacking a required encoding declaration), corrupt, fragmentary, or enclosed in a wrapper of some kind

Note:

In cases above which do not require reading the encoding declaration to determine the encoding, section 4.3.3 still requires that the encoding declaration, if present, be read and that the encoding name be checked to match the actual encoding of the entity. Also, it is possible that new character encodings will be invented that will make it necessary to use the encoding declaration to determine the encoding, in cases where this is not required at present.

This level of autodetection is enough to read the XML encoding declaration and parse the character-encoding identifier, which is still necessary to distinguish the individual members of each family of encodings (e.g. to tell UTF-8 from 8859, and the parts of 8859 from each other, or to distinguish the specific EBCDIC code page in use, and so on).

Because the contents of the encoding declaration are restricted to characters from the ASCII repertoire (however encoded), a processor can reliably read the entire encoding declaration as soon as it has detected which family of encodings is in use. Since in practice, all widely used character encodings fall into one of the categories above, the XML encoding declaration allows reasonably reliable in-band labeling of character encodings, even when external sources of information at the operating-system or transport-protocol level are unreliable. Character encodings such as UTF-7 that make overloaded usage of ASCII-valued bytes may fail to be reliably detected.

Once the processor has detected the character encoding in use, it can act appropriately, whether by invoking a separate input routine for each case, or by calling the proper conversion function on each character of input.

Like any self-labeling system, the XML encoding declaration will not work if any software changes the entity's character set or encoding without updating the encoding declaration. Implementors of character-encoding routines should be careful to ensure the accuracy of the internal and external information used to label the entity.

E.2 Priorities in the Presence of External Encoding Information

The second possible case occurs when the XML entity is accompanied by encoding information, as in some file systems and some network protocols. When multiple sources of information are available, their relative priority and the preferred method of handling conflict should be specified as part of the higher-level protocol used to deliver XML. In particular, please refer to [IETF RFC 3023] or its successor, which defines the `text/xml` and `application/xml` MIME types and provides some useful guidance. In the interests of interoperability, however, the following rule is recommended.

- If an XML entity is in a file, the Byte-Order Mark and encoding declaration are used (if present) to determine the character encoding.

F W3C XML Working Group (Non-Normative)

This specification was prepared and approved for publication by the W3C XML Working Group (WG). WG approval of this specification does not necessarily imply that all WG participants voted for its approval. The current and former members in the XML WG are:

- Jon Bosak, Sun (*Chair*)
- James Clark (*Technical Lead*)
- Tim Bray, Textuality and Netscape (*XML Co-editor*)
- Jean Paoli, Microsoft (*XML Co-editor*)
- C. M. Sperberg-McQueen, U. of Ill. (*XML Co-editor*)
- Dan Connolly, W3C (*W3C Liaison*)
- Paula Angerstein, Texcel

- Steve DeRose, INSO
- Dave Hollander, HP
- Eliot Kimber, ISOGEN
- Eve Maler, ArborText
- Tom Magliery, NCSA
- Murray Maloney, SoftQuad, Grif SA, Muzmo and Veo Systems
- MURATA Makoto (FAMILY Given), Fuji Xerox Information Systems
- Joel Nava, Adobe
- Conleth O'Connell, Vignette
- Peter Sharpe, SoftQuad
- John Tigue, DataChannel

G W3C XML Core Working Group (Non-Normative)

The present edition of this specification was prepared by the W3C XML Core Working Group (WG). The participants in the WG at the time of publication of this edition were:

- Leonid Arbouzov, Sun Microsystems
- Mary Brady
- John Cowan (*XML 1.1 First Edition Editor*)
- John Evdemon, Microsoft
- Andrew Fang, Arbortext
- Paul Grosso, Arbortext (*Co-Chair*)
- Arnaud Le Hors, IBM
- Dmitry Lenkov, Oracle
- Anjana Manian, Oracle
- Glenn Marcy, IBM
- Jonathan Marsh, Microsoft
- Sandra Martinez, NIST
- Liam Quin, W3C (*Staff Contact*)
- Lew Shannon
- Richard Tobin, University of Edinburgh
- Daniel Veillard
- Norman Walsh, Sun Microsystems (*Co-Chair*)
- François Yergeau

H Production Notes (Non-Normative)

This edition was encoded in a slightly modified version of the XMLspec DTD, 2.5. The XHTML versions were produced with a combination of the xmlspec.xsl, diffspec.xsl, and REC-xml-3e.xsl XSLT stylesheets.

I Suggestions for XML Names (Non-Normative)

The following suggestions define what is believed to be best practice in the construction of XML names used as element names, attribute names, processing instruction targets, entity names, notation names, and the values of attributes of type ID, and are intended as guidance for document authors and schema designers. All references to Unicode are understood with respect to a particular version of the Unicode Standard greater than or equal to 3.0; which version should be used is left to the discretion of the document author or schema designer.

The first two suggestions are directly derived from the rules given for identifiers in the Unicode Standard, version 3.0, and exclude all control characters, enclosing nonspacing marks, non-decimal numbers, private-use characters, punctuation characters (with the noted exceptions), symbol characters, unassigned codepoints, and white space characters. The other suggestions are mostly derived from [XML-1.0] Appendix B.

1. The first character of any name should have a Unicode General Category of Ll, Lu, Lo, Lm, Lt, or Nl, or else be '_' #x5F.
2. Characters other than the first should have a Unicode General Category of Ll, Lu, Lo, Lm, Lt, Mc, Mn, Nl, Nd, Pc, or Cf, or else be one of the following: '-' #x2D, '.' #x2E, ':' #x3A or '·' #xB7 (middle dot). Since Cf characters are not directly visible, they should be employed with caution and only when necessary, to avoid creating names which are distinct to XML processors but look the same to human beings.
3. Ideographic characters which have a canonical decomposition (including those in the ranges [#xF900-#xFAFF] and [#x2F800-#x2FFFD], with 12 exceptions) should not be used in names.
4. Characters which have a compatibility decomposition (those with a "compatibility formatting tag" in field 5 of the Unicode Character Database -- marked by field 5 beginning with a "<") should not be used in names. This suggestion does not apply to #x0E33 THAI CHARACTER SARA AM or #x0EB3 LAO CHARACTER AM, which despite their compatibility decompositions are in regular use in those scripts.
5. Combining characters meant for use with symbols only (including those in the ranges [#x20D0-#x20EF] and [#x1D165-#x1D1AD]) should not be used in names.
6. The interlinear annotation characters ([#xFFF9-#xFFFB) should not be used in names.
7. Variation selector characters should not be used in names.
8. Names which are nonsensical, unpronounceable, hard to read, or easily confusable with other names should not be employed.

Namensräume in XML

Deutsche Übersetzung

18. Juni 2001

Diese Version:

http://www.schumacher-netz.de/TR/1999/REC-xml-names-19990114-de.html

Übersetzer:

Stefan Schumacher, schumacher-netz.de <sts@schumacher-netz.de>

Namensräume in XML

W3C-Empfehlung 14. Januar 1999

Diese Version:

http://www.w3.org/TR/1999/REC-xml-names-19990114 http://www.w3.org/TR/1999/REC-xml-names-19990114/xml-names.xml http://www.w3.org/TR/1999/REC-xml-names-19990114/Overview.html

Aktuelle Version:

http://www.w3.org/TR/REC-xml-names

Vorherige Version:

http://www.w3.org/TR/1998/PR-xml-names-19981117

Editoren:

Tim Bray , Textuality <tbray@textuality.com>
Dave Hollander , Hewlett-Packard Company <dmh@corp.hp.com>
Andrew Layman , Microsoft <andrewl@microsoft.com>

Zusammenfassung

XML-Namensräume bieten eine einfache Möglichkeit, um Element- und Attributnamen, die in "Extensible Markup Language"-Dokumenten verwendet werden können, eindeutig zu benennen. Die Element- und Attributnamen werden mit Namensräumen verknüpft, die durch URI-Verweise identifiziert werden.

Status dieses Dokuments

Dieses Dokument wurde von Mitgliedern des W3C und anderen interessierten Gruppen überprüft und vom Direktor als W3C-Empfehlung gebilligt. Es ist ein stabiles Dokument und darf als Referenzmaterial verwendet werden oder als normative Referenz von anderen Dokumenten zitiert werden. Die Intention des W3C bei der Erstellung dieser Empfehlung ist es, das Interesse an dieser Spezifikation zu wecken und ihre weitläufige Verbreitung zu fördern. Dies erhöht die Funktionalität und die Interoperabilität des Webs.

Die Liste der bekannten Fehler in dieser Spezifikation ist verfügbar unter http://www.w3.org/XML/xml-names-19990114-errata.

Bitte senden Sie Berichte über Fehler in diesem Dokument an xml-names-

editor@w3.org.

Inhaltsverzeichnis

Anhänge

---

1 Beweggründe und Überblick

Wir betrachten Anwendungen der Extensible Markup Language (XML), in denen ein einzelnes XML-Dokument Elemente und Attribute (hier "Markup-Vokabular" genannt) enthalten kann, die für verschiedene Software-Module definiert sind und von verschiedenen Software-Modulen verwendet werden. Eine Motivation dafür ist Modularität; wenn ein Markup-Vokabular existiert, das gut verstanden wird und für welches nützliche Software vorhanden ist, ist es besser, dieses Markup wieder zu verwenden als es neu zu erfinden.

In Dokumenten, die unterschiedliches Markup-Vokabular enthalten, können Probleme mit der Erkennung und Kollisionen auftreten. Software-Module müssen die Tags und Attribute erkennen, für deren Verarbeitung sie geschaffen wurden, auch im Fall einer Kollision, wenn Markup, das für eine andere Software geschrieben wurde, die gleichen Elementtypen und

Attributnamen verwendet.

Diese Überlegungen erfordern, dass Dokumentkonstrukte, deren Geltungsbereich über den des beinhaltenden Dokuments hinausgeht, einzigartige Namen haben sollten. Diese Spezifikation beschreibt einen Mechanismus, *XML-Namensräume*, der diese Anforderungen erfüllt.

[Definition: Ein **XML-Namensraum** ist eine Zusammenstellung von Namen, identifiziert durch einen URI-Verweis [RFC2396], die in XML-Dokumenten als Elementtypen und Attributnamen verwendet werden.] XML-Namensräume unterscheiden sich von den "Namensräumen", die normalerweise im Computerbereich verwendet werden, in dem Maße, dass die XML-Version eine interne Struktur hat und im mathematischen Sinne keine Zusammensetzung ist. Dieses Thema wird in **A Die interne Struktur von XML-Namensräumen** besprochen.

[Definition: URI-Verweise, die Namensräume identifizieren, werden als **identisch** angesehen, wenn sie Zeichen für Zeichen genau gleich sind.] Beachten Sie, dass URI-Verweise, die in diesem Sinne nicht identisch sind, trotzdem die gleiche Funktionalität besitzen können. Zum Beispiel seien URI-Verweise genannt, die sich nur in der Groß- und Kleinschreibung unterscheiden, oder externe Entities, die unterschiedliche effektive Base-URIs haben.

Namen aus XML-Namensräumen können als qualifizierte Namen erscheinen, die einen einzelnen Doppelpunkt enthalten, der den Namen in ein Namensraum-Präfix und einen lokalen Teil aufteilt. Das Präfix, das Platzhalter für einen URI-Verweis ist, wählt einen Namensraum aus. Die Kombination aus dem allumfassend verwalteten URI-Namensraum und dem eigenen Namensraum des Dokuments erzeugt einen Identifier, der einzigartig ist. Mechanismen für die Präfixbereiche und Voreinstellungen werden gegeben.

URI-Verweise können Zeichen enthalten, die nicht in Namensraumnamen erlaubt sind, also nicht direkt als Namensraum-Präfixe benutzt werden können. Deshalb fungiert das Namensraum-Präfix als Proxy für einen URI-Verweis. Eine Attribut-basierte Syntax, wie unten beschrieben, wird verwendet, um die Verbindung zwischen dem Namensraum-Präfix und dem URI-Verweis zu deklarieren; Software, die dieser Spezifikation gerecht wird, muss diese Deklarationen und Präfixe erkennen und verarbeiten.

1.1 Eine Bemerkung zur Notation und zur Verwendung

Beachten Sie, dass viele Nicht-Terminale in den Produktionen dieser Spezifikation nicht hier, sondern in der XML-Spezifikation [XML] definiert sind.

Wenn hier definierte Nicht-Terminale den gleichen Namen haben wie Nicht-Terminale, die in der XML-Spezifikation definiert sind, unterliegen die hiesigen Produktionen in allen Fällen einer Untermenge der Strings, denen die entsprechenden Produktionen dort unterliegen würden.

In den Produktionen dieses Dokuments ist NSC ein Namensraumzwang, also eine der Regeln, die von Dokumenten befolgt werden müssen, die konform zu dieser Spezifikation sind.

Beachten Sie, dass alle Internet-Domain-Namen, die in diesen Beispielen verwendet werden, mit Ausnahme von `w3.org` zufällig ausgewählt sind und nicht so verstanden werden sollen, als würden sie Inhalte importieren.

2 Namensräume deklarieren

[Definition: Ein Namensraum wird **deklariert**, indem reservierte Attribute verwendet werden. Ein Attributname muss entweder **xmlns** sein oder es muss ein Präfix **xmlns:** verwendet werden. Diese Attribute müssen, wie jedes andere XML-Attribut, direkt oder durch eine Voreinstellung zur Verfügung gestellt werden.]

Attributnamen für Namensraum-Deklarationen

```
[1]   NSAttName          ::  PrefixedAttName
                         =
                             | DefaultAttName
[2]   PrefixedAttName    ::  'xmlns:' NCName
                         =
[3]   DefaultAttName     ::  'xmlns'
                         =
[4]   NCName             ::  (Letter | '_')          /* Ein
                         =   (NCNameChar)*           XML-
                                                     Name
                                                     ohne den
                                                     ":" */
[5]   NCNameChar         ::  Letter | Digit | '.' |
                         =   '-' | '_' |
                             CombiningChar | Extender
```

[Definition: Der Wert des Attributs, ein URI-Verweis, ist der **Namensraumname**, der den Namensraum identifiziert.] Der Namensraumname sollte, um seinen Zweck zu erfüllen, einzigartig und

dauerhaft sein. Es ist nicht notwendig, dass er direkt für den Empfang eines Schemas (sofern eines existiert) verwendet werden kann. Uniform Resource Names [RFC2141] sind ein Beispiel für eine Syntax, die mit diesen Zielen entwickelt wurde. Es soll jedoch erwähnt werden, dass auch normale URLs so verwendet werden können, dass sie diesen gleichen Vorstellungen entsprechen.

[Definition: Entspricht der Attributname dem PrefixedAttName, gibt der NCName das **Namensraum-Präfix** an, das verwendet wird, um die Element- und Attributnamen mit dem Namensraumnamen zu verbinden, der im Attributwert des mit der Deklaration verbundenen Elements angegeben ist.] In solchen Deklarationen sollte der Namensraumname nicht leer sein.

[Definition: Entspricht der Attributname dem DefaultAttName, dann ist der Namensraumname im Attributwert der Name des **voreingestellten Namensraums** im Geltungsbereich des mit der Deklaration verbundenen Elements.] In solch einer voreingestellten Deklaration kann der Attributwert leer sein. Mehr zu voreingestellten Namensräumen und dem Überschreiben von Deklarationen wird in **5 Elementen und Attributen Namensräume zuweisen** besprochen.

Das Beispiel einer Namensraum-Deklaration, die das Präfix **edi** mit dem Namensraumnamen `http://ecommerce.org/schema` verbindet:

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <!-- Das "edi"-Präfix wird für das Element "x" und
Inhalt
       an http://ecommerce.org/schema gebunden. -->
</x>
```

Namensraumzwang: Führendes "XML"

Präfixe, die mit den drei Buchstaben `x`, `m`, `l` beginnen, in jeglicher Kombination von Groß- und Kleinschreibung, sind für die Verwendung von XML oder XML-verwandten Spezifikationen reserviert.

3 Qualifizierte Namen

[Definition: In XML-Dokumenten, die konform zu dieser Spezifikation sind, können einige Namen (Konstrukte, die dem Nicht-Terminal Name entsprechen) als **qualifizierte Namen** vergeben werden, die wie folgt definiert sind:

Qualifizierter Name

```
[6]   QName      ::=  (Prefix ':')? LocalPart
[7]   Prefix     ::=  NCName
[8]   LocalPart  ::=  NCName
```

]

Das "Prefix" stellt den Part des Namensraum-Präfixes des qualifizierten Namens zur Verfügung und muss mit einem Namensraum-URI-Verweis durch eine Namensraum-Deklaration verbunden werden. [Definition: Der "LocalPart" stellt den **lokalen Teil** des qualifizierten Namens zur Verfügung.]

Beachten Sie, dass das Präfix *nur* als Platzhalter für den Namensraumnamen steht. Anwendungen sollten den Namensraumnamen verwenden, nicht das Präfix, wenn sie Namen erstellen, deren Geltungsbereich über den des beinhaltenden Dokuments hinausgeht.

4 Verwendung von qualifizierten Namen

In XML-Dokumenten, die konform zu dieser Spezifikation sind, werden Elementtypen als qualifizierte Namen wie folgt gegeben:

Elementtypen

```
[9]    STag          ::=  '<' QName (S Attribute)* S? '>'
[10]   ETag          ::=  '</' QName S? '>'
[11]   EmptyElemTag  ::=  '<' QName (S Attribute)* S? '/>'
```

Ein Beispiel eines qualifizierten Namens, der als Elementtyp dient:

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <!-- Der Namensraum des Elements 'preis' ist http://
ecommerce.org/schema -->
  <edi:preis einheit='Euro'>32.18</edi:preis>
</x>
```

Attribute sind entweder Namensraum-Deklarationen oder ihre Namen werden als qualifizierte Namen angegeben:

Attribut

```
[12]  Attribute  ::=  NSAttName Eq AttValue
                    | QName Eq AttValue
```

Ein Beispiel eines qualifizierten Namens, der als Attributname dient:

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <!-- Der Namensraum des Attributs 'taxClass' ist
http://ecommerce.org/schema -->
  <lineItem edi:taxClass="exempt">Babynahrung</
lineItem>
</x>
```

Namensraumzwang: Deklariertes Präfix

Das Namensraum-Präfix muss, es sei denn, es ist `xml` oder `xmlns`, in einem Namensraum-Deklarations-Attribut deklariert worden sein, entweder im Start-Tag des Elements, in dem das Präfix benutzt wird, oder in einem Elternelement (z.B. in einem Element, in dessen Inhalt das vorangestellte Markup (Präfix) auftaucht). Das Präfix `xml` ist laut Definition an den Namensraumnamen `http://www.w3.org/XML/1998/namespace` gebunden. Das Präfix `xmlns` wird nur für Namensraumeinbindungen verwendet und ist selbst nicht an irgendeinen Namensraum gebunden.

Dieser Zwang kann in der Praxis zu Schwierigkeiten führen, wenn die Namensraum-Deklaration nicht direkt im XML-Dokument-Entity angegeben wird, sondern über einen voreingestellten Wert in einem externen Entity. Diese Deklarationen können eventuell nicht von Software verarbeitet werden, die auf nicht validierenden XML-Prozessoren aufbaut. Viele XML-Anwendungen, vermutlich auch Namensraum-sensitive, versäumen es, validierende Prozessoren zu fordern. Um eine korrekte Zusammenarbeit mit solchen Anwendungen zu gewährleisten, müssen Namensraum-Deklarationen entweder direkt oder über voreingestellte Attribute im internen Subset der DTD angegeben werden.

Elementnamen und Attributtypen werden auch als qualifizierte Namen angegeben, wenn sie in Deklarationen innerhalb der DTD auftauchen:

Qualifizierte Namen in Deklarationen

```
[13]   doctypedecl   ::=   '<!DOCTYPE' S QName (S
                           ExternalID)? S? ('[' (markupdecl |
                           PEReference | S)* ']' S?)? '>'
[14]   elementdecl   ::=   '<!ELEMENT' S QName S contentspec
                           S? '>'
[15]   cp            ::=   (QName | choice | seq) ('?' | '*'
                           | '+')?
[16]   Mixed         ::=   '(' S? '#PCDATA' (S? '|' S? QName)
                           * S? ')*'
                           | '(' S? '#PCDATA' S? ')'
[17]   AttlistDecl   ::=   '<!ATTLIST' S QName AttDef* S? '>'
[18]   AttDef        ::=   S (QName | NSAttName) S AttType S
                           DefaultDecl
```

5 Elementen und Attributen Namensräume zuweisen

5.1 Geltungsbereich für Namensräume

Die Namensraum-Deklaration gilt für das Element, in dem sie angegeben ist, und für alle Elemente im Inhalt dieses Elements, es sei denn, sie wird durch eine andere Namensraum-Deklaration mit dem gleichen NSAttName-Teil überschrieben:

```
<?xml version="1.0"?>
<!-- alle Elemente hier sind explizit im HTML-
Namensraum -->
<html:html xmlns:html='http://www.w3.org/TR/REC-
html40'>
  <html:head><html:title>Frobnostication</html:title></
html:head>
  <html:body><html:p>Moved to
    <html:a href='http://frob.com'>here.</html:a></
html:p></html:body>
</html:html>
```

Mehrere Namensraum-Präfixe können als Attribute eines einzelnen Elements deklariert werden, wie in diesem Beispiel beschrieben:

```
<?xml version="1.0"?>
<!-- beide Namensraum-Präfixe sind durchgängig
verfügbar -->
<bk:book xmlns:bk='urn:loc.gov:books'
         xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <bk:title>Cheaper by the Dozen</bk:title>
    <isbn:number>1568491379</isbn:number>
</bk:book>
```

5.2 Voreinstellung des Namensraums

Ein voreingestellter Namensraum soll für das Element gelten, in dem er deklariert ist (sofern das Element kein Namensraum-Präfix hat), und für alle Elemente ohne Präfix im Inhalt des Elements. Wenn der URI-Verweis in der voreingestellten Namensraum-Deklaration leer ist, sollten Elemente ohne Präfix im Geltungsbereich der Deklaration so angesehen werden, als wären sie in keinem Namensraum. Beachten Sie, dass voreingestellte Namensräume nicht direkt für Attribute gelten.

```
<?xml version="1.0"?>
<!-- Elemente sind im HTML-Namensraum, in diesem Fall
voreingestellt -->
<html xmlns='http://www.w3.org/TR/REC-html40'>
  <head><title>Frobnostication</title></head>
  <body><p>Moved to
    <a href='http://frob.com'>here</a>.</p></body>
</html>
```

```
<?xml version="1.0"?>
<!-- Elementtypen ohne Präfix gehören zu "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <title>Cheaper by the Dozen</title>
    <isbn:number>1568491379</isbn:number>
</book>
```

Ein umfassenderes Beipiel für Namensraumgeltungsbereiche:

```
<?xml version="1.0"?>
<!-- anfangs ist der voreingestellte
Namensraum "books" -->
<book xmlns='urn:loc.gov:books'
      xmlns:isbn='urn:ISBN:0-395-36341-6'>
```

```
    <title>Cheaper by the Dozen</title>
    <isbn:number>1568491379</isbn:number>
    <notes>
      <!-- HTML wird zum voreingestellten Namensraum
für einige Kommentare -->
      <p xmlns='urn:w3-org-ns:HTML'>
          This is a <i>funny</i> book!
      </p>
    </notes>
</book>
```

Der voreingestellte Namensraum kann auf einen leeren String gesetzt werden. Das hat innerhalb des Geltungsbereichs der Deklaration den gleichen Effekt, als wäre kein voreingestellter Namensraum gegeben.

```
<?xml version='1.0'?>
<Beers>
  <!-- der voreingestellte Namensraum ist der von HTML
-->
  <table xmlns='http://www.w3.org/TR/REC-html40'>
   <th><td>Name</td><td>Origin</td><td>Description</
td></th>
   <tr>
     <!-- kein voreingestellter Namensraum in
Tabellenzellen -->
     <td><brandName xmlns="">Huntsman</brandName></td>
     <td><origin xmlns="">Bath, UK</origin></td>
     <td>
       <details xmlns=""><class>Bitter</
class><hop>Fuggles</hop>
         <pro>Wonderful hop, light alcohol, good
summer beer</pro>
         <con>Fragile; excessive variance pub to pub</
con>
         </details>
       </td>
     </tr>
   </table>
  </Beers>
```

5.3 Einzigartigkeit von Attributen

In XML-Dokumenten, die konform zu dieser Spezifikation sind, sollte kein Tag zwei Attribute enthalten, die

1. identische Namen haben, oder
2. qualifizierte Namen mit dem gleichen lokalen Namensteil und mit Präfixen, die mit identischen Namensraumnamen verbunden sind.

Zum Beispiel ist jedes Einzelne der Start-Tags `bad` nicht erlaubt:

```
<!-- http://www.w3.org ist an n1 und n2 gebunden -->
<x xmlns:n1="http://www.w3.org"
   xmlns:n2="http://www.w3.org" >
  <bad a="1"     a="2" />
  <bad n1:a="1"  n2:a="2" />
</x>
```

Jedoch ist jedes der folgenden Start-Tags `good` erlaubt. Das zweite Start-Tag ist zulässig, weil der voreingestellte Namensraum nicht für Attribute gilt:

```
<!-- http://www.w3.org ist an n1 gebunden und
voreingestellt -->
<x xmlns:n1="http://www.w3.org"
   xmlns="http://www.w3.org" >
  <good a="1"     b="2" />
  <good a="1"     n1:a="2" />
</x>
```

6 Konformität von Dokumenten

In XML-Dokumenten, die konform zu dieser Spezifikation sind, müssen Elementtypen und Attributnamen der Bauanleitung für QName entsprechen und die Namensraumzwänge erfüllen.

Ein XML-Dokument ist konform zu dieser Spezifikation, wenn alle anderen Tokens im Dokument, die aus Gründen der XML-Konformität der XML-Bauanleitung für Name entsprechen müssen, auch der in dieser Spezifikation gegebenen Bauanleitung für NCName entsprechen.

Konformität bedeutet für ein Dokument, dass

- alle Elementtypen und Attributnamen entweder keinen oder einen Doppelpunkt enthalten,
- Entity-Namen, PI-Ziele oder Notations-Namen keinen Doppelpunkt enthalten.

Streng genommen sind Attributwerte, die als **ID**, **IDREF(S)**, **ENTITY(IES)** und **NOTATION** deklariert sind, ebenfalls Namen und sollten keine Doppelpunkte enthalten. Jedoch ist der deklarierte Typ eines Attributwerts nur Prozessoren bekannt, welche die Markup-Deklarationen lesen. Ein Beispiel sind validierende Prozessoren. Deshalb ist nicht sicher, ob der Inhalt des Attributwerts auf Konformität zu dieser Spezifikation überprüft worden ist, auch wenn der Gebrauch von validierenden Prozessoren spezifiziert ist.

A Die interne Struktur von XML-Namensräumen (nicht normativ)

A.1 Die Unzulänglichkeit der traditionellen Namensräume

In der Computersprache bezieht sich der Begriff Namensraum normalerweise auf eine *Zusammenstellung* von Namen, z.B. einer Zusammenstellung ohne Duplikate. Werden die Namen in XML-Markup jedoch mit dieser Bedeutung von Namensraum verwendet, wird ein großer Teil ihres Potenzials vernachlässigt. Der vorrangige Nutzen solcher Namen in XML-Dokumenten ist die Indentifikation von logischen Strukturen in Dokumenten durch Software-Module wie Abfrage-Prozessoren, Style Sheet erkennende Render-Engines und Schema erkennende Validatoren. Erwägen Sie das folgende Beispiel:

```
<section><title>Book-Signing Event</title>
<signing>
  <author title="Mr" name="Vikram Seth" />
  <book title="A Suitable Boy" price="$22.95" /></
signing>
<signing>
  <author title="Dr" name="Oliver Sacks" />
  <book title="The Island of the Color-Blind"
price="$12.95" /></signing>
</section>
```

In diesem Beispiel taucht der Name `title` dreimal im Markup auf und der Name alleine bietet offensichtlich keine ausreichenden Informationen für eine korrekte Verarbeitung durch ein Software-Modul.

Ein weiterer problematischer Bereich rührt her vom Gebrauch globaler Attribute, wie in diesem Beispiel gezeigt. Ein Ausschnitt aus einem XML-Dokument, welches mit Hilfe von CSS Style Sheets dargestellt werden soll:

```
<RESERVATION>
 <NAME HTML:CLASS="largeSansSerif">Layman, A</NAME>
 <SEAT CLASS="Y" HTML:CLASS="reallyImportant">33B</
SEAT>
 <DEPARTURE>1997-05-24T07:55:00+1</DEPARTURE></
RESERVATION>
```

In diesem Fall unterscheidet sich das CLASS-Attribut, welches die Gebührengrundlage beschreibt und Werte wie "J", "Y" und "C" annimmt, in allen semantischen Bereichen vom Attribut HTML:CLASS, welches verwendet wird, um syntaktische Vielfalt in HTML zu simulieren, indem das begrenzte Repertoire an Elementen durch Unterklassen umgangen wird.

XML 1.0 bietet keinen eigenen Weg, globale Attribute zu deklarieren; Dinge wie das HTML-Attribut CLASS sind nur durch ihre selbsterklärende Beschreibung und durch die Interpretation durch HTML-Anwendungen global. Jedoch kann man beobachten, dass solche Attribute, deren wichtiges unterscheidendes Merkmal ist, dass ihre Namen einmalig sind, in vielen unterschiedlichen Anwendungen vorkommen.

A.2 XML-Namensraum-Partitionen

Um das Ziel zu unterstützen, dass sowohl qualifizierte als auch nicht qualifizierte Namen ihren beabsichtigten Zweck erfüllen, identifizieren wir die Namen, die in XML-Namensräumen vorkommen, als würden sie zu einem von mehreren auseinander genommenen traditionellen (z.B. set-strukturierten) Namensräumen gehören, welche sich Namensraum-Partitionen nennen. Diese Partitionen sind:

Die All Element Types Partition

Alle Elementtypen in einem XML-Namensraum gehören zu dieser Partition. Jeder hat einen einzigartigen lokalen Teil; die Kombination aus Namensraumname und dem lokalen Teil identifiziert den Elementtyp eindeutig.

Die Global Attribute Partition

Diese Partition enthält die Namen aller Attribute, die für diesen Namensraum als global definiert sind. Die einzige geforderte Charakteristik des globalen Attributs ist, dass dessen Name einzigartig in der Global Attribute Partition ist. Diese Spezifikation macht keine Vorschriften in Bezug auf die geeignete Verwendung dieser Attribute. Die Kombination des Namensraumnamens mit dem Attributnamen

identifiziert das globale Attribut eindeutig.

Die Per Element Type Partitions

Jeder Typ in der All Element Types Partition hat einen zugeordneten Namensraum, in dem die Namen der nicht qualifizierten Attribute erscheinen, die für das Element vorgesehen sind. Dies ist ein traditioneller Namensraum, weil das Vorhandensein von mehreren gleichen Attributnamen in einem Element laut XML 1.0 verboten ist. Die Kombination des Attributnamens mit dem Typ und dem Namensraumnamen des Elements identifiziert jedes nicht qualifizierte Attribut.

In XML-Dokumenten, die konform zu dieser Spezifikation sind, sind die Namen aller qualifizierten Attribute der Global Attribute Partition zugewiesen und die Namen aller nicht qualifizierten Attribute sind der passenden Per-Element-Type Partion zugewiesen.

A.3 Erweiterte Elementtypen und Attributnamen

Um einfacher Regeln spezifizieren und Vergleiche machen zu können, definieren wir eine ein XML-Syntax ausgedrückte erweiterte Form für jeden Elementtyp und jeden Attributnamen in einem XML-Dokument.

[Definition: Ein **erweiterter Elementtyp** wird als ein leeres XML-Element des Typs `ExpEType` ausgedrückt. Es hat ein erforderliches `type`-Attribut, das den LocalPart des Typs angibt, und ein optionales `ns`-Attribut, das, sofern das Element qualifiziert ist, seinen Namensraumnamen angibt.]

[Definition: Ein **erweiterter Attributname** wird als ein leeres XML-Element des Typs `ExpAName` ausgedrückt. Es hat ein erforderliches `name`-Attribut, das den Namen angibt. Sofern das Attribut global ist, hat es ein erforderliches `ns`-Attribut, das den Namensraumnamen angibt; ansonsten hat es ein erforderliches Attribut `elttype`, das den Typ des angehängten Elements angibt und ein optionales Attribut `elns`, das den Namensraumnamen des angehängten Elements angibt, sofern bekannt.]

Geringfügige Abweichungen der obigen Beispiele werden die Arbeitsweise der erweiterten Elementtypen und Attributnamen veranschaulichen. Den beiden folgenden Ausschnitten folgt jeweils eine Tabelle, welche die Erweiterung der Namen zeigt:

```
<!-- 1 --> <section xmlns='urn:com:books-r-us'>
<!-- 2 -->   <title>Book-Signing Event</title>
```

```
<!-- 3 -->   <signing>
<!-- 4 -->     <author title="Mr" name="Vikram Seth" />
<!-- 5 -->     <book title="A Suitable Boy"
price="$22.95" />
            </signing>
          </section>
```

Die Namen würden wie folgt erweitert:

Line	Name	Expanded
1	section	<ExpEType type="section" ns="urn:com:books-r-us" />
2	title	<ExpEType type="title" ns="urn:com:books-r-us" />
3	signing	<ExpEType type="signing" ns="urn:com:books-r-us" />
4	author	<ExpEType type="author" ns="urn:com:books-r-us" />
4	title	<ExpAName name='title' eltype="author" elns="urn:com:books-r-us" />
4	name	<ExpAName name='name' eltype="author" elns="urn:com:books-r-us" />
5	book	<ExpEType type="book" ns="urn:com:books-r-us" />
5	title	<ExpAName name='title' eltype="book" elns="urn:com:books-r-us" />
5	price	<ExpAName name='price' eltype="book" elns="urn:com:books-r-us" />

```
<!-- 1 --> <RESERVATION xmlns:HTML="http://www.w3.org/
TR/REC-html40">
<!-- 2 --> <NAME HTML:CLASS="largeSansSerif">Layman,
A</NAME>
<!-- 3 --> <SEAT CLASS="Y" HTML:
CLASS="largeMonotype">33B</SEAT>
<!-- 4 --> <HTML:A HREF='/cgi-bin/ResStatus'>Check
Status</HTML:A>
<!-- 5 --> <DEPARTURE>1997-05-24T07:55:00+1</
DEPARTURE></RESERVATION>
```

1	RESERVATION	<ExpEType type="RESERVATION" />
2	NAME	<ExpEType type="NAME" />
2	HTML:CLASS	<ExpAName name="CLASS" ns="http://www.w3.org/TR/REC-html40" />
3	SEAT	<ExpEType type="SEAT" />
3	CLASS	<ExpAName name="CLASS" eltype="SEAT" />
3	HTML:CLASS	<ExpAName name="CLASS" ns="http://www.w3.org/TR/REC-html40" />
4	HTML:A	<ExpEType type="A" ns="http://www.w3.org/TR/REC-html40" />
4	HREF	<ExpAName name="HREF" eltype="A" elns="http://www.w3.org/TR/REC-html40" />
5	DEPARTURE	<ExpEType type="DEPARTURE" />

A.4 Einzigartige erweiterte Attributnamen

Die Zwänge, ausgedrückt in **5.3 Einzigartigkeit von Attributen** oben, könnten einfach so implementiert werden, dass kein Element zwei Attribute haben darf, deren erweiterte Namen äquivalent sind, z.B. die gleichen Attributwert-Paare haben.

B Anerkennungen (nicht normativ)

Diese Arbeit spiegelt die Beiträge vieler Personen wieder, insbesondere der Mitglieder der World Wide Web Consortium XML Working Group und der Special Interest Group sowie der Mitarbeiter der W3C Metadata Activity. Die Mitwirkung von Charles Frankston von Microsoft war besonders wertvoll.

C Verweise

RFC2141

IETF (Internet Engineering Task Force) *RFC 2141: URN Syntax*, Editor R. Moats. Mai 1997.

RFC2396

IETF (Internet Engineering Task Force) *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*, Editoren T. Berners-Lee, R. Fielding, L. Masinter. August 1998.

XML

Extensible Markup Language (XML) 1.0, Editoren Tim Bray, Jean Paoli und C. M. Sperberg-McQueen. 10. Februar 1998. Verfügbar unter: http://www.edition-w3c.de/TR/REC-xml.

Stichwortverzeichnis

Miloslav Nic [nicmila@systinet.com]

Namespace Tutorial

Introduction

XML Namespaces are described in **Namespaces in XML**.

You can start from :

- **Example 1**
- **Contents**

See also

- **XML tutorial**

Related keywords: tutorial, Namespace, XML

Partner site books:

- **HTML Utopia: Designing Without Tables Using CSS** by Dan Shafer,
- **Build Your Own Database Driven Website Using PHP & MySQL** by Kevin Yank,
- **The Web Design Business Kit** by Brendon Sinclair

Related sites:

- **VisualBuilder.com** - Java, JSP, ASP, XML. scripting languages, ...
- **TopXML** - XML Tutorials, MS.NET XML Tutorials
- **Tek-Tips Forum** - Software and hardware forums for computer professionals
- **Resource Index** - PHP, CGI, Perl
- **Programmers Heaven** - Assembler, Basic, C/C++, C#, Delphi & Kylix, Java, ...
- **ASP Alliance** - ASP, ASP.NET, ADO.NET, C#, JScript, VBScript, VB.Net
- **DevelopersDex** - ASP, C#, SQL, VB, XML
- **Developer Fusion** - Free VB, ASP, .NET and C++ tutorials and source code
- **DevGuru** - ADO, ASP, CSS2, HTML, Javascript, JetSQL, VBScript, WML, XML, ...
- **Code Project** - Free C++, C# and .NET articles, code snippets, discussions, news
- **Planet Source Code** - The largest public source code database on the Internet

▶ TOPICS
Business
Databases
Graphics
Metadata
Mobile
Programming
Schemas
Style
Web
Web Services

▶ ESSENTIALS
Annotated XML
What is XML?
What is XSLT?
What is XSL-FO?
What is XLink?
What is XML Schema?
What is XQuery?
What is RDF?
What is RSS?
What are Topic Maps?
What are Web Services?
What are XForms?
XSLT Recipe of the Day

Manage Your Account
Forgot Your Password?

▶ FIND
Search
Article Archive

▶ COLUMNS
<taglines/>
Dive into XML
Hacking the Library
Jon Udell
Perl and XML
Practical XQuery

XML Namespaces by Example

Print
Email article link

by Tim Bray
January 19, 1999

January 14th saw the arrival of a new W3C Recommendation, Namespaces in XML. "Recommendation" is the final step in the W3C process; the status means that the document is done, frozen, agreed-upon and official.

Namespaces are a simple and straightforward way to distinguish names used in XML documents, no matter where they come from. However, the concepts are a bit abstract, and this specification has been causing some mental indigestion among those who read it. The best way to understand namespaces, as with many other things on the Web, is by example.

So let's set up a scenario: suppose XML.com wanted to start publishing reviews of XML books. We'd want to mark the info up with XML, of course, but we'd also like to use HTML to help beautify the display. Here's a tiny sample of what we might do:

```
<h:html xmlns:xdc="http://www.xml.com/books"
        xmlns:h="http://www.w3.org/HTML/1998/html4">
 <h:head><h:title>Book Review</h:title></h:head>
 <h:body>
  <xdc:bookreview>
   <xdc:title>XML: A Primer</xdc:title>
   <h:table>
    <h:tr align="center">
     <h:td>Author</h:td><h:td>Price</h:td>
     <h:td>Pages</h:td><h:td>Date</h:td></h:tr>
    <h:tr align="left">
     <h:td><xdc:author>Simon St. Laurent</xdc:author></h:td>
     <h:td><xdc:price>31.98</xdc:price></h:td>
     <h:td><xdc:pages>352</xdc:pages></h:td>
     <h:td><xdc:date>1998/01</xdc:date></h:td>
    </h:tr>
   </h:table>
  </xdc:bookreview>
 </h:body>
</h:html>
```

In this example, the elements prefixed with **xdc** are associated with a namespace whose name is **http://www.xml.com/books**, while those prefixed with **h** are associated with a namespace whose name is **http://www.w3.org/HTML/1998/html4**.

The prefixes are linked to the full names using the attributes on the top element whose names begin. **xmlns:**. The prefixes don't mean anything at all - they are just shorthand placeholders for the full names. Those full names, you will have noticed, are URLs, i.e. Web addresses. We'll get back to why that is and what those are the addresses of a bit further on.

Why Namespaces?

Python and XML
Rich Salz
Sacré SVG
Standards Lowdown
Transforming XML
XML Q&A
XML-Deviant

GUIDES
XML Resources
Buyer's Guide
Events Calendar
Standards List
Submissions List

TOOLBOX
Syntax Checker

But first, an obvious question: why do we need these things? They are there to help computer software do its job. For example, suppose you're a programmer working for XML.com and you want to write a program to look up the books at Amazon.com and make sure the prices are correct. Such lookups are quite easy, once you know the author and the title. The problem, of course, is that this document has XML.com's book-review tags and HTML tags all mixed up together, and you need to be sure that you're finding the book titles, not the HTML page titles.

The way you do this is to write your software to process the contents of **<title>** tags, but only when they're in the **http://www.xml.com/books** namespace. This is safe, because programmers who are not working for XML.com are not likely to be using that namespace.

Attributes Too

Attributes, not just elements, can have namespaces. For example, let's use the HTML **STYLE** attribute to allow an HTML browser to display our book review:

```
<h:html xmlns:xdc="http://www.xml.com/books"
        xmlns:h="http://www.w3.org/HTML/1998/html4">
 <h:head><h:title>Book Review</h:title></h:head>
 <h:body>
  <xdc:bookreview>
   <xdc:title h:style="font-family: sans-serif;">
     XML: A Primer</xdc:title>
   <h:table>
    <h:tr align="center">
     <h:td>Author</h:td><h:td>Price</h:td>
     <h:td>Pages</h:td><h:td>Date</h:td></h:tr>
    <h:tr align="left">
     <h:td><xdc:author>Simon St. Laurent</xdc:author></h:td>
     <h:td><xdc:price>31.98</xdc:price></h:td>
     <h:td><xdc:pages>352</xdc:pages></h:td>
     <h:td><xdc:date>1998/01</xdc:date></h:td>
    </h:tr>
   </h:table>
  </xdc:bookreview>
 </h:body>
</h:html>
```

Beautification

That example above is, perhaps, kind of ugly, with all those prefixes and colons clutering up the tags. The Namespaces Recommendation allows you to declare a default namespace and leave out some prefixes, like this:

```
<html xmlns="http://www.w3.org/HTML/1998/html4"
      xmlns:xdc="http://www.xml.com/books">
 <head><title>Book Review</title></head>
 <:body>
  <xdc:bookreview>
   <xdc:title>XML: A Primer</xdc:title>
   <table>
    <tr align="center">
     <td>Author</td><td>Price</td>
     <td>Pages</td><td>Date</td></tr>
    <tr align="left">
     <td><xdc:author>Simon St. Laurent</xdc:author></td>
     <td><xdc:price>31.98</xdc:price></td>
     <td><xdc:pages>352</xdc:pages></td>
     <td><xdc:date>1998/01</xdc:date></td>
    </tr>
   </table>
  </xdc:bookreview>
 </body>
</html>
```

In this example, anything without a prefix is assumed to be in the **http://www.w3.org/HTML/1998/html4** namespace, which we're using as the namespace name for HTML (presumably, now that namespaces are official, the W3C will give HTML an official namespace name).

What Do Namespace Names Point At?

One of the confusing things about all this is that namespace names are URLs; it's easy to assume that since they're Web addresses, they must be the address of something. They're not; these are URLs, but the namespace draft doesn't care what (if anything) they point at. Think about the example of the XML.com programmer looking for book titles; that works fine without the namespace name pointing at anything.

The reason that the W3C decided to use URLs as namespace names is that they contain domain names (e.g. **`www.xml.com`**), which work globally across the Internet.

Is That All There Is?

That's more or less all there is to it. The only purpose of namespaces is to give programmers a helping hand, enabling them to process the tags and attributes they care about and ignore those that don't matter to them.

Quite a few people, after reading earlier drafts of the Namespace Recommendation, decided that namespaces were actually a facility for modular DTDs, or were trying to duplicate the function of SGML's "Architectural Forms". None of these theories are true. The only reason namespaces exist, once again, is to give elements and attributes programmer-friendly names that will be unique across the whole Internet.

Namespaces are a simple, straightforward, unglamorous piece of syntax. But they are crucial for the future of XML programming. Because this is important, we at XML.com will be soon be posting an Annotated Namespaces, in a style similar to our Annotated XML 1.0.

▶ TOPICS
Business
Databases
Graphics
Metadata
Mobile
Programming
Schemas
Style
Web
Web Services

▶ ESSENTIALS
Annotated XML
What is XML?
What is XSLT?
What is XSL-FO?
What is XLink?
What is XML Schema?
What is XQuery?
What is RDF?
What is RSS?
What are Topic Maps?

Print
Email article link

Namespaces in XML Adopted by W3C

by Mark Walter
January 19, 1999

The "Namespaces in XML" specification has been formally adopted by the W3C as a recommendation from its members and its director, Tim Berners-Lee, paving the way for vendors to create software that will more easily support the rich markup vocabularies made possible by XML.

What's the problem?
XML is a language for creating markup -- think of it as the way to write down new tag sets that your software will recognize and make use of. The whole point of XML is to enable users to be able to create unique tags that identify their information in more meaningful ways than simply applying the basic set of HTML tags to all documents.

While this gives users great flexibility, it poses problems for interchange and software integration. What happens when two documents make use of the same tag names in different contexts? For example, a <PART> tag in an illustrated parts catalog identifies something quite different than a part in a dramatic play. Within a single document, the term "title" may refer to the document itself, the name of a book, and the formal appellation associated with its author (e. g., "Dr."). The problem is not just for element names; it extends to attributes as well.

This potential collision over different uses of the same names poses problems for anyone writing XML- based software and applications. It?s difficult

What are Web Services?
What are XForms?
XSLT Recipe of the Day

Manage Your Account
Forgot Your Password?

FIND
Search
Article Archive

Add the latest news from XML.com to your web site
Click here!

COLUMNS
<taglines/>
Dive into XML
Hacking the Library
Jon Udell
Perl and XML
Practical XQuery
Python and XML
Rich Salz
Sacré SVG
Standards Lowdown
Transforming XML
XML Q&A
XML-Deviant

GUIDES
XML Resources
Buyer's Guide
Events Calendar
Standards List
Submissions List

TOOLBOX
Syntax Checker

to write a style sheet that displays book titles in italic but gives no special formatting to people?s titles if I don?t have a good way to distinguish the two uses of the title tag.

A partial solution

The XML namespaces spec addresses this issue by allowing tags to have a context. That context is the tag or attributes? XML namespace, which is simply a Web address. Because Web addresses are unique, they?re a handy way to establish unique contexts. For example, you could create a namespace called EDI, linked to a URL; declare that namespace at the beginning of your XML document; and then add "EDI:" as a prefix to any element name in the document. The use of the declared prefix provides a way for software to treat tags with EDI prefixes differently than tags with different, prefixes. You can also declare a default namespace at the start of your document; any tags without prefixes are assumed to be in the default namespace.

One thing the namespace does not define is what the tags or attributes are, or what they mean. That larger effort, which would enable DTD designers to reference public tag sets, is part of what the XML schema working group is tackling. Further details on how namespaces operate can be seen in our accompanying story, and in the specification itself.

Further information about upcoming developments of XML are available at the W3C Web site.

The implications

All along, it has been envisioned that XML will be applied to specific vertical industries. Medicine has patient records; legal publishing has court cases and commentary; aerospace has its massive engines and planes. Many have presumed that XML wouldn?t take off until vertical industries published their DTDs.

Public DTDs are useful, and ultimately we are likely to see more of them. In the meantime, though, some people writing their own XML applications (software acting on private DTDs) would like a way to at least resolve tag and attribute name conflicts. Knowing the meaning of the tags is required for true

interchange, but for the purposes of creating style sheets for display, it may be enough to simply have a unique ID and make an educated guess about the meaning.

"Think of an invoice," suggests Dan Connolly, W3C XML Activity Lead. "Most of an invoice like the addresses and quantities and amounts are in regular commercial language. But maybe the descriptions of exactly what parts have been ordered would only be understood by experts manufacturing or using the parts. Still, many people can understand the invoice without having to understand what the part description means. XML namespaces allows a digitally coded document like this invoice to be processed -- without everyone who uses invoices having to agree on a vocabulary for turbojet engine side intake manifold monitor valve mounting nuts, or whatever."

With the namespace spec, what the W3C has said is that what is required is a standard way to resolve conflicts in tag names, so that programmers around the globe can work from a common understanding of how tags will be identified.

The XML namespace spec does provide this universal mechanism that everyone can use to create tag and attribute names that are unique in the context of specific documents, just as file names are unique in the context of the directory of a computer?s hard disk.

Though namespaces operate "under the hood" and are really aimed at programmers, they will enable a variety of functions -- from interesting typographic treatment of elements to sophisticated processing of orders and invoices -- that all manner of Web users, even novices, will appreciate.

Pages: **1**, 2 Next Page

Namespaces in XML

Best Practices, Risky Business

Simon St.Laurent

XMLDevCon2000, November 2000

Start >

Table of Contents

XML Namespaces FAQ

Maintained by Ronald Bourret

(Last updated January, 2004)

Table of Contents

PART I: OVERVIEW

1) SECTION 1: EXECUTIVE SUMMARY

1.1) Can you give me an executive summary of what XML namespaces are?

Sure. Here it is in five easy bullet points.

- The XML namespaces recommendation defines a way to distinguish between duplicate element type and attribute names. Such duplication might occur, for example, in an XSLT stylesheet or in a document that contains element types and attributes from two different DTDs.
- An XML namespace is a collection of element type and attribute names. The namespace is identified by a unique name, which is a URI. Thus, any element type or attribute name in an XML namespace can be uniquely identified by a two-part name: the name of its XML namespace and its local name. **This two-part naming system is the only thing defined by the XML namespaces recommendation.**

- XML namespaces are declared with an xmlns attribute, which can associate a prefix with the namespace. The declaration is in scope for the element containing the attribute and all its descendants. For example:

```
  <!-- Declares two XML namespaces. Their scope is the A and B
elements. -->
  <A xmlns:foo="http://www.foo.org/" xmlns="http://www.bar.
org/">
     <B>abcd</B>
  </A>
```

- If an XML namespace declaration contains a prefix, you refer to element type and attribute names in that namespace with the prefix. For example:

```
  <!-- A and B are in the http://www.foo.org/ namespace, which
is associated with the foo prefix. -->
  <foo:A xmlns:foo="http://www.foo.org/">
     <foo:B>abcd</foo:B>
  </foo:A>
```

- If an XML namespace declaration does not contain a prefix, the namespace is the default XML namespace and you refer to element type names in that namespace without a prefix. For example:

```
  <!-- This is equivalent to the previous example but uses a
default namespace instead of the foo prefix. -->
  <A xmlns="http://www.foo.org/">
     <B>abcd<B>
  </A>
```

1.2) Can you give me an executive summary of what XML namespaces are not?

They aren't a cure of cancer, they aren't a way to win the lottery, and they aren't a direct cause of world peace. They also aren't very difficult to understand or use. Two things that XML namespaces are not have caused a lot of confusion, so we'll mention them here:

- XML namespaces are not a technology for joining XML documents that use different DTDs. Although they might be used in such a technology, they don't provide it themselves.
- The URIs used as XML namespace names are not guaranteed to point to schemas, information about the namespace, or anything else -- they're just identifiers. URIs were used simply because they're a well-known system for creating unique identifiers. Don't even think about trying to resolve these URIs. (For details, see question 12.4.)

2) SECTION 2: TRADITIONAL NAMESPACES

2.1) What is a traditional namespace?

Before discussing XML namespaces, it is useful to discuss namespaces in general. In this FAQ, we will refer to such namespaces as *traditional namespaces*. We will refer to XML namespaces as *XML namespaces*. The word *namespace* can refer to either a traditional namespace or an XML namespace, depending on context, but will generally refer to an XML namespace.

A traditional namespace is a set of zero or more names, each of which must be unique within the namespace and constructed according to the rules (if any) of the namespace. For example, the names of element types in an XML document inhabit a traditional namespace, as do the names of tables in a relational database and the names of class variables in a Java class. Traditional namespaces also occur outside the field of computer science -- for example, the names of people could be thought to inhabit a traditional namespace, as could the names of species.

2.2) What is the relationship between different traditional namespaces?

They are disjoint -- that is, they are not related. Because of this, a name in one traditional namespace does not collide with the same name in a different traditional namespace. This property is useful to applications that have multiple sets of names. By assigning each set of names to a different traditional namespace, they can allow the same name to occur in each set of names without fear of collision.

For example, in the following XML document, there is no conflict between the three different uses of the name Value.

```
<AuctionItem>
   <Title Value="486Laptop"/>
   <Category Value="Computers"/>
   <Value>$100</Value>
</AuctionItem>
```

This is because an XML document has one traditional namespace for element type names and, for each element type, one traditional namespace for the names of the attributes that apply to that element type. Thus, the two Value attribute names don't conflict because each is assigned to a different traditional namespace -- the first to the attribute namespace for the Title element type and the second to the attribute namespace for the Category element type. Furthermore, neither of the Value attribute names conflicts with the Value element type name because element type names are kept in a traditional namespace that is separate from the attribute namespaces.

Other examples of applications that use multiple traditional namespaces include:

- Java classes. In a Java class, there is one traditional namespace for the names of class variables, one traditional namespace for the names of methods, and, for each method, one traditional namespace for the names of variables local to that method.
- Relational databases. In a relational database, there is one traditional namespace for the names of tables and, for each table, one traditional namespace for the names of columns in that table.

2.3) What are traditional namespaces used for?

Perhaps the most common (computer science) use of traditional namespaces is to provide a container for a set of identifiers. For example, traditional namespaces are used to contain the names (identifiers) of element types in an XML document, the names of class variables in a Java class, and the names of tables in a relational database. Traditional namespaces are useful in this regard because of their requirement that each name in the namespace be unique. Thus, when a new name (identifier) is added to the namespace, the uniqueness of the identifier can be verified by checking that the name does not already exist in the namespace.

(Note that just because a set of objects draws its names from some traditional namespace does not mean that those names uniquely identify the objects. For example, two different people can share the same name, as can two different element nodes in a DOM tree, which use element type names as their names. For the names in a traditional namespace to uniquely identify the objects in a set, the objects in the set must draw their names only from that namespace and no name can be applied to more than one object. In practice, the names from a single traditional namespace are only used to identify the objects in a single set; otherwise, additional information must be stored stating which names apply to which set.)

3) SECTION 3: XML NAMESPACES

3.1) What is the purpose of XML namespaces?

XML namespaces are designed to provide universally unique names for elements and attributes. This allows people to do a number of things, such as:

- Combine fragments from different documents without any naming conflicts. (See example below.)
- Write reusable code modules that can be invoked for specific elements and attributes. Universally unique names guarantee that such modules are invoked only for the correct elements and attributes.
- Define elements and attributes that can be reused in other schemas or instance documents without fear of name collisions. For example, you might use XHTML elements in a parts catalog to provide part descriptions. Or you might use the nil attribute defined in XML Schemas to indicate a missing value.

As an example of how XML namespaces are used to resolve naming conflicts in XML documents that contain element types and attributes from multiple XML languages, consider the following two XML documents:

```
<?xml version="1.0" ?>
<Address>
   <Street>Wilhelminenstr. 7</Street>
   <City>Darmstadt</City>
```

```
    <State>Hessen</State>
    <Country>Germany</Country>
    <PostalCode>D-64285</PostalCode>
  </Address>
```

and:

```
  <?xml version="1.0" ?>
  <Server>
    <Name>OurWebServer</Name>
    <Address>123.45.67.8</Address>
  </Server>
```

Each document uses a different XML language and each language defines an Address element type. Each of these Address element types is different -- that is, each has a different content model, a different meaning, and is interpreted by an application in a different way. This is not a problem as long as these element types exist only in separate documents. But what if they are combined in the same document, such as a list of departments, their addresses, and their Web servers? How does an application know which Address element type it is processing?

One solution is to simply rename one of the Address element types -- for example, we could rename the second element type IPAddress. However, this is not a useful long term solution. One of the hopes of XML is that people will standardize XML languages for various subject areas and write modular code to process those languages. By reusing existing languages and code, people can quickly define new languages and write applications that process them. If we rename the second Address element type to IPAddress, we will break any code that expects the old name.

A better answer is to assign each language (including its Address element type) to a different namespace. This allows us to continue using the Address name in each language, but to distinguish between the two different element types. The mechanism by which we do this is XML namespaces.

(Note that by assigning each Address name to an XML namespace, we actually change the name to a two-part name consisting of the name of the XML namespace plus the name Address. This means that any code that recognizes just the name Address will need to be changed to recognize the new two-part name. However, this only needs to be done once, as the two-part name is universally unique. For more information about the uniqueness of universal name, see question 10.13; for more information about processing universal names, see question 9.1.)

3.2) What is an XML namespace?

An XML namespace is a collection of element type and attribute names. The collection itself is unimportant -- in fact, a reasonable argument can be made that XML namespaces don't actually exist as physical or conceptual entities (see myth #1 in "Namespace Myths Exploded"). What is important is the name of the XML namespace, which is a URI. This allows XML namespaces to provide a two-part naming system for element types and attributes. The first part of the name is the URI used to

identify the XML namespace -- the *namespace name*. The second part is the element type or attribute name itself -- the *local part*, also known as the *local name*. Together, they form the *universal name*.

This two-part naming system is the only thing defined by the XML namespaces recommendation.

By using multiple XML namespaces, multiple element types with the same local name can inhabit the same XML document. For example, the following document uses XML namespaces to distinguish between two different element types named Address.

```
  <Department>
     <Name>DVS1</Name>
     <addr:Address xmlns:addr="http://www.tu-darmstadt.de/ito/
addresses">
        <addr:Street>Wilhelminenstr. 7</addr:Street>
        <addr:City>Darmstadt</addr:City>
        <addr:State>Hessen</addr:State>
        <addr:Country>Germany</addr:Country>
        <addr:PostalCode>D-64285</addr:PostalCode>
     </addr:Address>
     <serv:Server xmlns:serv="http://www.tu-darmstadt.de/ito/
servers">
        <serv:Name>OurWebServer</serv:Name>
        <serv:Address>123.45.67.8</serv:Address>
     </serv:Server>
  </Department>
```

The first Address element type name belongs to the http://www.tu-darmstadt.de/ito/addresses XML namespace. It has a universal (two-part) name of "http://www.tu-darmstadt.de/ito/addresses" plus "Address". (Following the convention proposed by James Clark in his paper XML Namespaces (see question 14.1), we write this as {http://www.tu-darmstadt.de/ito/addresses}Address.) The second Address element type name belongs to the http://www.tu-darmstadt.de/ito/servers XML namespace and has a universal name of {http://www.tu-darmstadt.de/ito/servers}Address. Thus, each universal name is unique, meeting the requirement that each element type in an XML document have a unique name.

3.3) Does the XML namespaces recommendation define anything except a two-part naming system for element types and attributes?

No.

This is a very important point and a source of much confusion, so we will repeat it:

THE XML NAMESPACES RECOMMENDATION DOES NOT DEFINE ANYTHING EXCEPT A TWO-PART NAMING SYSTEM

FOR ELEMENT TYPES AND ATTRIBUTES.

In particular, they do not provide or define any of the following:

- A way to merge two documents that use different DTDs. (See question 8.5.)
- A way to associate XML namespaces and schema information. (See question 12.2 and myth #8 in "Namespace Myths Exploded".)
- A way to validate documents that use XML namespaces. (See question 7.6.)
- A way to associate element type or attribute declarations in a DTD with an XML namespace. (See question 7.2.)

3.4) What do XML namespaces actually contain?

XML namespaces are collections of names, nothing more. That is, they contain the *names* of element types and attributes, not the elements or attributes themselves. For example, consider the following document.

```
<foo:A xmlns:foo="http://www.foo.org/">
   <B foo:C="foo" D="bar"/>
</foo:A>
```

The element type *name* A and the attribute *name* C are in the http://www.foo.org/ namespace because they are mapped there by the foo prefix. The element type *name* B and the attribute name D are not in any XML namespace because no prefix maps them there. On the other hand, the *elements* A and B and the attributes C and D are not in any XML namespace, even though they are physically within the scope of the http://www.foo.org/ namespace declaration. This is because XML namespaces contain *names*, not *elements* or *attributes*.

XML namespaces also do not contain the definitions of the element types or attributes. This is an important difference, as many people are tempted to think of an XML namespace as a schema, which it is not. (For more information, see myth #8 in "Namespace Myths Exploded".)

(For information about the structure of an XML namespace, see myth #6 in "Namespace Myths Exploded".)

3.5) Are the names of all element types and attributes in some XML namespace?

No.

If an element type or attribute name is not specifically declared to be in an XML namespace -- that is, it is unprefixed and (in the case of element type names) there is no default XML namespace -- then that name is not in any XML namespace. If you want, you can think of it as having a null URI as its name, although no "null" XML namespace actually exists. For example, in the following, the element type name B and the attribute names C and E are not in any XML namespace:

```
<foo:A xmlns:foo="http://www.foo.org/">
   <B C="bar"/>
   <foo:D E="bar"/>
</foo:A>
```

For more information about unprefixed attribute names and XML namespaces, see myth #4 in "Namespace Myths Exploded".

3.6) Do XML namespaces apply to entity names, notation names, or processing instruction targets?

No.

XML namespaces apply only to element type and attribute names. Furthermore, in an XML document that conforms to the XML namespaces recommendation, entity names, notation names, and processing instruction targets must not contain colons.

3.7) Who can create an XML namespace?

Anybody can create an XML namespace -- all you need to do is assign a URI as its name and decide what element type and attribute names are in it. The URI must be under your control and should not be being used to identify a different XML namespace, such as by a coworker.

(In practice, most people that create XML namespaces also describe the element types and attributes whose names are in it -- their content models and types, their semantics, and so on. However, this is not part of the process of creating an XML namespace, nor does the XML namespace include or provide a way to discover such information.)

3.8) Do I need to use XML namespaces?

Maybe, maybe not.

If you don't have any naming conflicts in the XML documents you are using today, as is often the case with documents used inside a single organization, then you probably don't need to use XML namespaces. However, if you do have conflicts today, or if you expect conflicts in the future due to distributing your documents outside your organization or bringing outside documents into your organization, then you should probably use XML namespaces.

Regardless of whether you use XML namespaces in your own documents, it is likely that you will use them in conjunction with some other XML technology, such as XSL, XHTML, or XML Schemas. For example, the following XSLT (XSL Transformations) stylesheet uses XML namespaces to distinguish between element types defined in XSLT and those defined elsewhere:

```
  <xsl:stylesheet version="1.0"
                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:template match="Address">
     <!-- The Addresses element type is not part of the XSLT
namespace. -->
     <Addresses>
        <xsl:apply-templates/>
     </Addresses>
  </xsl:stylesheet>
```

3.9) What is the relationship between XML namespaces and the XML 1.0 recommendation?

Although the XML 1.0 recommendation anticipated the need for XML namespaces by noting that element type and attribute names should not include colons, it did not actually support XML namespaces. Thus, XML namespaces are layered on top of XML 1.0. In particular, any XML document that uses XML namespaces is a legal XML 1.0 document and can be interpreted as such in the absence of XML namespaces. For example, consider the following document:

```
  <foo:A xmlns:foo="http://www.foo.org/">
     <foo:B foo:C="bar"/>
  </foo:A>
```

If this document is processed by a namespace-unaware processor, that processor will see two elements whose names are foo:A and foo:B. The foo:A element has an attribute named xmlns:foo and the foo:B element has an attribute named foo:C. On the other hand, a namespace-aware processor will see two elements with universal names {http://www.foo.org}A and {http://www.foo.org}B. The {http://www.foo.org}A does not have any attributes; instead, it has a namespace declaration that maps the foo prefix to the URI http://www.foo.org. The {http://www.foo.org}B element has an attribute named {http://www.foo.org}C.

Needless to say, this has led to a certain amount of confusion. One area of confusion is the relationship between XML namespaces and validating XML documents against DTDs. This occurs because the XML namespaces recommendation did not describe how to use XML namespaces with DTDs. (For more information, see section 7.) Fortunately, a similar situation does not occur with XML schema languages, as all of these support XML namespaces.

The other main area of confusion is in recommendations and specifications such as DOM and SAX whose first version predates the XML namespaces recommendation. Although these have since been updated to include XML namespace support, the solutions have not always been pretty due to backwards compatibility requirements. All recommendations in the XML family now support XML namespaces.

3.10) What is the difference between versions 1.0 and 1.1 of the XML namspaces recommendation?

There are only two differences between XML namespaces 1.0 and XML namespaces 1.1:

- Version 1.1 adds a way to undeclare prefixes. For more information, see question 4.7.
- Version 1.1 uses IRIs (Internationalized Resource Identifiers) instead of URIs. Basically, URIs are restricted to a subset of ASCII characters, while IRIs allow much broader use of Unicode characters. For complete details, see section 9 of Namespaces in XML 1.1.

NOTE: As of this writing (February, 2003), Namespaces in XML 1.1 is still a candidate recommendation and not widely used.

PART II: DECLARING AND USING XML NAMESPACES

4) SECTION 4: DECLARING XML NAMESPACES IN AN XML DOCUMENT

4.1) How do I declare an XML namespace in an XML document?

To declare an XML namespace, you use an attribute whose name has the form:

```
xmlns:prefix
      --OR--
xmlns
```

These attributes are often called *xmlns attributes* and their value is the name of the XML namespace being declared; this is a URI. The first form of the attribute (xmlns:*prefix*) declares a prefix to be associated with the XML namespace. The second form (xmlns) declares that the specified namespace is the default XML namespace.

For example, the following declares two XML namespaces, named http://www.tu-darmstadt.de/ito/addresses and http://www.tu-darmstadt.de/ito/servers. The first declaration associates the addr prefix with the http://www.tu-darmstadt.de/ito/addresses namespace and the second declaration states that the http://www.tu-darmstadt.de/ito/servers namespace is the default XML namespace.

```
<Department
     xmlns:addr="http://www.tu-darmstadt.de/ito/addresses"
     xmlns="http://www.tu-darmstadt.de/ito/servers">
```

NOTE: Technically, xmlns attributes are not attributes at all -- they are XML namespace declarations that just happen to look like attributes. Unfortunately, they are not treated consistently by the various XML recommendations, which means that you must be careful when writing an XML application.

For example, in the XML Information Set (http://www.w3.org/TR/xml-infoset), xmlns "attributes" do not appear as attribute information items. Instead, they appear as namespace declaration information items. On the other hand, both DOM level 2 and SAX 2.0 treat namespace attributes somewhat

ambiguously. In SAX 2.0, an application can instruct the parser to return xmlns "attributes" along with other attributes, or omit them from the list of attributes. Similarly, while DOM level 2 sets namespace information based on xmlns "attributes", it also forces applications to manually add namespace declarations using the same mechanism the application would use to set any other attributes.

4.2) Where can I declare an XML namespace?

You can declare an XML namespace on any element in an XML document. The namespace is in scope for that element and all its descendants unless it is overridden (see question 4.5 and question 4.6) or undeclared (see question 4.8). For more information about scope, see section 6.

4.3) Can I use an attribute default in a DTD to declare an XML namespace?

Yes.

For example, the following uses the FIXED attribute xmlns:foo on the A element type to associate the foo prefix with the http://www.foo.org/ namespace. The effect of this is that both A and B are in the http://www.foo.org/ namespace.

```
<?xml version="1.0" ?>
<!DOCTYPE foo:A [
   <!ELEMENT foo:A (foo:B)>
   <!ATTLIST foo:A
             xmlns:foo CDATA #FIXED "http://www.foo.org/">
   <!ELEMENT foo:B (#PCDATA)>
]>
<!-- foo prefix declared through default attribute. -->
<foo:A>
   <foo:B>abc</foo:B>
</foo:A>
```

IMPORTANT: You should be very careful about placing XML namespace declarations in external entities (external DTDs), as non-validating parsers are not required to read these. For example, suppose the preceding DTD was placed in an external entity (foo.dtd) and that the document was processed by a non-validating parser that did not read foo.dtd. This would result in a namespace error because the foo prefix was never declared:

```
<?xml version="1.0" ?>
<!-- foo.dtd might not be read by non-validating parsers. -->
<!DOCTYPE foo:A SYSTEM "foo.dtd">
<!-- foo prefix not declared unless foo.dtd is read. -->
<foo:A>
   <foo:B>abc</foo:B>
</foo:A>
```

4.4) Do the default values of xmlns attributes declared in the DTD apply to the DTD?

No.

Declaring a default value of an xmlns attribute in the DTD does not declare an XML namespace for the DTD. (In fact, no XML namespace declarations apply to DTDs.) Instead, these defaults (declarations) take effect only when the attribute is instantiated on an element. For example:

```
<?xml version="1.0" ?>
<!DOCTYPE foo:A [
   <!ELEMENT foo:A (foo:B)>
   <!ATTLIST foo:A
             xmlns:foo CDATA #FIXED "http://www.foo.org/">
   <!ELEMENT foo:B (#PCDATA)>
]>
<foo:A>  <========== Namespace declaration takes effect here.
   <foo:B>abc</foo:B>
</foo:A>  <========= Namespace declaration ends here.
```

For more information, see question 7.2. (Note that an earlier version of MSXML (the parser used by Internet Explorer) did use fixed xmlns attribute declarations as XML namespace declarations, but that this was removed in MSXML 4. For more information, see question 8.6.)

4.5) How do I override an XML namespace declaration that uses a prefix?

To override the prefix used in an XML namespace declaration, you simply declare another XML namespace with the same prefix. For example, in the following, the foo prefix is associated with the http://www.foo.org/ namespace on the A and B elements and the http://www.bar.org/ namespace on the C and D elements. That is, the names A and B are in the http://www.foo.org/ namespace and the names C and D are in the http://www.bar.org/ namespace.

```
<foo:A xmlns:foo="http://www.foo.org/">
   <foo:B>
      <foo:C xmlns:foo="http://www.bar.org/">
         <foo:D>abcd</foo:D>
      </foo:C>
   </foo:B>
</foo:A>
```

In general, this leads to documents that are confusing to read and should be avoided.

4.6) How do I override a default XML namespace declaration?

To override the current default XML namespace, you simply declare another XML namespace as the

default. For example, in the following, the default XML namespace is the http://www.foo.org/ namespace on the A and B elements and the http://www.bar.org/ namespace on the C and D elements. That is, the names A and B are in the http://www.foo.org/ namespace and the names C and D are in the http://www.bar.org/ namespace.

```
<A xmlns="http://www.foo.org/">
   <B>
      <C xmlns="http://www.bar.org/">
         <D>abcd</D>
      </C>
   </B>
</A>
```

Using multiple default XML namespaces can lead to documents that are confusing to read and should be done carefully. For more information, see question 5.4.

4.7) How do I undeclare an XML namespace prefix?

In version 1.0 of the XML namespaces recommendation, you cannot "undeclare" an XML namespace prefix. It remains in scope until the end of the element on which it was declared unless it is overridden. Furthermore, trying to undeclare a prefix by redeclaring it with an empty (zero-length) name (URI) results in a namespace error. For example:

```
<foo:A xmlns:foo="http://www.foo.org/">
   <foo:B>
      <foo:C xmlns:foo="">   <==== This is an error in v1.0,
legal in v1.1.
         <foo:D>abcd</foo:D>
      </foo:C>
   </foo:B>
</foo:A>
```

In version 1.1 of the XML namespaces recommendation [currently a candidate recommendation -- February, 2003], you can undeclare an XML namespace prefix by redeclaring it with an empty name. For example, in the above document, the XML namespace declaration xmlns:foo="" is legal and removes the mapping from the foo prefix to the http://www.foo.org URI. Because of this, the use of the foo prefix in the foo:D element results in a namespace error.

4.8) How do I undeclare the default XML namespace?

To "undeclare" the default XML namespace, you declare a default XML namespace with an empty (zero-length) name (URI). Within the scope of this declaration, unprefixed element type names do not belong to any XML namespace. For example, in the following, the default XML namespace is the http://www.foo.org/ for the A and B elements and there is no default XML namespace for the C and D elements. That is, the names A and B are in the http://www.foo.org/ namespace and the names C

and D are not in any XML namespace.

```
<A xmlns="http://www.foo.org/">
   <B>
      <C xmlns="">
         <D>abcd</D>
      </C>
   </B>
</A>
```

4.9) Why are special attributes used to declare XML namespaces?

I don't know the answer to this question, but the likely reason is that the hope that they would simplify the process of moving fragments from one document to another document. An early draft of the XML namespaces recommendation proposed using processing instructions to declare XML namespaces. While these were simple to read and process, they weren't easy to move to other documents. Attributes, on the other hand, are intimately attached to the elements being moved.

Unfortunately, this hasn't worked as well as was hoped. For example, consider the following XML document:

```
<foo:A xmlns:foo="http://www.foo.org/">
   <foo:B>
      <foo:C>bar</foo:C>
   </foo:B>
</foo:A>
```

Simply using a text editor to cut the fragment headed by the <B> element from one document and paste it into another document results in the loss of namespace information because the namespace declaration is not part of the fragment -- it is on the parent element (<A>) -- and isn't moved.

Even when this is done programmatically, the situation isn't necessarily any better. For example, suppose an application uses DOM level 2 to "cut" the fragment from the above document and "paste" it into a different document. Although the namespace information is transferred (it is carried by each node), the namespace declaration (xmlns attribute) is not, again because it is not part of the fragment. Thus, the application must manually add the declaration before serializing the document or the new document will be invalid.

4.10) How do different XML technologies treat XML namespace declarations?

This depends on the technology -- some treat them as attributes and some treat them as namespace declarations. For example, SAX1 treats them as attributes and SAX2 can treat them as attributes or namespace declarations, depending on how the parser is configured. (For more information, see question 9.3 and the SAX2 documentation.) DOM levels 1 and 2 treat them as attributes, but DOM level 2 also interprets them as namespace declarations. (For more information, see question 9.5.)

XPath, XSLT, and XML Schemas treat them as namespaces declarations.

The reason that different technologies treat these differently is that many of these technologies predate XML namespaces. Thus, newer versions of them need to worry both about XML namespaces and backwards compatibility issues.

5) SECTION 5: USING XML NAMESPACES IN AN XML DOCUMENT

5.1) How do I use prefixes to refer to element type and attribute names in an XML namespace?

Make sure you have declared the prefix (see question 4.1) and that it is still in scope (see section 6). All you need to do then is prefix the local name of an element type or attribute with the prefix and a colon. The result is a *qualified name* (see question 10.1), which the application parses to determine what XML namespace the local name belongs to.

For example, suppose you have associated the serv prefix with the http://www.tu-darmstadt.de/ito/servers namespace and that the declaration is still in scope. In the following, serv:Address refers to the Address name in the http://www.tu-darmstadt.de/ito/servers namespace. (Note that the prefix is used on both the start and end tags.)

```
  <!-- serv refers to the http://www.tu-darmstadt.de/ito/servers
namespace. -->
  <serv:Address>123.45.67.8</serv:Address>
```

Now suppose you have associated the xslt prefix with the http://www.w3.org/1999/XSL/Transform namespace. In the following, xslt:version refers to the version name in the http://www.w3.org/1999/XSL/Transform namespace:

```
  <!-- xslt refers to the http://www.w3.org/1999/XSL/Transform
namespace. -->
  <html xslt:version="1.0">
```

5.2) How do I use the default XML namespace to refer to element type names in an XML namespace?

Make sure you have declared the default XML namespace (see question 4.1) and that that declaration is still in scope (see section 6). All you need to do then is use the local name of an element type. Even though it is not prefixed, the result is still a *qualified name* (see question 10.1), which the application parses to determine what XML namespace it belongs to.

For example, suppose you declared the http://www.tu-darmstadt.de/ito/addresses namespace as the default XML namespace and that the declaration is still in scope. In the following, Address refers to the Address name in the http://www.tu-darmstadt.de/ito/addresses namespace.

```
  <!-- http://www.tu-darmstadt.de/ito/addresses is the default XML
namespace. -->
  <Address>123.45.67.8</Address>
```

For information about how to use the default XML namespace with attribute names, see question 5.3.

5.3) How do I use the default XML namespace to refer to attribute names in an XML namespace?

You can't.

The default XML namespace only applies to element type names, so you can refer to attribute names that are in an XML namespace only with a prefix. For example, suppose that you declared the http://www.tu-darmstadt.de/ito/addresses namespace as the default XML namespace. In the following, the type attribute name does not refer to that namespace, although the Address element type name does. That is, the Address element type name is in the http://www.tu-darmstadt.de/ito/addresses namespace, but the type attribute name is not in any XML namespace.

```
  <!-- http://www.tu-darmstadt.de/ito/addresses is the default XML
namespace. -->
  <Address type="home">
```

To understand why this is true, remember that the purpose of XML namespaces is to uniquely identify element and attribute names. Unprefixed attribute names can be uniquely identified based on the element type to which they belong, so there is no need identify them further by including them in an XML namespace. In fact, the only reason for allowing attribute names to be prefixed is so that attributes defined in one XML language can be used in another XML language.

For information about how to use the default XML namespace with element type names, see question 5.2. For more information about unprefixed attribute names and XML namespaces, see myth #4 in "Namespace Myths Exploded".

5.4) When should I use the default XML namespace instead of prefixes?

This is purely a matter of choice, although your choice may affect the readability of the document. When elements whose names all belong to a single XML namespace are grouped together, using a default XML namespace might make the document more readable. For example:

```
  <!-- A, B, C, and G are in the http://www.foo.org/ namespace. -->
  <A xmlns="http://www.foo.org/">
     <B>abcd</B>
     <C>efgh</C>
     <!-- D, E, and F are in the http://www.bar.org/ namespace. -->
```

```
    <D xmlns="http://www.bar.org/">
       <E>1234</E>
       <F>5678</F>
    </D>
    <!-- Remember! G is in the http://www.foo.org/ namespace. -->
    <G>ijkl</G>
</A>
```

When elements whose names are in multiple XML namespaces are interspersed, default XML namespaces definitely make a document more difficult to read and prefixes should be used instead. For example:

```
<A xmlns="http://www.foo.org/">
   <B xmlns="http://www.bar.org/">abcd</B>
   <C xmlns="http://www.foo.org/">efgh</C>
   <D xmlns="http://www.bar.org/">
      <E xmlns="http://www.foo.org/">1234</E>
      <F xmlns="http://www.bar.org/">5678</F>
   </D>
   <G xmlns="http://www.foo.org/">ijkl</G>
</A>
```

In some cases, default namespaces can be processed faster than namespace prefixes, but the difference is certain to be negligible in comparison to total processing time.

6) SECTION 6: SCOPE OF XML NAMESPACE DECLARATIONS

6.1) What is the scope of an XML namespace declaration?

The *scope* of an XML namespace declaration is that part of an XML document to which the declaration applies. An XML namespace declaration remains in scope for the element on which it is declared and all of its descendants, unless it is overridden or undeclared on one of those descendants (see questions 4.5, 4.6, and 4.8).

For example, in the following, the scope of the declaration of the http://www.foo.org/ namespace is the element A and its descendants (B and C). The scope of the declaration of the http://www.bar.org/ namespace is only the element C.

```
<foo:A xmlns:foo="http://www.foo.org/">
   <foo:B>
      <bar:C xmlns:bar="http://www.bar.org/" />
   </foo:B>
</foo:A>
```

6.2) Does the scope of an XML namespace declaration include the element it is

declared on?

Yes.

For example, in the following, the names B and C are in the http://www.bar.org/ namespace, not the http://www.foo.org/ namespace. This is because the declaration that associates the foo prefix with the http://www.bar.org/ namespace occurs on the B element, overriding the declaration on the A element that associates it with the http://www.foo.org/ namespace.

```
<foo:A xmlns:foo="http://www.foo.org/">
   <foo:B xmlns:foo="http://www.bar.org/">
      <foo:C>abcd</foo:C>
   </foo:B>
</foo:A>
```

Similarly, in the following, the names B and C are in the http://www.bar.org/ namespace, not the http://www.foo.org/ namespace because the declaration declaring http://www.bar.org/ as the default XML namespace occurs on the B element, overriding the declaration on the A element.

```
<A xmlns="http://www.foo.org/">
   <B xmlns="http://www.bar.org/">
      <C>abcd</C>
   </B>
</A>
```

A final example is that, in the following, the attribute name D is in the http://www.bar.org/ namespace.

```
<foo:A xmlns:foo="http://www.foo.org/">
   <foo:B foo:D="In http://www.bar.org/ namespace"
          xmlns:foo="http://www.bar.org/">
      <C>abcd</C>
   </foo:B>
</foo:A>
```

One consequence of XML namespace declarations applying to the elements they occur on is that they actually apply before they appear. Because of this, software that processes qualified names should be particularly careful to scan the attributes of an element for XML namespace declarations before deciding what XML namespace (if any) an element type or attribute name belongs to.

6.3) If an element or attribute is in the scope of an XML namespace declaration, is its name in that namespace?

Not necessarily.

When an element or attribute is in the scope of an XML namespace declaration, the element or attribute's name is checked to see if it has a prefix that matches the prefix in the declaration. Whether the name is actually in the XML namespace depends on whether the prefix matches. For example, in the following, the element type names A, B, and D and the attribute names C and E are in the scope of the declaration of the http://www.foo.org/ namespace. While the names A, B, and C are in that namespace, the names D and E are not.

```
<foo:A xmlns:foo="http://www.foo.org/">
   <foo:B foo:C="foo" />
   <bar:D bar:E="bar" />
</foo:A>
```

6.4) What happens when an XML namespace declaration goes out of scope?

When an XML namespace declaration goes out of scope, it simply no longer applies. For example, in the following, the declaration of the http://www.foo.org/ namespace does not apply to the C element because this is outside its scope. That is, it is past the end of the B element, on which the http://www.foo.org/ namespace was declared.

```
<!-- B is in the http://www.foo.org/ namespace; C is not in any
XML namespace. -->
<A>
   <B xmlns="http://www.foo.org/">abcd</B>
   <C>efgh</C>
</A>
```

In addition to the declaration no longer applying, any declarations that it overrode come back into scope. For example, in the following, the declaration of the http://www.foo.org/ namespace is brought back into scope after the end of the B element. This is because it was overridden on the B element by the declaration of the http://www.bar.org/ namespace.

```
<!-- A and C are in the http://www.foo.org/ namespace. B is in
the http://www.bar.org/ namespace. -->
<A xmlns="http://www.foo.org/">
   <B xmlns="http://www.bar.org/">abcd</B>
   <C>efgh</C>
</A>
```

6.5) What happens if no XML namespace declaration is in scope?

If no XML namespace declaration is in scope, then any prefixed element type or attribute names result in namespace errors. For example, in the following, the names foo:A and foo:B result in namespace errors.

```
<?xml version="1.0" ?>
```

```
<foo:A foo:B="error" />
```

In the absence of an XML namespace declaration, unprefixed element type and attribute names do not belong to any XML namespace. For example, in the following, the names A and B are not in any XML namespace. For more information, see question 3.5.

```
<?xml version="1.0" ?>
<A B="no error" />
```

6.6) Can multiple XML namespace declarations be in scope at the same time?

Yes, as long as they don't use the same prefixes and at most one of them is the default XML namespace. For example, in the following, the http://www.foo.org/ and http://www.bar.org/ namespaces are both in scope for all elements:

```
<A xmlns:foo="http://www.foo.org/"
   xmlns:bar="http://www.bar.org/">
   <foo:B>abcd</foo:B>
   <bar:C>efgh</bar:C>
</A>
```

One consequence of this is that you can place all XML namespace declarations on the root element and they will be in scope for all elements. This is the simplest way to use XML namespaces.

6.7) How can I declare XML namespaces so that all elements and attributes are in their scope?

XML namespace declarations that are made on the root element are in scope for all elements and attributes in the document. This means that an easy way to declare XML namespaces is to declare them only on the root element. For example:

```
<Department
       xmlns:addr="http://www.tu-darmstadt.de/ito/addresses"
       xmlns:serv="http://www.tu-darmstadt.de/ito/servers">
   <Name>DVS1</Name>
   <addr:Address>
      <addr:Street>Wilhelminenstr. 7</addr:Street>
      <addr:City>Darmstadt</addr:City>
      <addr:State>Hessen</addr:State>
      <addr:Country>Germany</addr:Country>
      <addr:PostalCode>D-64285</addr:PostalCode>
   </addr:Address>
   <serv:Server>
      <serv:Name>OurWebServer</serv:Name>
      <serv:Address>123.45.67.8</serv:Address>
```

```
    </serv:Server>
  </Department>
```

6.8) Does the scope of an XML namespace declaration ever include the DTD?

No.

XML namespaces can be declared only on elements and their scope consists only of those elements and their descendants. Thus, the scope can never include the DTD. For more information, see question 7.2.

7) SECTION 7: XML NAMESPACES AND DTDs

7.1) Can I use XML namespaces in DTDs?

Yes and no.

In particular, DTDs can contain qualified names (see question 7.3) but XML namespace declarations do not apply to DTDs (see question 7.2).

This has a number of consequences. Because XML namespace declarations do not apply to DTDs:

1. There is no way to determine what XML namespace a prefix in a DTD points to. Which means...
2. Qualified names in a DTD cannot be mapped to universal names. Which means...
3. Element type and attribute declarations in a DTD are expressed in terms of qualified names, not universal names. Which means...
4. Validation cannot be redefined in terms of universal names as might be expected.

This situation has caused numerous complaints but, as XML namespaces are already a recommendation, is unlikely to change. The long term solution to this problem is an XML schema language: all of the proposed XML schema languages provide a mechanism by which the local name in an element type or attribute declaration can be associated with an XML namespace. This makes it possible to redefine validity in terms of universal names.

7.2) Do XML namespace declarations apply to DTDs?

No.

In particular, an xmlns attribute declared in the DTD with a default is not an XML namespace declaration for the DTD. For more information, see question 4.4. (Note that an earlier version of MSXML (the parser used by Internet Explorer) did use such declarations as XML namespace declarations, but that this was removed in MSXML 4. For more information, see question 8.6.)

7.3) Can I use qualified names in DTDs?

Yes.

For example, the following is legal:

```
<!ELEMENT foo:A (foo:B)>
<!ATTLIST foo:A
          foo:C CDATA #IMPLIED>
<!ELEMENT foo:B (#PCDATA)>
```

However, because XML namespace declarations do not apply to DTDs (see question 7.2), qualified names in the DTD cannot be converted to universal names. As a result, qualified names in the DTD have no special meaning. For example, foo:A is just foo:A -- it is not A in the XML namespace to which the prefix foo is mapped.

The reason qualified names are allowed in the DTD is so that validation will continue to work. For more information, see question 7.6.

7.4) Can the content model in an element type declaration contain element types whose names come from other XML namespaces?

Yes and no.

The answer to this question is yes in the sense that a qualified name in a content model can have a different prefix than the qualified name of the element type being declared. For example, the following is legal:

```
<!ELEMENT foo:A (bar:B, baz:C)>
```

The answer to this question is no in the sense that XML namespace declarations do not apply to DTDs so the prefixes used in an element type declaration are technically meaningless. In particular, they do not specify that the name of a certain element type belongs to a certain namespace. Nevertheless, the ability to mix prefixes in this manner is crucial when: a) you have a document whose names come from multiple XML namespaces (see question 8.5), and b) you want to construct that document in a way that is both valid and conforms to the XML namespaces recommendation (see question 7.6).

7.5) Can the attribute list of an element type contain attributes whose names come from other XML namespaces?

Yes and no, for the reasons listed in question 7.4.

For example, the following is legal:

```
<!ATTLIST foo:A
          bar:B CDATA #IMPLIED>
```

7.6) How can I construct an XML document that is valid and conforms to the XML namespaces recommendation?

In answering this question, it is important to remember that:

- Validity is a concept defined in XML 1.0,
- XML namespaces are layered on top of XML 1.0 (see myth #11 in "Namespace Myths Exploded"), and
- The XML namespaces recommendation does not redefine validity, such as in terms of universal names (see myth #9 in "Namespace Myths Exploded").

Thus, validity is the same for a document that uses XML namespaces and one that doesn't. In particular, with respect to validity:

- xmlns attributes are treated as attributes, not XML namespace declarations.
- Qualified names are treated like other names. For example, in the name foo:A, foo is not treated as a namespace prefix, the colon is not treated as separating a prefix from a local name, and A is not treated as a local name. The name foo:A is treated simply as the name foo:A.

Because of this, XML documents that you might expect to be valid are not. For example, the following document is not valid because the element type name A is not declared in the DTD, in spite of the fact both foo:A and A share the universal name {http://www.foo.org/}A:

```
<?xml version="1.0" ?>
<!DOCTYPE foo:A [
   <!ELEMENT foo:A (#PCDATA)>
   <!ATTLIST foo:A
             xmlns:foo CDATA #FIXED "http://www.foo.org/"
             xmlns CDATA #FIXED "http://www.foo.org/">
]>
<A/>
```

Similarly, the following is not valid because the xmlns attribute is not declared in the DTD:

```
<?xml version="1.0" ?>
<!DOCTYPE A [
   <!ELEMENT A (#PCDATA)>
]>
<A xmlns="http://www.foo.org/" />
```

Furthermore, documents that you might expect to be invalid are valid. For example, the following document is valid but contains two definitions of the element type with the universal name {http://www.foo.org/}A:

```
<?xml version="1.0" ?>
<!DOCTYPE foo:A [
   <!ELEMENT foo:A (bar:A)>
   <!ATTLIST foo:A
             xmlns:foo CDATA #FIXED "http://www.foo.org/">
   <!ELEMENT bar:A (#PCDATA)>
   <!ATTLIST bar:A
             xmlns:bar CDATA #FIXED "http://www.foo.org/">
]>
<foo:A>
   <bar:A>abcd</bar:A>
</foo:A>
```

Finally, validity has nothing to do with correct usage of XML namespaces. For example, the following document is valid but does not conform to the XML namespaces recommendation because the foo prefix is never declared:

```
<?xml version="1.0" ?>
<!DOCTYPE foo:A [
   <!ELEMENT foo:A (#PCDATA)>
]>
<foo:A />
```

Therefore, when constructing an XML document that uses XML namespaces, you need to do both of the following if you want the document to be valid:

- Declare xmlns attributes in the DTD.
- Use the same qualified names in the DTD and the body of the document.

For example:

```
<?xml version="1.0" ?>
<!DOCTYPE foo:A [
   <!ELEMENT foo:A (foo:B)
   <!ATTLIST foo:A
             xmlns:foo CDATA #FIXED "http://www.foo.org/">
   <!ELEMENT foo:B (#PCDATA)>
]>
<foo:A>
   <foo:B>abcd</foo:B>
</foo:A>
```

There is no requirement that the same prefix always be used for the same XML namespace. For example, the following is also valid:

```
<?xml version="1.0" ?>
<!DOCTYPE foo:A [
   <!ELEMENT foo:A (bar:B)>
   <!ATTLIST foo:A
             xmlns:foo CDATA #FIXED "http://www.foo.org/">
   <!ELEMENT bar:B (#PCDATA)>
   <!ATTLIST bar:B
             xmlns:bar CDATA #FIXED "http://www.foo.org/">
]>
<foo:A>
   <bar:B />
</foo:A>
```

However, documents that use multiple prefixes for the same XML namespace or the same prefix for multiple XML namespaces are confusing to read and thus prone to error. They also allow abuses such as defining an element type or attribute with a given universal name more than once, as was seen earlier. Therefore, a better set of guidelines for writing documents that are both valid and conform to the XML namespaces recommendation is:

- Declare all xmlns attributes in the DTD.
- Use the same qualified names in the DTD and the body of the document.
- Use one prefix per XML namespace.
- Do not use the same prefix for more than one XML namespace.
- Use at most one default XML namespace.

The latter three guidelines guarantee that prefixes are unique. This means that prefixes fulfill the role normally played by namespace names (URIs) -- uniquely identifying an XML namespace -- and that qualified names are equivalent to universal names, so a given universal name is always represented by the same qualified name. Although this is contrary to the spirit of prefixes, which were designed for their flexibility, there is no other solution at this point.

7.7) How can I validate an XML document that uses XML namespaces?

When people ask this question, they usually assume that validity is different for documents that use XML namespaces and documents that don't. In fact, it isn't -- it's the same for both. Thus, there is no difference between validating a document that uses XML namespaces and validating one that doesn't. In either case, you simply use a validating parser or other software that performs validation.

For information about how to construct an XML document that is valid and conforms to the XML namespace recommendation, see question 7.6.

7.8) If I start using XML namespaces, do I need to change my existing DTDs?

Probably.

If you want your XML documents to be both valid and conform to the XML namespaces recommendation, you need to declare any xmlns attributes and use the same qualified names in the DTD as in the body of the document. (For more information, see question question 7.6.)

If your DTD contains element type and attribute names from a single XML namespace, the easiest thing to do is to use your XML namespace as the default XML namespace. To do this, declare the attribute xmlns (no prefix) for each possible root element type. If you can guarantee that the DTD is always read (see question 4.3), set the default value in each xmlns attribute declaration to the URI used as your namespace name. Otherwise, declare your XML namespace as the default XML namespace on the root element of each instance document.

If your DTD contains element type and attribute names from multiple XML namespaces, you need to choose a single prefix for each XML namespace and use these consistently in qualified names in both the DTD and the body of each document. You also need to declare your xmlns attributes in the DTD and declare your XML namespaces. As in the single XML namespace case, the easiest way to do this is add xmlns attributes to each possible root element type and use default values if possible.

Note that you should only need to make these changes once.

8) SECTION 8: USING DOCUMENTS THAT USE XML NAMESPACES

8.1) How do I create documents that use XML namespaces?

The same as you create documents that don't use XML namespaces. If you're currently using Notepad on Windows or emacs on Linux, you can continue using Notepad or emacs. If you're using an XML editor that is not namespace-aware, you can also continue to use that, as qualified names are legal names in XML documents and xmlns attributes are legal attributes. And if you're using an XML editor that is namespace-aware, it will probably provide features such as automatically declaring XML namespaces and keeping track of prefixes and the default XML namespace for you.

8.2) How can I check that a document conforms to the XML namespaces recommendation?

Unfortunately, I know of no software that only checks for conformance to the XML namespaces recommendation. It is possible that some namespace-aware validating parsers (such as those from DataChannel (Microsoft), IBM, Oracle, or Sun) check XML namespace conformance as part of parsing and validating. Thus, you might be able to run your document through such parsers as a way of testing conformance.

Note that writing an application to check conformance to the XML namespaces recommendation is

not as easy as it might seem. The problem is that most parsers do not make DTD information available to the application, so it might not be possible to check conformance in the DTD. Also note that writing a SAX 1.0 application that checks conformance in the body of the document (as opposed to the DTD) should be an easy thing to do. (John Cowan's Namespace SAX Filter probably already does this. For more information, see question 9.2.)

8.3) Can I use the same document with both namespace-aware and namespace-unaware applications?

Yes.

This situation is quite common, such as when a namespace-aware application is built on top of a namespace-unaware parser. Another common situation is when you create an XML document with a namespace-unaware XML editor but process it with a namespace-aware application.

Using the same document with both namespace-aware and namespace-unaware applications is possible because XML namespaces use XML syntax. That is, an XML document that uses XML namespaces is still an XML document and is recognized as such by namespace-unaware software.

The only thing you need to be careful about when using the same document with both namespace-aware and namespace-unaware applications is when the namespace-unaware application requires the document to be valid. In this case, you must be careful to construct your document in a way that is both valid and conforms to the XML namespaces recommendation. (It is possible to construct documents that conform to the XML namespaces recommendation but are not valid and vice versa.) For information about how to do this, see question 7.6.

8.4) What software is needed to process XML namespaces?

From a document author's perspective, this is generally not a relevant question. Most XML documents are written in a specific XML language and processed by an application that understands that language. If the language uses an XML namespace, then the application will already use that namespace -- there is no need for any special XML namespace software.

For information about the software application writers use to process XML namespaces, see section 9.

8.5) How can I use XML namespaces to combine documents that use different element type and attribute names?

Before we answer this question, we need to clear up a common misconception about XML namespaces. It is often believed that XML namespaces provide some sort of magic that allows you to automatically combine XML documents that use separate sets of element types and attributes. This is not true. Although XML namespaces provide some of the tools you need to do this, you still need to do most of the work yourself, as you will see in the answer to this question.

To combine documents that use separate sets of element types and attributes, all you really need to do is decide where the elements and attributes go in the final document. For example, does element type A from the first document go inside element type B? Is it a sibling? Or are they completely unrelated? Although the procedure for actually combining two documents can be easily automated, deciding how to combine them is not likely to ever be automated. Instead, it is a strictly human problem, requiring somebody to make choices.

For example, suppose we have a document containing addresses:

```
<Address>
   <Street>Wilhelminenstr. 7</Street>
   <City>Darmstadt</City>
   <State>Hessen</State>
   <Country>Germany</Country>
   <PostalCode>D-64285</PostalCode>
</Address>
```

and a document containing Web server information:

```
<Server>
   <Name>OurWebServer</Name>
   <Address>123.45.67.8</Address>
</Server>
```

Now suppose we would like a document of departments, their addresses, and their Web servers, which we can make from the above documents plus a little additional information:

```
<Departments>
   <Department>
      <Name>DVS1</Name>
      <Address>
         <Street>Wilhelminenstr. 7</Street>
         <City>Darmstadt</City>
         <State>Hessen</State>
         <Country>Germany</Country>
         <PostalCode>D-64285</PostalCode>
      </Address>
      <Server>
         <Name>OurWebServer</Name>
         <Address>123.45.67.8</Address>
      </Server>
   </Department>
   ...
</Departments>
```

Unfortunately, we have a problem: there are two Address element types and two Name element types. (Although the Name element types both have a content model of PCDATA, they have different meanings -- one is the name of the department and the other is the name of the server -- so we would like to recognize them as different element types.)

To solve these conflicts, we can use XML namespaces. We assign the address information to the http://www.tu-darmstadt.de/ito/addresses namespace, the server information to the http://www.tu-darmstadt.de/ito/servers namespace, and the newly added departmental information to the http://www.tu-darmstadt.de/ito/depts namespace. Thus, our new document looks like:

```
<dept:Departments
      xmlns:dept="http://www.tu-darmstadt.de/ito/depts
      xmlns:addr="http://www.tu-darmstadt.de/ito/addresses
      xmlns:serv="http://www.tu-darmstadt.de/ito/servers>
   <dept:Department>
      <dept:Name>DVS1</dept:Name>
      <addr:Address>
         <addr:Street>Wilhelminenstr. 7</addr:Street>
         <addr:City>Darmstadt</addr:City>
         <addr:State>Hessen</addr:State>
         <addr:Country>Germany</addr:Country>
         <addr:PostalCode>D-64285</addr:PostalCode>
      </addr:Address>
      <serv:Server>
         <serv:Name>OurWebServer</serv:Name>
         <serv:Address>123.45.67.8</serv:Address>
      </serv:Server>
   </dept:Department>
   ...
</dept:Departments>
```

Although we could have assigned only the conflicting element type names to XML namespaces, it is generally a better idea to keep the names of related element types in a single XML namespace. This avoids questions of having to remember which names have been placed in XML namespaces and which haven't, as well as avoiding future conflicts that might occur when these types are used elsewhere. (This does not necessarily mean that the names of all related element types and attributes should be placed in a single XML namespace. If you have a large and complex set of element types and attributes and expect to reuse subsets of it elsewhere, it might make more sense to place the names of each reusable subset in its own XML namespace.)

This example was fairly simple in that it did not require us to change any existing content models. However, suppose we wanted to keep information only about servers: their name, IP address, and physical location. In this case, we might want to add the Address element type that describes street address to the content of the Server element type:

```
<serv:Servers
      xmlns:addr="http://www.tu-darmstadt.de/ito/addresses
      xmlns:serv="http://www.tu-darmstadt.de/ito/servers2>
   <serv:Server>
      <serv:Name>OurWebServer</serv:Name>
      <serv:Address>123.45.67.8</serv:Address>
      <addr:Address>
         <addr:Street>Wilhelminenstr. 7</addr:Street>
         <addr:City>Darmstadt</addr:City>
         <addr:State>Hessen</addr:State>
         <addr:Country>Germany</addr:Country>
         <addr:PostalCode>D-64285</addr:PostalCode>
      </addr:Address>
   </serv:Server>
   ...
</serv:Servers>
```

Notice that we have changed the name of the XML namespace for the server information to http://www.tu-darmstadt.de/ito/servers2. This is because we changed the content model of the Server element type. In other words, the content models of the two Server element types are different, so they are two different element types with the same name and should reside in separate XML namespaces. Had we not made this change, software searching for the old Server element type would have incorrectly processed the new Server element type and vice versa.

As you can see, there is nothing magic about combining documents. It is simply a matter of deciding how to fit them together and, although there are likely to be tools in the future that make this as easy as drag-and-drop, the decisions about exactly how to combine them are likely to always require human intervention.

You can also see that XML namespaces played a vital role in this process -- they allowed us to combine the documents without changing the local names of any of the element types. This is important, as without XML namespaces we would endlessly have to invent new ways of renaming simple element types like Address. However, it is also important to understand that resolving duplicate names is the only role XML namespaces played in this process -- the rest was left to the people making the decisions.

8.6) How do I use XML namespaces with Internet Explorer 5.0 and/or the MSXML parser?

WARNING! The following applies only to earlier versions of MSXML. It does not apply to MSXML 4, which is the currently shipping version [July, 2002].

An early version of the MSXML parser, which was shipped as part of Internet Explorer 5.0, required that every XML namespace prefix used in an element type or attribute declaration had to be "declared" in the attribute declaration for that element type. This had to be done with a fixed xmlns

attribute declaration. For example, the following was accepted by MSXML and both xmlns:foo attributes were required:

```
<!ELEMENT foo:A (#PCDATA)>
<!ATTLIST foo:A
          xmlns:foo CDATA #FIXED "http://www.foo.org/">
<!ELEMENT foo:B (#PCDATA)>
<!ATTLIST foo:B
          xmlns:foo CDATA #FIXED "http://www.foo.org/">
```

MSXML returned an error for the following because the second foo prefix was not "declared":

```
<!ELEMENT foo:A (#PCDATA)>
<!ATTLIST foo:A
          xmlns:foo CDATA #FIXED "http://www.foo.org/">
<!ELEMENT foo:B (#PCDATA)>
```

The reason for this restriction was so that MSXML could use universal names to match element type and attribute declarations to elements and attributes during validation. Although this would have simplified many of the problems of writing documents that are both valid and conform to the XML namespaces recommendation (see question 7.6), some users complained about it because it was not part of the XML namespaces recommendation. In response to these complaints, Microsoft removed this restriction in later versions, which are now shipping. Ironically, the idea was later independently derived as a way to resolve the problems of validity and namespaces. However, it has not been implemented by anyone.

9) SECTION 9: PROCESSING XML NAMESPACES IN XML APPLICATIONS

9.1) How do applications process documents that use XML namespaces?

Applications process documents that use XML namespaces in almost exactly the same way they process documents that don't use XML namespaces. For example, if a namespace-unaware application adds a new sales order to a database when it encounters a SalesOrder element, the equivalent namespace-aware application does the same. The only difference is that the namespace-aware application:

- Might need to check for xmlns attributes and parse qualified names. Whether it does this depends on whether such processing is already done by lower-level software, such as a namespace-aware DOM implementation.
- Uses universal (two-part) names instead of local (one-part) names. For example, the namespace-aware application might add a new sales order in response to an {http://www.tu-darmstadt.de/ito/sales}SalesOrder element instead of a SalesOrder element.

9.2) How do I use XML namespaces with SAX 1.0?

The easiest way to use XML namespaces with SAX 1.0 is to use John Cowan's Namespace SAX Filter (see http://www.ccil.org/~cowan/XML). This is a SAX filter that keeps track of XML namespace declarations, parses qualified names, and returns element type and attribute names as universal names in the form:

```
URI^local-name
```

For example:

```
http://www.tu-darmstadt.de/ito/sales^SalesOrder
```

Your application can then base its processing on these longer names. For example, the code:

```
public void startElement(String elementName, AttributeList attrs)
   throws SAXException
{
   ...
   if (elementName.equals("SalesOrder"))
   {
      // Add new database record.
   }
   ...
}
```

might become:

```
public void startElement(String elementName, AttributeList attrs)
   throws SAXException
{
   ...
   if (elementName.equals("http://www.tu-darmstadt.de/
sales^SalesOrder"))
   {
      // Add new database record.
   }
   ...
}
```

or:

```
public void startElement(String elementName, AttributeList attrs)
   throws SAXException
{
   ...
   // getURI() and getLocalName() are utility functions to parse
```

```
universal names.

      if (getURI(elementName).equals("http://www.tu-darmstadt.de/
ito/sales"))
      {
         if (getLocalName(elementName).equals("SalesOrder"))
         {
            // Add new database record.
         }
      }
      ...
   }
```

If you do not want to use the Namespace SAX Filter, then you will need to do the following in addition to identifying element types and attributes by their universal names:

- In startElement, scan the attributes for XML namespace declarations before doing any other processing. You will need to maintain a table of current prefix-to-URI mappings (including a null prefix for the default XML namespace).
- In startElement and endElement, check whether the element type name includes a prefix. If so, use your mappings to map this prefix to a URI. Depending on how your software works, you might also check if the local part of the qualified name includes any colons, which are illegal.
- In startElement, check whether attribute names include a prefix. If so, process as in the previous point.

9.3) How do I use XML namespaces with SAX 2.0?

SAX 2.0 primarily supports XML namespaces through the following methods:

- startElement and endElement in the ContentHandler interface return namespace names (URIs) and local names as well as qualified names.
- getValue, getType, and getIndex in the Attributes interface can retrieve attribute information by namespace name (URI) and local name as well as by qualified name.

For example, the namespace-unaware SAX 1.0 code:

```
   public void startElement(String elementName, AttributeList attrs)
      throws SAXException
   {
      ...
      if (elementName.equals("SalesOrder"))
      {
         // Add new database record.
      }
      ...
```

```
}
```

might become:

```
public void startElement(String namespaceURI,
                         String localName,
                         String qualifiedName,
                         Attributes attrs)
   throws SAXException
{
   ...
   if (namespaceURI.equals("http://www.tu-darmstadt.de/sales") &&
       localName.equals("SalesOrder"))
   {
      // Add new database record.
   }
   ...
}
```

Although the above methods are sufficient for most applications, SAX 2.0 also supports the following:

- startPrefixMapping and endPrefixMapping in the ContentHandler interface return scope information about individual prefixes.
- getURI, getLocalName, and getQName in the Attributes interface return the namespace name (URI), local name, and qualified name of an attribute by index.
- The http://xml.org/features/namespaces and http://xml.org/features/namespace-prefixes properties allow applications to request whether parsers return qualified names and xmlns attributes.
- The NamespaceSupport class helps applications track the currently declared namespace prefixes and names (URIs).

For complete details, see the SAX 2.0 specification (http://www.saxproject.org/).

9.4) How do I use XML namespaces with DOM level 1?

This depends on what DOM level 1 implementation you are using. If you are using a namespace-aware DOM implementation, such as those from Data Channel (Microsoft), IBM, Oracle, or Sun, then all you need to do is use the methods provided by those implementations to retrieve the namespace name (URI) and local names. For example, the namespace-unaware code:

```
// Check the local name.
// getNodeName() is a DOM level 1 method.

if (elementNode.getNodeName().equals("SalesOrder"))
```

```
  {
     // Add new database record.
  }
```

might become the following namespace-aware code when using Oracle's DOM implementation:

```
  // Check the XML namespace name (URI).
  // getNamespace() and NSElement are Oracle-specific.

  if ((NSElement)elementNode).getNamespace().equals("http://www.tu-
darmstadt.de/ito/sales"))
  {

     // Check the local name.
     // getLocalName() is Oracle-specific.

     if ((NSElement)elementNode.getLocalName().equals("SalesOrder"))
     {
        // Add new database record.
     }
  }
```

Because DOM level 1 does not include XML namespace support, each DOM level 1 implementation provides a slightly different interface for accessing XML namespace information. Most of these implementations include some combination of methods for getting the qualified name, namespace name (URI), local name, and prefix. Therefore, if you are writing a DOM-neutral application, you will need to define your own XML namespace interface and write converters for each DOM implementation you support. Fortunately, this is easy to do.

If you are using a namespace-unaware DOM implementation, such as Docuverse version 1 or OpenXML version 1 (these may support XML namespaces in a later version), you will need to perform namespace processing yourself. Although this is largely the same as in a SAX 1.0 application (see question 9.2) -- checking for xmlns attributes on each Element node and converting qualified names on Element and Attr nodes to universal names -- it is potentially more difficult to determine what XML namespace declarations are in scope for any given node. If your application traverses the DOM tree in order from the root, this is fairly easy, as you can maintain prefix-to-URI mappings as you go. However, if you access the tree randomly, you may want to construct a parallel tree with mapping information before you do any other processing or rebuild the DOM tree using prefixes that map to known URIs.

9.5) How do I use XML namespaces with DOM level 2?

DOM level 2 supports XML namespaces through a number of new methods and attributes. The most important of these are the namespaceURI and localName attributes in the Node interface; these allow applications to identify nodes by their namespace name (URI) and local name rather than by qualified

name. For example, the namespace-unaware code:

```
  // Check the local name.
  // getNodeName() is a DOM level 1 method.

  if (elementNode.getNodeName().equals("SalesOrder"))
  {
     // Add new database record.
  }
```

might become the following namespace-aware code:

```
  // Check the XML namespace name (URI).
  // getNamespaceURI() is a DOM level 2 method.

  if (elementNode.getNamespaceURI().equals("http://www.tu-darmstadt.
de/ito/sales"))
  {

     // Check the local name.
     // getLocalName() is a DOM level 2 method.

     if (elementNode.getLocalName().equals("SalesOrder"))
     {
        // Add new database record.
     }
  }
```

Note that, unlike SAX 2.0, DOM level 2 treats xmlns attributes as normal attributes.

For complete details, see the DOM level 2 recommendation (http://www.w3.org/TR/DOM-Level-2/).

9.6) Can an application process documents that use XML namespaces and documents that don't use XML namespaces?

Yes.

This is a common situation for generic applications, such as editors, browsers, and parsers, that are not wired to understand a particular XML language. Such applications simply treat all element type and attribute names as qualified names. Those names that are not mapped to an XML namespace -- that is, unprefixed element type names in the absence of a default XML namespace and unprefixed attribute names -- are simply processed as one-part names, such as by using a null XML namespace name (URI).

Note that such applications must decide how to treat documents that do not conform to the XML

namespaces recommendation. For example, what should the application do if an element type name contains a colon (thus implying the existence of a prefix), but there are no XML namespace declarations in the document? The application can choose to treat this as an error, or it can treat the document as one that does not use XML namespaces, ignore the "error", and continue processing.

9.7) Can an application be both namespace-aware and namespace-unaware?

Yes.

However, there is generally no reason to do this. The reason is that most applications understand a particular XML language, such as one used to transfer sales orders between companies. If the element type and attribute names in the language belong to an XML namespace, the application must be namespace-aware; if not, the application must be namespace-unaware. For example, such an application should never need to recognize the element type names SalesOrder and {http://www.tu-darmstadt.de/ito/sales}SalesOrder.

For a few applications, being both namespace-aware and namespace-unaware makes sense. For example, a parser might choose to redefine validity in terms of universal names (see myth #9 in "Namespace Myths Exploded") and have both namespace-aware and namespace-unaware validation modes. However, such applications are uncommon.

9.8) What does a namespace-aware application do when it encounters an error?

The XML namespaces recommendation does not specify what a namespace-aware application does when it encounters a document that does not conform to the recommendation. Therefore, the behavior is application-dependent. For example, the application could stop processing, post an error to a log and continue processing, or ignore the error.

PART III: NAMES, PREFIXES, AND URIs

10) SECTION 10: NAMES

10.1) What is a qualified name?

A *qualified name* is a name of the following form. It consists of an optional *prefix* and colon, followed by the *local part*, which is sometimes known as a *local name*.

```
prefix:local-part
      --OR--
local-part
```

For example, both of the following are qualified names. The first name has a prefix of serv; the second name does not have a prefix. For both names, the local part (local name) is Address.

```
serv:Address
Address
```

In most circumstances, qualified names are mapped to universal names. For more information, see question 10.5.

10.2) What characters are allowed in a qualified name?

The prefix can contain any character that is allowed in the Name [5] production in XML 1.0 except a colon. The same is true of the local name. Thus, there can be at most one colon in a qualified name -- the colon used to separate the prefix from the local name.

10.3) Where can qualified names appear?

Qualified names can appear anywhere an element type or attribute name can appear: in start and end tags, as the document element type, and in element type and attribute declarations in the DTD. For example:

```
<!DOCTYPE foo:A [
   <!ELEMENT foo:A (foo:B)>
   <!ATTLIST foo:A
             foo:C CDATA #IMPLIED>
   <!ELEMENT foo:B (#PCDATA)>
]>
<foo:A xmlns:foo="http://www.foo.org/" foo:C="bar">
   <foo:B>abcd</foo:B>
<foo:A>
```

Qualified names cannot appear as entity names, notation names, or processing instruction targets.

10.4) Can qualified names be used in attribute values?

Yes, but they have no special significance. That is, they are not necessarily recognized as such and mapped to universal names. For example, the value of the C attribute in the following is the string "foo:D", not the universal name {http://www.foo.org/}D.

```
<foo:A xmlns:foo="http://www.foo.org/">
   <foo:B C="foo:D"/>
<foo:A>
```

In spite of this, there is nothing to stop an application from recognizing a qualified name in an attribute value and processing it as such. This is being done in various technologies today. For example, in the following XML Schemas definition, the attribute value xsd:string identifies the type of the foo attribute as the universal name {http://www.w3.org/1999/XMLSchema}string.

```
<xsd:attribute name="foo" type="xsd:string" />
```

There are two potential problems with this. First, the application must be able to retrieve the prefix mappings currently in effect. Fortunately, both SAX 2.0 and DOM level 2 support this capability. Second, any general purpose transformation tool, such as one that writes an XML document in canonical form and changes namespace prefixes in the process, will not recognize qualified names in attribute values and therefore not transform them correctly. Although this may be solved in the future by the introduction of the QName (qualified name) data type in XML Schemas, it is a problem today.

10.5) How are qualified names mapped to names in XML namespaces?

If a qualified name in the body of a document (as opposed to the DTD) includes a prefix, then that prefix is used to map the local part of the qualified name to a *universal name* -- that is, a name in an XML namespace. (Note that the prefix must be in scope -- see section 6.) For example, in the following, the prefix foo is used to map the local names A, B, and C to names in the http://www.foo.org/ namespace:

```
<?xml version="1.0" ?>
<foo:A xmlns:foo="http://www.foo.org/" foo:C="bar">
   <foo:B>abcd</foo:B>
<foo:A>
```

If a qualified name in the body of a document does not include a prefix and a default XML namespace is in scope then one of two things happens. If the name is used as an element tag, it is mapped to a name in the default XML namespace. If it is used as an attribute name, it is not in any XML namespace (see myth #4 in "Namespace Myths Exploded"). For example, in the following, A and B are in the http://www.foo.org/ namespace and C is not in any XML namespace:

```
<?xml version="1.0" ?>
<A xmlns="http://www.foo.org/" C="bar">
   <B>abcd</B>
<A>
```

If a qualified name in the body of a document does not include a prefix and no default XML namespace is in scope, then that name is not in any XML namespace. For example, in the following, A, B, and C are not in any XML namespace:

```
<?xml version="1.0" ?>
<A C="bar">
   <B>abcd</B>
<A>
```

Qualified names in the DTD are never mapped to names in an XML namespace because they are never in the scope of an XML namespace declaration. (For more information, see question 7.2.)

10.6) What is a prefixed name?

A prefixed name is a qualified name (see question 10.1) that contains a prefix.

10.7) What is an unprefixed name?

An unprefixed name is a qualified name (see question 10.1) that does not contain a prefix.

10.8) Are unprefixed names in an XML namespace?

Only if they are used in the body of a document (as opposed to the DTD) as an element tag and a default XML namespace is in scope. For more information, see question 5.2.

10.9) What is a local name?

This is another term for the local part of a qualified name. For more information, see question 10.1.

10.10) What is a namespace name?

This is the name used to identify a namespace. It is a URI. For more information, see question 12.2.

10.11) What is a universal name?

A *universal name* is a two part name consisting of an XML namespace name (URI) and a local name. For example, in the following, the universal name is the namespace name "http://www.tu-darmstadt.de/ito/servers" plus the local name "Address":

```
  <serv:Address xmlns:serv="http://www.tu-darmstadt.de/ito/
servers">123.45.67.8</serv:Address>
```

The ability to specify universal names is the only function of XML namespaces.

10.12) How are universal names represented?

There is no standard way to represent a universal name. However, three representations are common.

The first representation keeps the XML namespace name (URI) and the local name separate. For example, many DOM level 1 implementations have different methods for returning the XML namespace name (URI) and the local name of an element or attribute node.

The second representation concatenates the namespace name (URI) and the local name with caret (^). The result is a universally unique (see question 10.12) name, since carets are not allowed in URIs or

local names. This is the method used by John Cowan's Namespace SAX Filter (see question 9.2). For example, the universal name that has the URI http://www.tu-darmstadt.de/ito/servers and the local name Address would be represented as:

```
http://www.tu-darmstadt.de/ito/servers^Address
```

The third representation, which we use in this FAQ, was suggested by James Clark in his XML Namespaces paper (see question 14.1). It places the XML namespace name (URI) in braces and concatenates this with the local name. This notation is suggested only for documentation and I am aware of no code that uses it. For example, the above name would be represented as:

```
{http://www.tu-darmstadt.de/ito/servers}Address
```

10.13) Are universal names universally unique?

No, but it is reasonable to assume they are.

Universal element type and attribute names are not guaranteed to be universally unique -- that is, unique within the space of all XML documents -- because it is possible for two different people, each defining their own XML namespace, to use the same URI and the same element type or attribute name. However, this occurs only if:

- One or both people use a URI that is not under their control, such as somebody outside Netscape using the URI http://www.netscape.com/, or
- Both people have control over a URI and both use it.

The first case means somebody is cheating when assigning URIs (a process governed by trust) and the second case means that two people within an organization are not paying attention to each other's work. For widely published element type and attribute names, neither case is very likely. Thus, it is reasonable to assume that universal names are universally unique. (Since both cases are possible, applications that present security risks should be careful about assuming that universal names are universally unique.)

For information about the ability of universal names to uniquely identify element types and attributes (as opposed to the names themselves being unique), see myth #2 in "Namespace Myths Exploded".

11) SECTION 11: XML NAMESPACE PREFIXES

11.1) What is an XML namespace prefix?

An XML namespace prefix is a prefix used to specify that a local element type or attribute name is in a particular XML namespace. For example, in the following, the serv prefix specifies that the Address element type name is in the http://www.tu-darmstadt.de/ito/addresses namespace:

```
  <serv:Addresses xmlns:serv="http://www.tu-darmstadt.de/ito/
addresses">
```

11.2) What characters are allowed in an XML namespace prefix?

The prefix can contain any character that is allowed in the Name [5] production in XML 1.0 except a colon.

11.3) Are prefixes significant?

No.

For example, the following are equivalent:

```
  <serv:Addresses xmlns:serv="http://www.tu-darmstadt.de/ito/
addresses">
```

and

```
  <foo:Addresses xmlns:foo="http://www.tu-darmstadt.de/ito/
addresses">
```

However, prefixes provide a visual cue to readers. Because of this, software (especially editors) should consider preserving prefixes. For example, this is much easier to read:

```
  <xsl:stylesheet version="1.0"
                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                  xmlns:html="http://www.w3.org/TR/xhtml1/strict">
  <xsl:template match="/">
     <html:html>
        <html:head><html:title>Title</html:title></html:head>
        <html:body>
           <xsl:apply-templates/>
        </html:body>
  <xsl:template>
  ...
  </xsl:stylesheet>
```

than this:

```
  <aaa:stylesheet version="1.0"
                  xmlns:aaa="http://www.w3.org/1999/XSL/Transform"
                  xmlns:bbb="http://www.w3.org/TR/xhtml1/strict">
  <aaa:template match="/">
```

```
   <bbb:html>
      <bbb:head><bbb:title>Title</bbb:title></bbb:head>
      <bbb:body>
         <aaa:apply-templates/>
      </bbb:body>
<aaa:template>
...
</aaa:stylesheet>
```

or (much worse yet) this:

```
<html:stylesheet version="1.0"
                xmlns:html="http://www.w3.org/1999/XSL/Transform"
                xmlns:xsl="http://www.w3.org/TR/xhtml1/strict">
<html:template match="/">
   <xsl:html>
      <xsl:head><xsl:title>Title</xsl:title></xsl:head>
      <xsl:body>
         <html:apply-templates/>
      </xsl:body>
<html:template>
...
</html:stylesheet>
```

In spite of this, people are likely to read significance into prefixes, so it is a good idea to use prefixes appropriate to their XML namespace, such as xslt for the XSLT namespace.

Finally, although prefixes are not significant in element and attribute names, they may be significant when used elsewhere in an XML document, such as in a DTD (see question 7.6) or an attribute value. Because such usage is common -- for example, XML Schemas uses qualified names in attribute values -- it is tempting to think that it is defined by the XML namespaces recommendation. This is not the case: the XML namespaces recommendation only defines the use of prefixes in element and attribute names. The use of prefixes elsewhere in an XML document is defined outside the XML namespaces recommendation and is entirely the responsibility of the defining application. For more information, see question 10.4.

11.4) Can I use the same prefix for more than one XML namespace?

Yes.

For example, in the following, B is in the http://www.foo.org/ namespace and C is in the http://www.bar.org/ namespace:

```
<A>
   <foo:B xmlns:foo="http://www.foo.org/">abc</foo:B>
```

```
   <foo:C xmlns:foo="http://www.bar.org/">abc</foo:C>
</A>
```

Remember, however, that the prefix in an XML namespace declaration overrides the same prefix if that prefix is declared on an ancestor. For example, in the following, C is in the http://www.bar.org/ namespace, not the http://www.foo.org/ namespace:

```
<A>
   <foo:B xmlns:foo="http://www.foo.org/">
      <foo:C xmlns:foo="http://www.bar.org/">abc</foo:C>
   </foo:B>
</A>
```

In general, it is a good idea to use a prefix for only one XML namespace, as this reduces confusion when reading the document. In spite of this, there is a very real possibility that the same prefix will be used for multiple XML namespaces when combining fragments from different documents. For more information, see question 4.9.

11.5) Can I use more than one prefix for the same XML namespace?

Yes.

For example, in the following, both B and C are in the http://www.foo.org/ namespace:

```
<A xmlns:foo="http://www.foo.org/"
   xmlns:bar="http://www.foo.org/">
   <foo:B>abc</foo:B>
   <bar:C>abc</bar:C>
</A>
```

In general, it is a good idea to use only one prefix for an XML namespace, as this reduces confusion when reading the document.

11.6) How are prefixes declared?

Prefixes are declared in an XML namespace declaration (xmlns attribute). For more information, see question 4.1.

11.7) Can I undeclare a prefix -- that is, dissociate a prefix from an XML namespace?

No.

You can override a prefix (see question 4.5) or the XML namespace declaration that declares the prefix can go out of scope (see question 6.4), but you can't undeclare the prefix. (Note that you can

undeclare the default XML namespace; for more information, see question 4.8.)

11.8) What happens if I use a prefix that is not declared?

This results in a namespace error. For example, the following document does not conform to the XML namespaces recommendation.

```
<?xml version="1.0" ?>
<foo:A />
```

However, the XML namespaces recommendation does not specify what a namespace-aware application does when it encounters a non-conformant document, so the behavior is application-dependent. For example, the application could stop processing, post an error to a log and continue processing, or ignore the error.

11.9) What happens if there is no prefix on an element type name?

If a default XML namespace declaration is in scope, then the element type name is in the default XML namespace. Otherwise, the element type name is not in any XML namespace. For example, in the following, B and C are in the http://www.foo.org/ namespace and A is not in any XML namespace:

```
<A>
   <B xmlns="http://www.foo.org/">
      <C>abc</C>
   </B>
</A>
```

11.10) What happens if there is no prefix on an attribute name?

The attribute name is not in any XML namespace. That is, the default XML namespace **does not** apply to unprefixed attribute names.

For more information, see myth #4 in "Namespace Myths Exploded".

12) SECTION 12: XML NAMESPACE NAMES (URIs)

12.1) What is an XML namespace URI?

Technically, the term *namespace URI* is never defined in the XML namespaces recommendation. In practice, this is the term commonly used to denote the URI used as an XML namespace name. That is, it is a synonym for the term *namespace name*. Because of the widespread use of this term, we tend to use the term "namespace name (URI)" in this FAQ.

Note: Version 1.1 of the XML namespaces recommendation uses IRIs (Internationalized Resource Identifiers) instead of URIs. However, because version 1.1 is not yet a full recommendation [February, 2003] and because the IRI RFC is not yet complete, this document continues to refer to URIs instead of IRIs.

12.2) What is an XML namespace name?

An XML namespace name is a URI that uniquely identifies the namespace. URIs are used because they are widely understood and well documented. Because people may only allocate URIs under their control, it is easy to ensure that no two XML namespaces are identified by the same URI.

12.3) What does the URI used as an XML namespace name point to?

The URI used as an XML namespace name is simply an identifier. It is not guaranteed to point to anything and, in general, it is a bad idea to assume that it does. This point causes a *lot* of confusion, so we'll repeat it here:

> **URIs USED AS XML NAMESPACE NAMES ARE JUST IDENTIFIERS. THEY ARE NOT GUARANTEED TO POINT TO ANYTHING.**

While this might be confusing when URLs are used as namespace names, it is obvious when other types of URIs are used as namespace names. For example, the following namespace declaration uses an ISBN URN:

```
xmlns:xbe="urn:ISBN:0-7897-2504-5"
```

and the following namespace declaration uses a UUID URN:

```
xmlns:foo="urn:uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"
```

Clearly, neither namespace name points to anything on the Web.

NOTE: Namespace URIs that are URLs may point to RDDL documents, although this does not appear to be widely implemented. For details, see the next question.

NOTE: An early version of the W3C's XML Schemas used namespace URIs to point to an XML Schema document containing the definitions of the element types and attributes named in the namespace. However, this proved very controversial and the idea has been withdrawn.

12.4) Can I resolve the URI used as an XML namespace name?

Yes.

You can also eat a tractor, but that doesn't mean it's a good idea. The URIs used as XML namespace names are not guaranteed to point to anything, so there is generally no reason to resolve them. Furthermore, there is nothing in the processing of XML namespaces that requires you to resolve these URIs. Finally, as was noted in the previous question, many types of namespace URIs are unresolvable on the Web.

That said, some people have advocated placing RDDL (Resource Directory Description Language, pronounced "riddle") documents at the locations pointed to by namespace URIs. A RDDL document "provides a text description of some class of resources and of individual resources related to that class", including such things as a human-readable description of what the namespace describes and links to resources such as schemas (in a variety of languages), stylesheets, and code. RDDL documents apparently are being used, although it is not clear how widely.

For more information about RDDL and the ideas behind it, see:

- Resource Directory Description Language
- RDDL Me This: What Does a Namespace URL Locate?
- Architectural Theses on Namespaces and Namespace Documents

12.5) Can I use a relative URI as a namespace name?

Yes. However, such usage is deprecated, so you should **never** do it.

The original XML namespaces recommendation does not explicitly disallow the use of relative URIs as namespace names. Instead, it states that two namespace names are identical if their URIs match character for character. It then notes that non-identical URIs might be functionally equivalent, such as when they differ only in case. Combined with the URI specification (RFC2396), which states that relative URIs take on the base URI of their containing document, this provided a loophole with which to use relative URIs.

When this was discovered more than a year after the recommendation was released it proved hugely controversial. For example, some people wanted to use relative URIs as namespace names so that elements and attributes would have different XML namespace names (URIs) depending on what document they were in. The objection to this was that it would undermine the purpose of XML namespaces. That is, XML namespaces would no longer provide universally unique names.

After more than two months of discussion, it was decided that relative URIs are not expanded when used as namespace names and that their use was deprecated. That is, suppose that the following document has a URI of "http://www.foo.org/xml/relativeExample.xml".

```
<A xmlns:foo="../relativeURI">
   <foo:B>abc</foo:B>
   <bar:C>abc</bar:C>
</A>
```

Although the *URI* "../relativeURI" does expand to the URI "http://www.foo.org/relativeURI", the *name* of the XML namespace associated with the foo prefix is "../relativeURI", not the expanded URI.

It was also decided that:

- The infoset recommendation does not define an infoset for documents that use relative URIs as XML namespace names.
- The value of the namespaceURI attribute of the Node interface in the DOM is undefined if the URI is relative.
- The return value of the namespace-uri() function in XPath is undefined if the URI is relative.
- All new W3C recommendations should include a statement saying that they do not apply to XML documents that use relative URIs as XML namespace names.

For the archives of the mailing list discussing the use of relative URIs as namespace names, see http://lists.w3.org/Archives/Public/xml-uri/. For the results of the W3C XML Plenary Ballot on the use of relative URIs in XML namespace names, see http://www.w3.org/2000/09/xppa.

PART IV: MISCELLANEOUS

13) SECTION 13: POLITICS AND REVOLUTIONS

13.1) Why are XML namespaces so hard to understand and use?

Actually, they aren't. When trying to understand XML namespaces, all you need to do is remember one thing:

- XML namespaces provide a two-part naming system for element types and attributes.

That's it. They don't do anything else and what little complexity there is comes from the flexibility of declaration and scoping rules. You can minimize this complexity by doing the following:

- Declare all XML namespaces on the root element.
- Use one prefix per XML namespace.

Processing XML namespaces isn't that hard either. In a SAX 1.0 application, you can use John Cowan's Namespace SAX Filter to return universal names; SAX 2.0 does this for you. In a DOM level 1 application, you can usually ask the DOM implementation for namespace information; DOM level 2 always provides this information. If you do your own processing, all you need to remember is to keep the prefix-to-URI mappings as part of your state, scan the attributes for xmlns attributes before doing anything else, and scan element type and attribute names for prefixes.

13.2) Are there any alternatives to XML namespaces?

Yes. You could resolve name clashes by simply renaming the offending element types or attributes and then, during processing, using architectural forms to transform them to names that are recognizeable by different modules in the application. Although you may have to rename some element types or attributes, you won't have to change your application modules. A discussion of architectural forms is beyond the scope of this paper. For more information, see Architectural Forms and SGML/XML Architectures on Robin Cover's XML Cover Pages.

As to the actual syntax used by XML namespaces, the idea of declaring XML namespace prefixes in a processing instruction at the start of an XML document is occasionally raised. Processing instructions were used in an early draft of the XML namespaces specification and have the advantage of simplicity (see question 4.9). They were later replaced with the more flexible xmlns attributes. Although you might be tempted to strike out on your own and use processing instructions or some other strategy, this is probably a bad idea, although not necessarily for technical reasons. The real problem is that XML namespaces are already widely used and supported, so any XML documents you produce will be non-standard and non-portable.

13.3) How controversial are XML namespaces?

The need for XML namespaces and the basic idea that a two-part naming system (or something similar) is needed is not controversial. However, the design of XML namespaces -- that is, the way XML namespaces are declared and used in an XML document, as well the confusion discussed in "Namespace Myths Exploded" -- has, at times, been very controversial. (If you want to see just how controversial, go to the archives of the XML-DEV mailing list and search on the word "namespace".) Although XML namespaces still have some very vocal detractors, most people have accepted and are using them. Furthermore, most new XML tools and technologies use them, a state of affairs that is only likely to increase.

14) SECTION 14: XML NAMESPACE RESOURCES

14.1) What resources are available for learning about XML namespaces?

Recommendations and other official stuff:

- Namespaces in XML (W3C Recommendation): http://www.w3.org/TR/REC-xml-names
- Namespaces in XML 1.1 (W3C Candidate Recommendation): http://www.w3.org/TR/xml-names11/
- Namespaces in XML (auf Deutsch): http://www.schumacher-netz.de/TR/1999/REC-xml-names-19990114-de.html
- Results of W3C XML Plenary Ballot on Relative URI References in Namespace Declarations: http://www.w3.org/2000/09/xppa

Articles about XML namespaces:

- Namespace Myths Exploded, by Ronald Bourret: http://www.xml.com/pub/a/2000/03/08/namespaces/index.html
- XML Namespaces by Example, by Tim Bray: http://www.xml.com/pub/1999/01/namespaces.html
- XML Namespaces, by James Clark: http://www.jclark.com/xml/xmlns.htm
- Plan to use XML Namespaces, Parts 1 and 2, by David Marston: http://www-106.ibm.com/developerworks/xml/library/x-nmspace.html
  http://www-106.ibm.com/developerworks/library/x-nmspace2.html
- 19 Short Questions about Namespaces (with Answers), by David Megginson: http://www.megginson.com/docs/namespaces/namespace-questions.html
- Namespace Nuances, by John E. Simpson: http://www.xml.com/pub/a/2001/07/05/namespaces.html
- Computers : Data Formats : Markup Languages : XML : Namespaces, by the Open Directory Project: http://dir.google.com/Top/Computers/Data_Formats/Markup_Languages/XML/Namespaces/
  A list of articles (including some of those above) about XML namespaces.

More links are welcome.

15) SECTION 15: COMMENTS, COMPLAINTS, AND SUGGESTIONS

If you have comments, complaints, or suggestions for new questions, you can mail them to me (Ronald Bourret) at:

rpbourret@rpbourret.com

Please note that I travel frequently, so it might be two to three weeks before I reply.

Thanks also to Adrian Boyko, Tim Bray, Derek Denny-Brown, Bob DuCharme, Sam Hunting, Madhav Lakkapragada, Andrew Layman, Gustaf Liljegren, Christopher Lott, Marc McDonald, Anders Møller, Jim Palmer, Ganesan Radhakrishnan, Arjun Ray, Aron Roberts, and Richard Tobin for their input.

XML-Data

W3C Note 05 Jan 1998

This version:
http://www.w3.org/TR/1998/NOTE-XML-data-0105
Latest version:
http://www.w3.org/TR/1998/NOTE-XML-data
Authors:
Andrew Layman, Microsoft Corporation
Edward Jung, Microsoft Corporation
Eve Maler, ArborText
Henry S. Thompson, University of Edinburgh
Jean Paoli, Microsoft Corporation
John Tigue, DataChannel
Norbert H. Mikula, DataChannel
Steve De Rose, Inso Corporation

Status of this Document

This document is a NOTE made available by the World Wide Web Consortium for discussion only. This indicates no endorsement of its content, nor that the Consortium has, is, or will be allocating any resources to the issues addressed by the NOTE. A list of current NOTEs can be found at: http://www.w3.org/TR/.

This document is a submission to the W3C. Please see Acknowledged W3C Submissions regarding its disposition.

Contents:

Acknowledgements

We thank Paul Grosso(ArborText), Sharon Adler (Inso Corporation), Anders Berglund (Inso Corporation), François Chahuneau (AIS/Berger-Levrault)for their help and contributions to this proposal.

Introduction

Schemas define the characteristics of classes of objects. This paper describes an XML vocabulary for schemas, that is, for defining and documenting object classes. It can be used for classes which as strictly syntactic (for example, XML) or those which indicate concepts and relations among concepts (as used in relational databases, KR graphs and RDF). The former are called "syntactic schemas;" the latter "conceptual schemas."

For example, an XML document might contain a "book" element which lexically contains an "author" element and a "title" element. An XML-Data schema can describe such syntax. However, in another context, we may simply want to represent more abstractly that books have titles and authors, irrespective of any syntax. XML-Data schemas can describe such conceptual relationships. Further, the information about books, titles and authors might be stored in a relational database, in which XML-Data schemas describe row types and key

relationships.

One immediate implication of the ideas in this paper is that XML document types can now be described using XML itself, rather than DTD systax. Another is that XML-Data schemas provide a common vocabulary for ideas which overlap between syntactic, database and conceptual schemas. All features can be used together as appropriate.

Schemas are composed principally of declarations for:

- Concepts
- Classes of objects
  - Class hierarchies
  - Properties
  - Constraints
  - Relationships
  - Indicated by primary key to foreign key matching
  - Indicated by URI
- XML DTD Grammars and Compatibility
  - grammatical rules governing the valid nesting of the elements and attributes
  - attributes of elements
  - internal and external entities, represented by intEntityDecl and extEntityDecl
  - notations, represented by notationDcl
- Datatypes giving parsing rules and implementation formats.
- Mapping rules allowing abbreviated grammars to map to a conceptual data model.

The Schema Element Type

All schema declarations are contained within a schema element, like this:

```
<?XML version='1.0' ?>
<?xml:namespace name="urn:uuid:BDC6E3F0-6DA3-11d1-A2A3-
00AA00C14882/" as="s"/?>
<s:schema id='ExampleSchema'>
  <!-- schema goes here. -->
</s:schema>
```

The namespace of the vocabulary described in this document is named "urn:uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882/".

The *ElementType* Declaration

The heart of an XML-Data schema is the *elementType* declaration, which defines a class of objects (or "type of element" in XML terminology). The *id* attribute serves a dual role of identifying the definition, and also naming the specific class.

```
<elementType id="author"/>
```

Within an elementType, the *description* subelement may be used to provide a human-readable description of the element's purpose.

```
<elementType id="author">
  <description>The person, natural or otherwise, who wrote the
book.</description>
</elementType >
```

Properties and Content Models

Subelements within *elementType* define characteristics of the class's members. An XML "content model" is a description of the contents that may validly appear within a particular element type in a document instance.

```
<elementType id="author">
    <string/>
</elementType>

<elementType id="Book">
    <element type="#author" occurs="ONEORMORE"/>
</elementType>
```

The example above defines two elements, author and book, and says that a book has one or more authors. The author element may contain a string of character data (but no other elements). For example, the following is valid:

```
<Book>
    <author>Henry Ford</author>
    <author>Samuel Crowther</author>
</Book>
```

Within an elementType, various specialized subelements (element, group, any, empty, string etc.) indicate which subelements (properties) are allowed/required. Ordinarily, these imply not only the cardinality of the subelements but also their sequence. (We discuss a means to relax sequence later.)

Element

Element indicates the containment of a single element type (property). Each *element* contains an *href* attribute referencing another *elementType*, thereby including it in the content model syntacticly, or declaring it to be a property of the object class conceptually. The element may be required or optional, and may occur multiple times, as indicated by its *occurs* attribute having one of the four values "REQUIRED", "OPTIONAL", "ZEROORMORE" or "ONEORMORE". It has a default of "REQUIRED".

```
<elementType id="Book">
    <element type="#title" occurs="OPTIONAL"/>
```

```
    <element type="#author" occurs="ONEORMORE"/>
</elementType>
```

The example above describes a book element type. Here, each instance of a book *may* contain a title, and *must* contain one or more authors.

```
<Book>
    <author>Henry Ford</author>
    <author>Samuel Crowther</author>
    <title>My Life and Work</title>
</Book>
```

When we discuss type hierarchies, later, we will see that an element type may have subtypes. If so, inclusion of an element type in a content model permits elements of that type directly and all its subtypes.

Empty, Any, String, and Mixed Content

Empty and *any* content are expressed using predefined elements *empty* and *any*. (*Empty* may be omitted.) *String* means any character string not containing elements, known as "PCDATA" in XML. *Any* signals that any mixture of subelements is legal, but no free characters. *Mixed* content (a mixture of parsed character data and one or more elements) is identified by a *mixed* element, whose content identifies the element types allowed in addition to parsed character data. When the content model is mixed, any number of the listed elements are allowed, in any order.

```
<?XML version='1.0' ?>
<?xml:namespace name="urn:uuid:BDC6E3F0-6DA3-11d1-A2A3-
00AA00C14882/" as="s"/?>
<s:schema>

    <elementType id="name">
        <string/>
    </elementType>

    <elementType id="Person">
        <any/>
    </elementType>

    <elementType id="author">
        <string/>
    </elementType>

    <elementType id="titlePart">
        <string/>
    </elementType>

    <elementType id="title">
        <mixed><element type="#titlePart"/></mixed>
    </elementType>

    <elementType id="Book">
```

```
        <element type="#title" occurs="OPTIONAL"/>
        <element type="#author" occurs="ONEORMORE"/>
    </elementType>

</s:schema>

...

<Book>
    <author>Henry Ford</author>
    <author>Samuel Crowther</author>
    <title>My Life and<titlePart>Work</titlePart></title>
</Book>
```

Here, *book* is defined to have an optional *title* and one or more *authors.* The *name* element has content model of *any*, meaning that free text is not allowed, but any arrangement of subelements is valid. The *content model* of *title* is *mixed*, allowing a free intermixture of characters and any number of *titleParts*. The author, *name* and *titleParts* elements have a *content model* of *string*.

Group

Group indicates a set or sequence of elements, allowing alternatives or ordering among the elements by use of the groupOrder attribute. The group as a whole is treated similarly to an element.

```
<elementType id="Book">
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
    <group occurs="OPTIONAL">
        <element type="#preface"/>
        <element type="#introduction"/>
    </group>
</elementType>
```

In the above example, if a preface or introduction appears, both must, with the preface preceding the introduction. Each of the following is valid:

```
<Book>
    <author>Henry Ford</author>
</Book>

<Book>
    <author>Henry Ford</author>
    <preface>Prefatory text</preface>
    <introduction>This is a swell book.</introduction>
</Book>
```

Sometimes a schema designer wants to relax the ordering restrictions among elements, allowing them to appear in any order. This is indicated by setting the groupOrder attribute to "AND":

```
<elementType id="Book">
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
    <group groupOrder="AND" occurs="OPTIONAL">
        <element type="#preface"/>
        <element type="#introduction"/>
    </group>
</elementType>
```

Now the following is also valid:

```
<Book>
    <author>Henry Ford</author>
    <introduction>This is a swell book.</introduction>
    <preface>Prefatory text</preface>
</Book>
```

Finally, a schema can indicate that any one of a list of elements (or groups) is needed. For example, either a preface *or* an introduction. The groupOrder attribute value "OR" signals this.

```
<elementType id="Book">
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
    <group groupOrder="OR">
        <element type="#preface"/>
        <element type="#introduction"/>
    </group>
</elementType>
```

Now each of the following is valid:

```
<Book>
    <author>Henry Ford</author>
    <preface>Prefatory text</preface>
</Book>

<Book>
    <author>Henry Ford</author>
    <introduction>This is a swell book.</introduction>
</Book>
```

Open and Closed Content Models

XML typically does not allow an element to contain content unless that content was listed in the model. This is useful in some cases, but overly in others in which we would like the listed content model to govern the cardinality and other aspects of whichever subelements are explicitly named, while allowing that other subelements can appear in instances as well.

The distinction is effected by the *content* attribute taking the values "OPEN" and "CLOSED." The default is

"OPEN" meaning that all element types not explicitly listed are valid, without order restrictions. (This idea has a close relation to the Java concept of a final class.)

For example, the following instance data for a book, including the unmentioned element *copyrightDate* would be valid given the content models declared so far, because they have all been *open*.

```
<Book>
    <author>Henry Ford</author>
    <author>Samuel Crowther</author>
    <title>My Life and Work</title>
    <copyrightDate>1922</copyrightDate>
</Book>
```

However, had the content model been declared closed, as follows, the *copyrightDate* element would be invalid.

```
<elementType id="Book" content="CLOSED">
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
    <group groupOrder="SEQ" occurs="OPTIONAL">
        <element type="#preface"/>
        <element type="#introduction" occurs="REQUIRED"/>
    </group>
</elementType>
```

A closed content model does not allow instances to contain any elements or attributes beyond those explicitly listed in the elementType declaration.

Default Values

An element with occurs of REQUIRED or OPTIONAL (but not ONEORMORE or ZEROORMORE) can have a default value specified.

```
<elementType id="Book">
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
    <element type="#ageGrp" occurs="OPTIONAL">
        <default>adult</default>
    </element>
</elementType>
```

The default value is implied for all element *instances* in which it is syntactically omitted.

To indicate that the default value is the only allowed value, the *presence* attribute is set to "FIXED".

```
<elementType id="Book">
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
```

```
    <element type="#ageGrp" occurs="OPTIONAL" presence="FIXED">
        <default>ADULT</default>
    </element>
</elementType>
```

Presence has values of "IMPLIED," "SPECIFIED," "REQUIRED," and "FIXED" with the same meanings as defined in XML DTD.

Aliases and Correlatives

ElementTypes can be know be different names in different languages or domains. The equivalence of several names is effected by the sameAs attribute, as in

```
<elementTypeEquivalent id="livre" type="#Book"/>
<elementTypeEquivalent id="auteur" type="#author"/>
```

Elements are used to represent both primary object types (nouns) and also properties, relations and so forth. Relations are often known by two names, each reflecting one direction of the relationship. For example, husband and wife, above and below, earlier and later, etc. The *correlative* element identifies such a pairing.

```
<elementType id= "author">
    <string/>
</elementType>

<elementType id= "wrote">
    <correlative type="#author" />
    <string/>
</elementType>
```

This indicates that "wrote" is another name for the "author" relation, but from the perspective of the person, not the book. That is, the two fragments below express the same fact:

```
<Person>
    <name>Henry Ford</name>
</Person>

<Book>
    <title>My Life and Work</title>
    <author>Henry Ford</author>
</Book>

...

<Person>
    <name>Henry Ford</name>
    <wrote>My Life and Work</wrote>
</Person>
```

```
<Book>
    <title>My Life and Work</title>
</Book>
```

A correlative may be defined simply to document the alternative name for the relation. However, it may also be used within a content model where it permits instances to use the alternative name. Further it may to establish constraints on the relation, indicate key relationships, etc.

Class Hierarchies

ElementTypes can be organized into categories using the *superType* attribute, as in

```
<elementType id="price">
    <string/>
</elementType>

<elementType id="ThingsI’veBoughtRecently">
    <element type="#price"/>
</elementType>

<elementType id="PencilsI’veBoughtRecently">
    <superType type="#ThingsI’veBoughtRecently"/>
    <element type="#price"/>
</elementType>

<elementType id="BooksI’veBoughtRecently">
    <superType type="#ThingsI’veBoughtRecently"/>
    <element type="#price"/>
</elementType>
```

This simply indicates that, in some fashion, *PencilsI’veBoughtRecently* and *BooksI’veBoughtRecently* are subsets of *ThingsI’veBoughtRecently*. It implies that every valid instance of the subset is a valid instance of the superset. The superset type must have an *open* content model.

There are restrictions that should be followed, based on the principle that all instances of the species (subtype) must be instances of the genus (supertype):

- The genus type must have content="OPEN".
- It must have either no groups or only groups with groupOrder="AND" (that is, no order constraints).
- You can add new elements and attributes.
- Occurs cardinality can be decreased but not increased.
- Ranges and other constraints are cummulative, that is, all apply (though the exact effect of this depends on the semantics of the constraint type).
- Default values can be made FIXED defaults.

To indicate that the content model of the subset should inherit the content model of a superset, we use a

particular kind of superType called "genus" of which only one is allowed per ElementType. This copies the content model of the referenced element type and permits addition of new elements to it. Further, sub-elements occurring in the superset type, if declared again, are replaced by the newer declarations.

```
<elementType id="Book">
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
</elementType>

<elementType id="BooksI'veBoughtRecently">
    <genus type="#Book"/>
    <superType type="#ThingsI'veBoughtRecently"/>
    <element type="#price"/>
</elementType>
```

The above has the same effect as

```
<elementType id="Book">
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
</elementType>

<elementType id="BooksI'veBoughtRecently">
    <superType type="#Book"/>
    <superType type="#ThingsI'veBoughtRecently"/>
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
    <element type="#price"/>
</elementType>
```

Elements which are References

ElementTypes and the content model elements defined so far are sufficient to declare a tree structure of elements. However, some elements such as "author" are *not only* usable on their own, they also act as references to other elements. For example, "Henry Ford" is the value of the *author* subelement of a *book* element. "Henry Ford" is also the value of the *name* element in a *person* element, and it can be used to connect these two.

```
<Book>
    <author>Henry Ford</author>
    <author>Samuel Crowther</author>
    <title>My Life and Work</title>
</Book>

<Person><name>Henry Ford</name></Person>

<Person><name>Samuel Crowther</name></Person>
```

In this capacity, such subelement are often referred to as *relations* when using "knowledge representation" terminology or "keys" when using database terms. (The meaning of "relation" and "key" are slightly different, but the fact which the terms recognize is the same.)

To make such references explicit in the schema, we add declarations for *keys* and *foreign keys.*

```
<elementType id="name">
    <string/>
</elementType>

<elementType id="Person">
    <element id="p1" type="#name"/>
    <key id="k1"><keyPart href="#p1"/></key>
</elementType>

<elementType id="author">
    <string/>
    <foreignKey range="#Person" key="#k1"/>
</elementType>

<elementType id="title">
    <string/>
</elementType>

<elementType id="Book">
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
</elementType>
```

The *key* element within *person* tells us that a person can be uniquely identified by his *name*. The *foreignKey* element within the *author* element definition says that the contents of an author element are a foreign key indentifying a person by *name*.

An uninformed user agent can still display the string "Henry Ford" even if it cannot determine that is supposed to be a person. A savvy agent that reads the schema can do more. It can locate the actual person.

This is the information needed for a *join* in database terminology.

This mechanism not only handles the typical way in which properties are expressed in databases, it also handles all cases in which the contents of an element are to be interpreted as strings from a restricted vocabulary, such as enumerations, XML nmtokens, etc.

```
<Book>
    <author>Henry Ford</author>
    <author>Samuel Crowther</author>
    <title>My Life and Work</title>
    <lccn>HD9710.U54 F58 1973</lccn>
    <dewey>629.2/092/4 B</dewey >
```

```
    <isbn>0405050887</isbn>
    <series>Business<series>
</Book>
```

Although not shown here, presumably *lccn*, *dewey* and *isbn* are declared in the schema to be foreign keys to corrresponding fields of catalog records. *Series* is a foreign key to a categorization of books, of which "Business" is one category.

Keys can contain URIs, as in

```
<Book>
    <author>http://SSA.gov/blab/people/Henry+Ford</author>
    <author>http://SSA.gov/blab/people/Samuel+Crowther</author>
    <title>My Life and Work</title>
</Book>
```

This is indicated in the schema by a datatype of "URI".

```
<elementType id="author">
    <string/>
    <datatype dt="uri"/>
</elementType>
```

One-to-Many Relations

Element relations are binary. That is, we never express an n-to-1 relationship directly. We do not, for example, list within *books* a single relation that somehow resolves to all the *authors*. Instead, we always write the relationship on the 1-to-n side, but allow multiple occurrances of the *subelement*, for example, allowing *books* to have multiple occurrences of *author*.

```
<Person><name>Henry Ford</name></Person>

<Person><name>Samuel Crowther</name></Person>

<Person><name>Harvey S. Firestone</name></Person>

<Book>
    <author>Henry Ford</author>
    <author>Samuel Crowther</author>
    <title>My Life and Work</title>
</Book>

<Book>
    <author>Harvey S. Firestone</author>
    <author>Samuel Crowther</author>
    <title>Men and Rubber</title>
</Book>
```

This example shows a book with several persons as author, and also a person who is author of several books.

We discussed such many-to-many relations more under the topic of *correlations*.

Multipart Keys

When the foreignKey element does not have foreignKeyPart sub-elements (as it does not above) then the entirety of the element's contents (e.g. "Henry Ford") should be used as the key value. However, for multipart foreign keys, or cases where the element has several sub-elements, foreignKeyPart is used, as shown below.

```
<elementType id="firstName">
    <string/>
</elementType>

<elementType id="lastName">
    <string/>
</elementType>

<elementType id="Person">
    <element id="pp1" type="#firstName"/>
    <element id="pp2" type="#lastName"/>
    <key id="k1">
        <keyPart href="#pp1"/>
        <keyPart href="#pp2"/>
    </key>
</elementType>

<elementType id="author">
    <element id="ap1" type="#firstName"/>
    <element id="ap2" type="#lastName"/>
    <domain type="#Book"/>
    <range type="#Person"/>
    <foreignKey range="#Person" key="#k1">
        <foreignKeyPart href="#ap1"/>
        <foreignKeyPart href="#ap2"/>
    </foreignKey>
</elementType>

...

<Book>
    <title>My Life and Work</title>
    <author>
        <firstName>Henry</firstName>
        <lastName>Ford</lastName>
    </author>
</Book>
```

Attributes as References

An alternative way to express a reference is with an attribute.

```
<person id="person1"><name>Henry Ford</name></Person>

<person id="person2"><name>Samuel Crowther</name></Person>

<Book>
    <author name="Henry Ford"/>
    <author name="Samuel Crowther"/>
    <title>My Life and Work</title>
</Book>
```

This allows us to link a book to a person, through the author relation, using an attribute of the relation. This exactly parallels the construction we saw above under "multipart keys," where a *subelement* of author contained the author's name. Here, an *attribute* of author contains the name. We can express this in our schema as

```
<elementType id="author">
    <attribute name="name" id="authorname"/>
    <foreignKey range="#Person" key="#k1">
        <foreignKeyPart href="#authorname"/>
   </foreignKey>
</elementType>
```

A widely-used variant of this is to use a URI as a foreign key:

```
<Book>
    <author href="http://SSA.gov/blab/people/Henry+Ford"/>
    <author href="http://SSA.gov/blab/people/Samuel+Crowther"/>
    <title>My Life and Work</title>
</Book>
```

In this case, we are using the *href* attribute to contain a URI. This is a particular kind of foreign key, where the *range* is any possible resource, and where that resource is not identified by some combination of its properties but instead by a name-resolution service. We indicate this by using an attribute element, with dt= "URI".

```
<elementType id="author">
    <attribute name="href" id="authorhref" dt="uri"/>
</elementType>
```

Constraints & Additional Properties

Min and Max Constraints

Elements can be limited to restricted ranges of values. The *min* and *max* elements define the lower and upper bounds.

```
<elementType id="age">
    <string/>
</elementType>

<elementType id="Person">
    <element hef="#age"><min>0</min><max>131</max></element>
</elementType>
```

Such intervals are *half-open* (that is, the *min* value is in the interval, and the *max* value is the smallest value not in the interval).

This rule leads to the simplest calculation in most cases, and is unambiguous with respect to precision. In the above example, it is clear by these rules the 130.9999 is in the interval and 131 is not. However, had we said "all numbers from 0 to 130.99," in practice we would have some ambiguity regarding the status of 130.9999. Or interpretation would depend on the precision that we inferred for the original statement. The issue is particularly ambiguous for dates. (What exactly does "From December 5 to December 8" mean? The use of half-open intervals for representation does not, however, put any requirements on how processors must display intervals. For example, dates in some contexts display differently than their storage. That is, the interval `<min>1997-12-05</min><max>1997-12-09</max>` might be displayed as "December 5 through December 8".

In certain cases this rule for a half-open interval is impractical (for example, what letter follows "z" in the latin alphabet?) If so, use *maxInclusive*:

```
<elementType id="student">
    <element type="#grade"><min>A</min><maxInclusive>Z</
maxInclusive></element>
</elementType>
```

Domain and Range Constraints

We can use the *domain* and *range* elements to add constraints to an element's use or value. The *domain* element, if present, indicates that the element may only be used as a property of certain other elements. That is, syntactically it may appear only in the content model of those other element types. It constrains the sorts of schemas that can be written with the element.

```
<elementType id="author">
    <string/>
    <domain type="#Book"/>
    <attribute name="href" dt="uri"/>
</elementType>
```

The *domain* property above permits *author* elements to be used only within elements which are either *books* or subsets of *books*. Use of domain is optional. If omitted, there is simply no restriction.

The *range* element is used with elements which are references and declares a restriction on the types of elements to which the relation may refer. Graphically, it describes the target end of a directed edge. Each *range* element references one elementType, any of which are valid. In this case, below, we have said that an *author* element must have an *href* attribute which is a URI reference to a *Person* or to an element type which is *Person*

or a subset of *Person*.

```
<elementType id="author">
    <string/>
    <domain type="#Book"/>
    <attribute name="href" dt="uri" range="#Person" />
</elementType>
```

Other useful properties

Element and attribute types can have an unlimited amount of further information added to them in the schema due to the open nature of XML with namespaces.

Using Elements from Other Schemas

A schema may use elements and attributes from other schemas in content models. For example, a subelement named "http://books.org/date" could be used within a *book* element as follows:

```
<?XML version='1.0' ?>
<?xml:namespace name="urn:uuid:BDC6E3F0-6DA3-11d1-A2A3-
00AA00C14882/" as="s"/?>
<s:schema>
    <elementType id="author">
        <string/>
    </elementType>

    <elementType id="title">
        <string/>
    </elementType>

    <elementType id="Book">
        <element type="#title" occurs="OPTIONAL"/>
        <element type="#author" occurs="ONEORMORE"/>
        <element href="http://books.org/date" />
    </elementType>
</s:schema>
```

This can be abbreviated by adopting the rule that namespace-qualified names may be used within the *href attribute* value of an *element* or *attribute* element.

```
<?XML version='1.0' ?>
<?xml:namespace name="urn:uuid:BDC6E3F0-6DA3-11d1-A2A3-
00AA00C14882/" as="s"/?>
<?xml:namespace name=" http://books.org/" as="bk"/?>
<s:schema>
    <elementType id="author">
        <string/>
```

```
        </elementType>

        <elementType id="title">
            <string/>
        </elementType>

        <elementType id="Book">
            <element type="#title" occurs="OPTIONAL"/>
            <element type="#author" occurs="ONEORMORE"/>
            <element href="bk:date" />
        </elementType>
    </s:schema>
```

XML-Specific Elements

Attributes

XML-Data schemas contain a number of facilities to match features of XML DTDs or to support certain characteristics of XML. The XML syntax allows that certain properties can be expressed in a form called "attributes." To support this, an elementType can contain attribute declarations, which are divided into attributes with enumerated or notation values, and all other kinds.

An attribute may be given a default value. Whether it is required or optional is signaled by *presence*. (Presence ordinarily defaults to IMPLIED, but if omitted and there is an explicit default, presence is set to the SPECIFIED.) See the DTD at the end of this document for syntactic details.

Attributes with enumerated (and notation) values permit a values attribute, a space-separated list of legal values. The values attribute is required when the atttype is ENUMERATION or NOTATION, else it is forbidden. In these cases, if a default is specified, it must be one of the specified values.

```
    <elementType id="Book">
        <element type="#title"/>
        <element type="#author" occurs="ONEORMORE"/>
        <attribute name="copyright" />
        <attribute name="ageGrp" atttype="ENUMERATION" values="child
    adult" default="adult" />
    </elementType>
```

describes an instance such as

```
    <book copyright="1922" ageGrp="adult">
        <title>My Life and Work</title>
        <author>
            <firstName>Henry</firstName>
            <lastName>Ford</lastName>
        </author>
    </Book>
```

Attributes may also reference elementTypes, meaning that one may use the element type but with attribute syntax. This allows an attribute to explicitly have the same name and semantics even when used on different element types. There are of course some limits: The attribute can still occur only once in an instance, and it cannot contain other elements. However, this allows the semantics of the element type to be employed in attribute syntax.

```
<elementType id="Book">
    <attribute href="bk:title"/>
    <attribute href="bk:author"/>
    <attribute name="copyright" />
    <attribute name="ageGrp" type="ENUMERATION" values="children
adult" default="adult" />
</elementType>
```

describes an instance such as

```
<book  bk:author="Henry Ford"  bk:title="My Life and Work"
ageGrp="adult"/>
```

The internal and external entity declaration element type: intEntityDcl and extEntityDcl

This and the next two declarations cover entities. Entities are a shorthand mechanism, similar to macros in a programming language.

```
<intEntityDcl name="LTG">
    Language Technology Group
</intEntityDcl>

<extEntityDcl name="dilbert" notation="#gif"
              systemId="http://www.ltg.ed.ac.uk/~ht/dilb.gif"/>
```

Here as elsewhere, following XML, systemId must be a URI, absolute or relative, and publicId, if present, must be a Public Identifier (as defined in ISO/IEC 9070:1991, Information technology -- SGML support facilities -- Registration procedures for public text owner identifiers). If a notation is given, it must be declared (see below) and the entity will be treated as binary, i.e., not substituted directly in place of references.

```
<notationDcl name="gif" systemId='http://who.knows.where/'/>
```

The external declarations element type: extDcls

Although we allow an external entity with declarations to be included, we recommend a different declaration for schema modularization. The extDcls declaration gives a clean mechanism for importing (fragments of)

other schemas. It replaces the common SGML idiom of declaring an external parameter entity and then immediately referring to it, and has the same import, namely, that the text referred to by the combination of systemId and publicId is included in the schema in place of the extDcls element, and that replacement text is then subject to the same validity constraints and interpretation as the rest of the schema.

Note that in many cases the desired effect may be better represented by referencing elements (and attributes) from the other schema or subclassing from them.

Datatypes

A dataype indicates that the contents of an element can be interpreted as both a string and also, more specifically, as an object that can be interpreteted more specifically as a number, date, etc. The datatype indicates that the element’s contents can be parsed or interpreted to yeild an object more specific than a string.

That is, we distinguish the "type" of an element from its "datatype." The former gives the semantic meaning of an element, such as "birthday" indicating the date on which someone was born. The "datatype" represents the parser class needed to decode the element's contents into an object type more specific than "string." For example, "19541022" is the 22nd of October, 1954 in ISO 8601 date format. (That is, ISO 8601 parsing rules will decode "19541022" into a date, which can then be stored as a date rather than a string.

For example, we would like an XML author to be able to say that the contents of a "size" element is an integer, meaning that it should be parsed according to numeric parsing rules and that it can be stored in integer format. In some contexts an API can expose it as an integer rather than a string.

```
<item>
    <name>shirt</name>
    <size>8</size>
</item>
```

There are two main contexts for datatypes. First, when dealing with database APIs, such as ODBC, all elements with the same name typically contain the same type of contents. For example, all sizes contain integers or all birthdays contain dates. We will return to this case shortly.

Second, and by contrast, the type of the content may vary widely from instance to instance. The softer we make our software, the more often these flexible cases occur. For example, size could contain the integer 8, or the word "small" or even a formula for computing the size.

We expose the datatype of an element instance by use of a *dt:dt* attribute, where the value of the attribute is a URI giving the datatype. (The URI might be explicitly in URI format or might rely on the XML namespace facility for resolution.) For example, we might find a document containing something like:

```
<?namespace name="urn:uuid:C2F41010-65B3-11d1-A29F-
```

```
00AA00C14882/" as="dt"?>
<?namespace name="http://zoosports.com/dt?" as="zoo"?>
<purchases>
  <item>
    <name>shirt</name>
    <size dt:dt="int">8</size>
  </item>
  <item>
    <name>shoes</name>
    <size>large</size>
  </item>
  <item>
    <name>suit</name>
    <size dt:dt="zoo:script">
        =(shirtsize*1.05) + 3
    </size>
  </item>
</purchases>
```

Clearly this technique works for the heterogeneous typing in the above example. It also works for the database case where all element's of the same type have the same datatype.

```
<item> <name>shirt</name> <size dt:dt="int">8</size>   </item>
<item> <name>shoes</name> <size dt:dt="int">6</size>   </item>
<item> <name>suit</name>  <size dt:dt="int">12</size>  </item>
```

As written above, this is inefficient. Fortunately, XML allows us in schemasto put attributes with default or fixed values, so we could say once that all *size* elements have a datatype with value "int". Having done so, our our instance just looks like:

```
<item> <name>shirt</name> <size>14</size>  </item>
<item> <name>shoes</name> <size>6</size>   </item>
<item> <name>suit</name>  <size>16</size>  </item>
```

In a DTD, we can set a *fixed* attribute value, so that all *size* elements have datatype "int" or we can set it as a *default* attribute value so that it is an integer except where explicitly noted otherwise.

```
<item> <name>shirt</name> <size>14</size>  </item>
<item> <name>shoes</name> <size dt:dt="string">large</size>   </
item>
<item> <name>suit</name>  <size>16</size>  </item>
```

XML DTDs today allow such attributes. For example, a DTD can say that all *shirt* elements have *integer datatype* by the following:

```
<!ELEMENT size PCDATA >
<!ATTLIST size   dt:dt "int" #FIXED >
```

XML-Data schemas allow the equivalent, though with specialized syntax:

```
<elementType id="size" >
    <datatype dt="int" />
</elementType>
```

Elements use *datatype* subelements to give the datatype so that an optional *presence* attribute of the datatype element can indicate whether the datatype is *fixed* or merely a *default*. Attributes can also have datatypes. Because there is no possibility of their being anything other than a fixed type, the datatype of an attribute is signalled by a dt attribute:

```
<attribute id="size" dt="int" />
```

How Typed Data is Exposed in the API

Different APIs to typed data will use the datatype attribute differently. The basic XML parser API should expose all element contents as strings regardless of any datatype attribute. (It might also contain supplementary methods to read values as more specific types such as "integer," thereby getting more efficiency.) An ODBC interface could use the datatype attribute to expose each type of element as a column, with the column's datatype determined by the element type's datatype.

Complex Data Types

If a datatype requires a complex structure for storage, or an object-based storage, this is also handled by the dt: dt attribute, where the datatype's storage format can be a structure, Java class, COM++ class, etc. For example, if an application needed to have an element stored in a "ScheduleItem" structure and using some private format, it could note this like

```
<when dt:dt="zoo:ScheduleItem">M*D1W4B19971022;100</when>
```

The datatype does not require a private format. It could also use subelements and attributes such as

```
<when dt:dt="zoo:ScheduleItem2">
    <month>*</month>
    <day>1</day>
    <week>4</week>
    <begin>19971022</begin>
    <recurs>100</recurs>
</when>
```

In the case of the graph-oriented interfaces (e.g. XML/RDF) the mapping from the XML tree to a graph should add a *wrapping node* for each non-string data type. The datatype property gives the type of that node. For example, the following two are graphically equivalent:

```
<size dt:dt="int">8</size>
<size><dt:int>8</dt:int></size>
```

Versioning of Instances

Adding an attribute to an element does not change the other attributes or pose any special versioning problems.

For example, an application written to expect an instance to contain "<birthday>19541022</birthday>" is not harmed if the schema reveals that this is ISO 8601 format. Versioning within datatypes should be handled by the author's making sure that that subtypes of datatypes retain all the characteristics of the supertype.

If a down-level application is given a datatype it cannot process, it should expose the element contents as a supertype of the indicated datatype. In practice, this will usually mean that unrecognized datatypes will be the same as "dt:string". However, there are cases in which a type will be promoted, for example exposing a boolean in a byte or word rather than a bit, exposing a floating point number in a language's native format, etc.

The Datatypes Namespace

The datatype attribute "dt" is defined in the namespace named "urn:uuid:C2F41010-65B3-11d1-A29F-00AA00C14882/". (See the XML Namespaces Note at the W3C site for details of namespaces.) The full URN of the attribute is "urn:uuid:C2F41010-65B3-11d1-A29F-00AA00C14882/dt".

You will have noticed that the value of the attribute, as used in the examples above, is not lexically a full URI. For example, it reads "int" or "string" etc. Datatype attribute values are abbreviated according to the following rule: If it does not contain a colon, it is a datatype defined in the datatypes namespace "urn:uuid:C2F41010-65B3-11d1-A29F-00AA00C14882/". If it contains a colon, it is to be expanded to a full URI according to the same rules used for other names, as defined by the XML Namespaces Note. For example

```
<?namespace name="urn:uuid:C2F41010-65B3-11d1-A29F-
00AA00C14882/" as="dt"?>
<?namespace name="http://zoosports.com/dt?" as="zoo"?>
<item>
    <size dt:dt="int">8</size>
    <name dt:dt="zoo:clothing">shirt</name>
</item>
```

has two datatypes whose full names are "urn:uuid:C2F41010-65B3-11d1-A29F-00AA00C14882/integer" and "http://zoosports.com/dt?clothing".

What a datatype's URI Means

Datatypes are identified by URIs. The URI as simply a reference to a section of a document that defines the appropriate parser and storage format of the element. To make this broadly useful, this document defines a set of common data types including all common forms of dates, plus all basic datatypes commonly used in SQL, C, C++, Java and COM (including strings).

The best form of such a document is that it should itself be an XML-Data schema where each datatype is an element declaration. For this purpose we define a *<Syntax>* subelement which can be used in lieu of a content model. We also define an *<objecttype>* subelement. Each has a URI as its value. This integrates data types with element types in general.

```
<schema:elementType id="int">
  <syntax  href="urn:uuid:C2F41010-65B3-11d1-A29F-00AA00C14882/
num_to_int" />
  <objectType href="urn:uuid:C2F41010-65B3-11d1-A29F-
00AA00C14882/integer32" />
```

```
</schema:elementType>

<schema:elementType id="date.iso8601">
  <syntax  href="urn:uuid:C2F41010-65B3-11d1-A29F-00AA00C14882/
date.iso8601_to_int32" />
  <objecttype href="urn:uuid:C2F41010-65B3-11d1-A29F-
00AA00C14882/integer32" />
</schema:elementType>
```

The objecttype sub-element can reference a structure, Java class, COM++ coClass, etc. The syntax subelement identifies a parser which can decode the element's content (and/or attributes) into the object type given the storage type URI. Input to the parser is the element object exposing all its attributes and content tree (that is, the subtree of the grove beginning with the element containing the dt attribute). The objectType attribute in particular is assumed available to the parser so that a single parser can support several objecttypes.

Having said this, all *basic* data types should be built into the parsers for efficiency and in order to ground the process. For these, the datatype elements serve only to formally document the storage types and parsers, and to give higher-level systems (such as RDF) a more formal basis for datatypes.

I do not currently propose that we attempt to write any universal notation for parsing rules. Certain popular kinds of formats, particularly dates, are not easily expressed in anything but natural language or code, and the parsers must be custom written code. In other words, the URIs for the basic syntax and objecttype elements probably resolve only to text descriptions.

Structured Data Type Attributes

Attributes in cannot XML have structure. I will separately propose some techniques to avoid this problem, specifically that the XML API should contain a method that treats attributes and subelements indistinguishably, and also that the content which is an element's value can be syntactically separated from content which is an element's properties.

Specific Datatypes

This includes all highly-popular types and all the built-in types of popular database and programming languages and systems such as SQL, Visual Basic, C, C++ and Java(tm).

Name	Parse type	Storage type	Examples
`string`	pcdata	string (Unicode)	Ομωνυμα λεγαται ων ονομα μονον κοινον, ο δε κατα τουνομα λογοσ της ουσιας ετεροσ, οιον ζυον ο τε ανθροπος και το γεγραμμενον.

`number`	A number, with no limit on digits, may potentially have a leading sign, fractional digits, and optionally an exponent. Punctuation as in US English.	string	15, 3.14, -123.456E+10
`int`	A number, with optional sign, no fractions, no exponent.	32-bit signed binary	1, 58502, -13
`float`	Same as for "number."	64-bit IEEE 488	.314159265358979E+1
`fixed.14.4`	Same as "number" but no more than 14 dights to the left of the decimal point, and no more than 4 to the right.	64-bit signed binary	12.0044
`boolean`	"1" or "0"	bit	0, 1 (1=="true")
`dateTime.iso8601`	A date in ISO 8601 format, with optional time and no optional zone. Fractional seconds may be as precise as nanoseconds.	Structure or object containing year, month, hour, minute, second, nanosecond.	19941105T08:15:00301
`dateTime.iso8601tz`	A date in ISO 8601 format, with optional time and optional zone. Fractional seconds may be as precise as nanoseconds.	Structure or object containing year, month, hour, minute, second, nanosecond, zone.	19941105T08:15:5+03
`date.iso8601`	A date in ISO 8601 format. (no time)	Structure or object containing year, month, day.	19541022

time.iso8601	A time in ISO 8601 format, with no date and no time zone.	Structure or object exposing day, hour, minute	
time.iso8601.tz	A time in ISO 8601 format, with no date but optional time zone.	Structure or object containing day, hour, minute, zonehours, zoneminutes.	08:15-05:00
i1	A number, with optional sign, no fractions, no exponent.	8-bit binary	1, 255
i2	"	16-bit binary	1, 703, -32768
i4	"	32-bit binary	
i8	"	64-bit binary	
ui1	A number, unsigned, no fractions, no exponent.	8-bit unsigned binary	1, 255
ui2	"	16-bit unsigned binary	1, 703, -32768
ui4	"	32-bit unsigned binary	
ui8	"	64-bit unsigned binary	
r4	Same as "number."	IEEE 488 4-byte float	
r8	"	IEEE 488 8-byte float	

`float.IEEE.754.32`	"	IEEE 754 4-byte float	
`float.IEEE.754.64`	"	IEEE 754 8-byte float	
`uuid`	Hexidecimal digits representing octets, optional embedded hyphens which should be ignored.	128-bytes Unix UUID structure	F04DA480-65B9-11d1-A29F-00AA00C14882
`uri`	Universal Resource Identifier	Per W3C spec	http://www.ics.uci.edu/pub/ietf/uri/draft-fielding-uri-syntax-00.txt http://www.ics.uci.edu/pub/ietf/uri/ http://www.ietf.org/html.charters/urn-charter.html
`bin.hex`	Hexidecimal digits representing octets	no specified size	
`char`	string	1 Unicode character (16 bits)	
`string.ansi`	string containing only ascii characters <= 0xFF.	Unicode or single-byte string.	This does not look Greek to me.

All of the dates and times above reading "iso8601.." actually use a restricted subset of the formats defined by ISO 8601. Years, if specified, must have four digits. Ordinal dates are not used. Of formats employing week numbers, only those that truncate year and month are allowed (5.2.3.3 d, e and f).

Mapping between Schemas

Certain uses of data emphasize syntax, others "conceptual" relations. Syntactic schemas often have fewer elements compared to explicitly conceptual ones. Further, it is usually easier to design a schema that merely covers syntax rather than designing a well-thought-out conceptual data model. An effect of this is that many practical schemas will not contain all the elements that a conceptual schema would, either for reasons of economy or because the initial schema was simply syntactic. But is it useful to make the implicit explicit over time so that more generic processors can make use of data.

For example, the following schema is essentially syntax:

```
<elementType id="author">
    <string/>
</elementType>

<elementType id="title">
    <string/>
</elementType>

<elementType id="Book">
    <element type="#title"/>
    <element type="#author" occurs="ONEORMORE"/>
</elementType>
```

with instances looking like this

```
<Book>
    <title>Paradise Lost</title>
    <author>Milton</author>
</Book>
```

On the other hand, a conceptual schema could look like this:

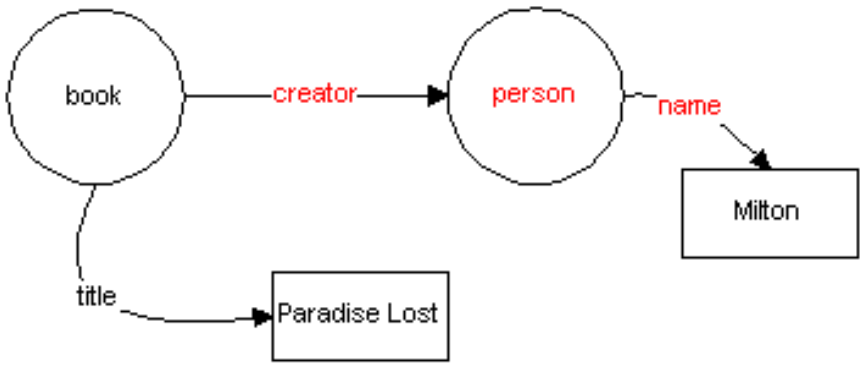
```
<elementType id="name">
    <string/>
</elementType>

<elementType id="Person">
    <element type="#name/>
</elementType>

<elementType id="creator">
    <range type="#Person/>
</elementType>

<elementType id="title">
    <string/>
</elementType>

<elementType id="Book">
    <element type="#title"/>
    <element type="#creator" occurs="ONEORMORE"/>
</elementType>
```

If fully explicit, its instances would look something like this:

```
<Person id="thing1">
     <name>Milton</Person>
</Person>
```

```
<Book>
    <title>Paradise Lost</title>
    <creator>
        <Person>
            <name>Milton</name>
        </Person>
    </creator>
</Book>
```

In any case, what we want to express is a diagram such as this:

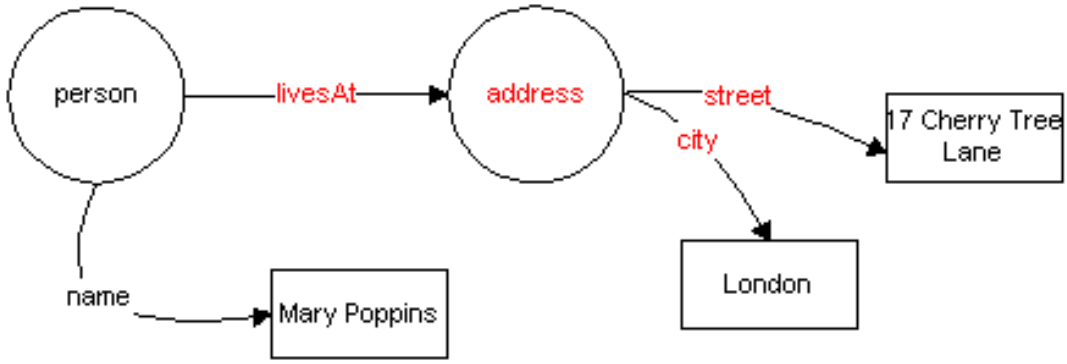


To do this, we will add mapping information into the syntactic schema which tells us how to interpolate the implied elements (and also to map *author* to *creator*) thereby creating a conceptual data model.

```
<?xml:namespace href="uri-to-the-conceptual-schema" as="c" ?>
<elementType id="author">
    <string/>
</elementType>

<elementType id="title">
    <string/>
</elementType>

<elementType id="Book">
    <mapsTo type="c:book"/>
    <element type="#title"> <mapsTo type="c:title"/> </element>
    <element type="#author" occurs="ONEORMORE">
        <mapsTo type="string">
            <implies type="c:name">
                <implies type="c:person">
                    <implies type="c:creator"/>
                </implies>
            </implies>
        </mapsTo>
    </element>
</elementType>
```

A more complex case could involve needing to map several properties to have a common implied node. For example, suppose we wanted that a *street* element and *city* element should both imply the same *address* node.

```
<Person>
    <name>Mary Poppins</name>
    <street>17 Cherry Tree Lane</street>
    <city>London</city>
</Person>
```



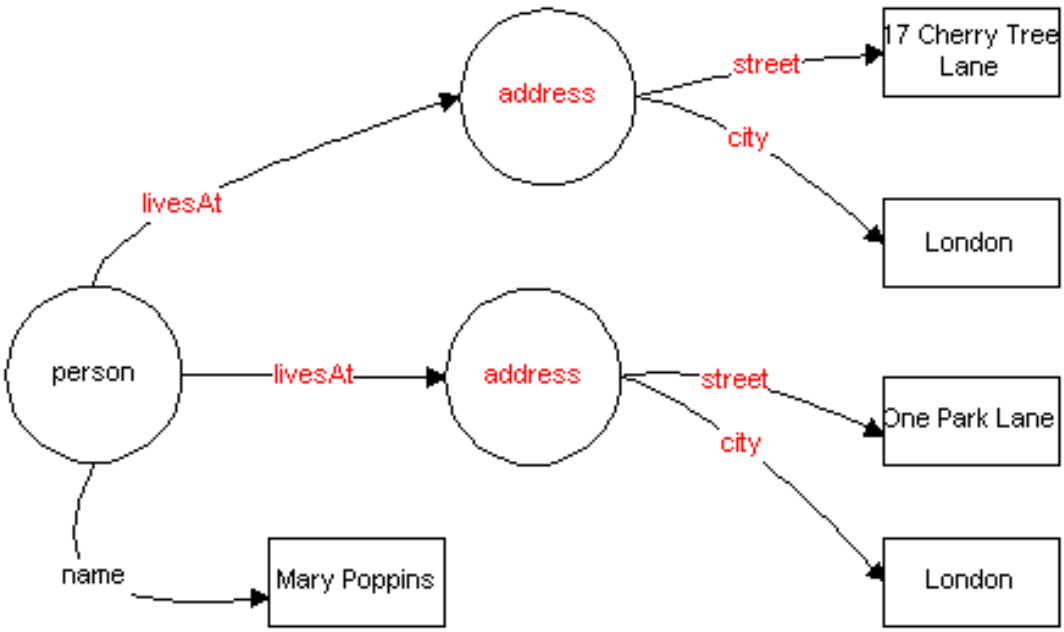
That is, rather than creating two *address* nodes, we want to create only a single onc, and subordinate both the *street* and *city* to it. If the conceptual schema has elements *livesAt*, *address*, *street* and *city*, we could write a mapping thus:

```
...definitions of name, street and city...

<elementType id="Person">
    <mapsTo type="c:person"/>
    <element type="#name">    <string/>   <mapsTo type="c:name"/
>   </element>
    <element type="#street">
        <string/>
        <mapsTo type="c:street">
              <implies type="c::address" id="livesAtAddress">
                  <implies type="c:livesAt"/>
              </implies>
        </mapsTo>
    </element>
    <element type="#city">
        <string/>
        <mapsTo type="c:city">
            <implies type="#livesAtAddress"/>
        </mapsTo>
    </element>
</elementType>
```

Elements may be repeated, so mapping rules need to accommodate repetitions. Suppose that someone has two addresses in the grammatical syntax, this needs to map to two addresses in the graph while still keeping the structure correct.

```
<Person>
    <name>Mary Poppins</name>
```

```
        <street>17 Cherry Tree Lane</street>
        <city>London</city>
        <street>One Park Lane</street>
        <city>London</city>
    </Person>
```



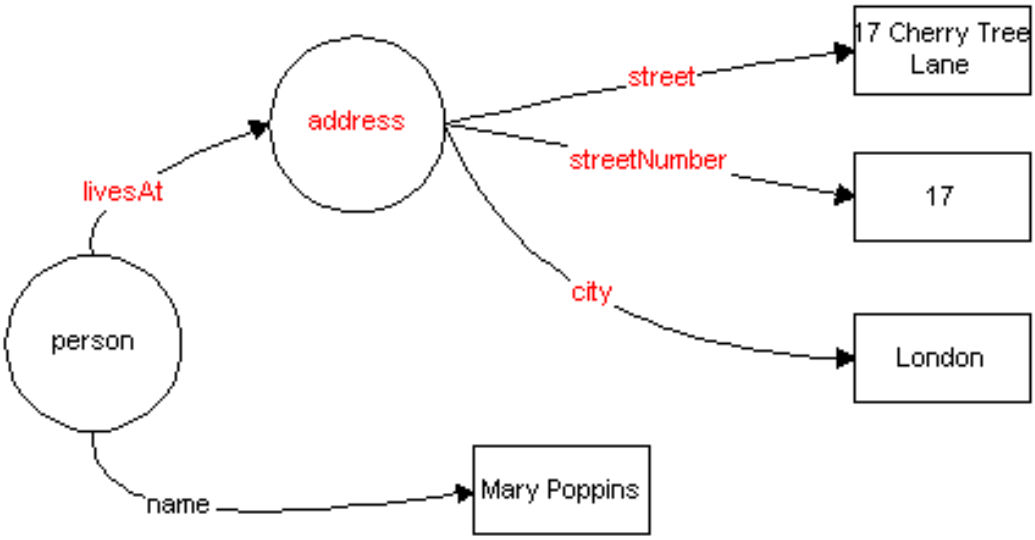

```
    <elementType id="Person">
        <mapsTo type="c:person"/>
        <element type="#name">    <string/>   <mapsTo type="c:name"/
    >   </element>
        <group occurs="ZEROORMORE"/>
            <element type="#street">
                <string/>
                <mapsTo type="c:street">
                    <implies type="c::address" id="livesAtAddress">
                        <implies type="c:livesAt"/>
                    </implies>
                </mapsTo>
            </element>
            <element type="#city">
                <string/>
                <mapsTo type="c:city">
                    <implies type="#livesAtAddress"/>
                </mapsTo>
            </element>
        </group>
    </elementType>
```

Mappings within groups are handled together. Since *street* and *city* are in a single group, each occurrence of such a group results in one *address*.

Text markup can also be handled by mapping. Suppose that for some reason we choose to markup the number portion of a street address:

```
<Person>
    <name>Mary Poppins</name>
    <street>< streetNumber>17</ streetNumber > Cherry Tree Lane</
street>
    <city>London</city>
</Person>
```

If this should be reflected in the graph,



We can do that with mapping such as:

```
<elementType id="streetNumber">
    <string/>
</elementType>

<elementType id="street>
    <mixed>
        <element type="# streetNumber">
            <mapsTo type="c: streetNumber">
                <implies type="#livesAtAddress"/>
            </mapsTo>
        </element>
    </mixed>
</elementType>

...Person defined as before...
```

Appendix A: Examples

Some data:

```
<?xml:namespace name="http://company.com/schemas/books/" as="bk"/>
```

```
<?xml:namespace name="http://www.ecom.org/schemas/dc/" as="ecom" ?>

<bk:booksAndAuthors>
    <Person>
        <name>Henry Ford</name>
        <birthday>1863</birthday>
    </Person>

    <Person>
        <name>Harvey S. Firestone</name>
    </Person>

    <Person>
        <name>Samuel Crowther</name>
    </Person>

    <Book>
        <author>Henry Ford</author>
        <author>Samuel Crowther</author>
        <title>My Life and Work</title>
    </Book>

    <Book>
        <author>Harvey S. Firestone</author>
        <author>Samuel Crowther</author>
        <title>Men and Rubber</title>
        <ecom:price>23.95</ecom:price>
    </Book>
</bk:booksAndAuthors>
```

The schema for http://company.com/schemas/books:

```
<?xml:namespace name="urn:uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882/"
as="s"/?>
<?xml:namespace href="http://www.ecom.org/schemas/ecom/" as="ecom" ?>

<s:schema>

    <elementType id="name">
        <string/>
    </elementType>

    <elementType id="birthday">
        <string/>
        <dataType dt="date.ISO8601"/>
    </elementType>

    <elementType id="Person">
        <element type="#name" id="p1"/>
        <element type="#birthday" occurs="OPTIONAL">
```

```
            <min>1700-01-01</min><max>2100-01-01</max>
        </element>
        <key id="k1"><keyPart href="#p1" /></key>
    </elementType>

    <elementType id="author">
        <string/>
        <domain type="#Book"/>
        <foreignKey range="#Person" key="#k1"/>
    </elementType>

    <elementType id="writtenWork">
        <element type="#author" occurs="ONEORMORE"/>
    </elementType>

    <elementType id="Book" >
        <genus type="#writtenWork"/>
        <superType href=" http://www.ecom.org/schemas/ecom/commercialItem"/
>
        <superType href=" http://www.ecom.org/schemas/ecom/inventoryItem"/>
        <group groupOrder="SEQ" occurs="OPTIONAL">
            <element type="#preface"/>
            <element type="#introduction"/>
        </group>
        <element href="http://www.ecom.org/schemas/ecom/price"/>
        <element href="ecom:quantityOnHand"/>
    </elementType>

    <elementTypeEquivalent id="livre" type="#Book"/>
    <elementTypeEquivalent id="auteur" type="#author"/>

</s:schema>
```

Appendix B : An XML DTD for XML-Data schemas

```
<!ENTITY % nodeattrs 'id ID #IMPLIED'>

<!-- href is as per XML-LINK, but is not required unless there is
      no content -->

<!ENTITY % linkattrs
                    'id ID #IMPLIED
                     href CDATA #IMPLIED'>

<!ENTITY % typelinkattrs
                    'id ID #IMPLIED
                     type CDATA #IMPLIED'>
```

```
<!ENTITY % exattrs
                     'name CDATA #IMPLIED
                      content (OPEN|CLOSED) "OPEN" >

<!ENTITY % elementTypeElements1
                     ‘genus? correlative? superType*’>

<!ENTITY % elementTypeElements2
                     ‘description,
                      (min|minExclusive)?,
                      (max | maxInclusive)?,
                      domain*,
                      key*,
                      foreignKey*,
                      (datatype | ( syntax?, objecttype+ ) )?
                      mapsTo?’>

<!ENTITY % elementConstraints
                     'min? max? default?'>

<!ENTITY % elementAttrs
                     'occurs (REQUIRED|OPTIONAL|ONEORMORE|ZEROORMORE)
"REQUIRED" '>

<!ENTITY % rangeAttribute
                     'range CDATA #IMPLIED'  >

<!-- The top-level container -->
<!element schema     ((elementType|linkType|
                       extendType|
                       intEntityDcl|extEntityDcl|
                       notationDcl|extDcls)*)>
<!attlist schema %nodeattrs;>

<!-- Element Type Declarations -->

<!element elementType (%elementTypeElements1;,
                       ((element|group)*|empty|any|string|mixed)?,
                        attribute*
                        %elementTypeElements2 )>

<!attlist elementType %nodeattrs;
                      %exattrs >

<!-- Element types allowed in content model -->

<!-- Note this is just short for a model group with only one element in it
-->
<!element element  (%elementConstraints;) >

<!-- The type is required -->
```

```
<!attlist element   %typelinkattrs;
                    %elementAttrs;
                    presence (FIXED) #IMPLIED >

<!-- A group in a content model: and, sequential or disjunctive -->
<!element group     ((group|element)+)>
<!attlist group     %nodeattrs;
                    %elementattrs;
                    presence (FIXED) #IMPLIED
                    groupOrder (AND|SEQ|OR) 'SEQ'>

<!element any       EMPTY>
<!element empty     EMPTY>
<!element string    EMPTY>

<!-- mixed content is just a flat, non-empty list of elements -->
<!-- We don't need to say anything about <string/> (CDATA), it's implied --
>

<!element mixed         (element+)>
<!attlist mixed         %nodeattrs;>

<!element superType  EMPTY>
<!attlist superType %linkattrs;>

<!element genus  EMPTY>
<!attlist genus %typelinkattrs;>

<!element description  MIXED>
<!attlist description %nodeattrs;>

<!element domain  EMPTY>
<!attlist domain %typelinkattrs;>

<!element default MIXED>
<!attlist default %nodeattrs;>

<!element min MIXED>
<!attlist min %nodeattrs; >

<!element max MIXED>
<!attlist max %nodeattrs; >

<!element maxInclusive MIXED>
<!attlist maxInclusive %nodeattrs; >

<!element minExclusive MIXED>
<!attlist minExclusive %nodeattrs; >

<!element key (keyPart+)>
<!attlist key %nodeattrs;>
```

```
<!element keyPart EMPTY>
<!attlist keyPart %linkattrs;>

<!element foreignKey foreignKeyPart* >
<!attlist foreignKey %nodeattrs;
                     %rangeAttribute;
                     key CDATA #IMPLIED >

<!element foreignKeyPart EMPTY>
<!attlist foreignKeyPart %linkattrs;>

<!-- Datatype support -->

<!element datatype (elementType?) >
<!attlist datatype %typelinkattrs;
       presence (IMPLIED|SPECIFIED|REQUIRED|FIXED) #IMPLIED >

<!element syntax >
<!attlist syntax %linkattrs; >

<!element objecttype >
<!attlist objecttype %linkattrs; >


<!-- Mapping support -->

<!element mapsTo (implies?)>
<!attlist mapsTo %typelinkattrs;>

<!element implies (implies?)>
<!attlist implies %typelinkattrs;>

<!-- Alias support -->


<!element elementTypeEquivalent EMPTY>
<!attlist elementTypeEquivalent %typelinkattrs; >

<!element correlative EMPTY>
<!attlist correlative %linkattrs;>


<! Subtype of ElementType that is explicitly a relation. -->

<!element relationType (%elementTypeElements1;,
                        ((element|group)*|empty|any|string|mixed)?,
                        attribute*
                        %elementTypeElements2)>
<!attlist relationType %nodeattrs;
```

```
                %exattrs; >



<!-- Attributes -->
<!-- default value must be present if presence is specified or fixed -->
<!-- presence defaults to specified if default is present, else implied -->

<!element attribute  (%PropertyElements1,
                      %PropertyElements2,
                      %elementConstraints)>
<!attlist attribute  %typelinkattrs;
                     name CDATA #IMPLIED
                     %elementAttrs
                     dt          CDATA #IMPLIED
                     atttype    (URIREF|
                                 ID|IDREF|IDREFS|ENTITY|ENTITIES|
                                 NMTOKEN|NMTOKENS|
                                 ENUMERATION|NOTATION|CDATA) CDATA
                     %rangeAttribute;
                     default CDATA #IMPLIED
                     values NMTOKENS #IMPLIED
                     presence (IMPLIED|SPECIFIED|REQUIRED|FIXED) #IMPLIED >

<!-- Notation and Entity Declarations -->
<!-- Note: as this is written, only external entities
      can have structure without escaping it -->
<!-- 'par' is TRUE iff parameter entity. -->
<!-- systemID and publicId (if present) must have the required syntax -->

<!ENTITY % notationattrs '%nodeattrs
                          systemID CDATA #IMPLIED
                          publicID CDATA #IMPLIED'>

<!ENTITY % entityattrs   '%notationattrs
                          name CDATA #IMPLIED
                          par     (TRUE | FALSE) "FALSE">



<!-- Notation Declarations -->

<!element notationDcl        EMPTY>
<!attlist notationDcl        %notationattrs>

<!element intEntityDcl       PCDATA>
<!attlist intEntityDcl       %entityattrs; >

<!-- The entity will be treated as binary if a notation is present -->
<!element extEntityDcl       EMPTY>
<!attlist extEntityDcl       %entityattrs;
                             notation CDATA #IMPLIED>
```

```
<!-- External entity with declarations to be included -->
<!element extDcls            EMPTY>
<!attlist extDcls            %entityattrs;>
```

Datatypes for DTDs (DT4DTD) 1.0

W3C Note 13 January 2000

This Version:
http://www.w3.org/TR/2000/NOTE-dt4dtd-20000113
Latest version:
http://www.w3.org/TR/dt4dtd
Editors:
Lee Buck, Extensibility <leebuck@extensibility.com>
Charles F. Goldfarb, The XML Handbook™ <charles@xmlbooks.com>
Paul Prescod, ISOGEN International <paul@isogen.com>

Abstract

The presented specification allows legacy systems that may presently be unable to convert their DTD markup declarations to XML Schema, to utilize XML Schema conformant datatypes. With it, DTD creators can specify datatypes for attribute values and data content, thereby providing the foundation for a smoother future transition path.

> **NOTE:** Free open-source code that supports this specification for both SAX and DOM is available at www.extensibility.com/dt4dtd.

Status of This Document

This document is a submission to the World Wide Web Consortium from Extensibility, Inc. (see Submission Request, W3C Staff Comment). For a full list of all acknowledged Submission, please see Acknowledged Submissions to W3C.

This document is a Note made available by W3C for discussion only. This work does not imply endorsement by, or the consensus of the W3C membership, nor that W3C has, is, or will be allocating any resources to the issues addressed by the Note. This document is a work in progress and may be updated, replaced, or rendered obsolete by other documents at any time.

A list of current W3C technical documents can be found at the Technical Reports page.

Table of Contents

1. Datatype Declarations

A datatype declaration is a fixed attribute named "a-dtype" or "e-dtype". The fixed, "a-dtype" default value must consist of a series of name/value pairs according to the following syntax:

Attribute datatyping

[1]	a-dtype	::=	S? (attrName S dtypeName) (S attrName S dtypeName)* S?
[2]	dtypeName	::=	NCName

The fixed, e-dtype default value must consist merely of a name.

Content datatyping

[3]	e-dtype	::=	dtypeName

For example:

```
<!ATTLIST person
   birthdate CDATA #IMPLIED
   height CDATA #IMPLIED
   e-dtype CDATA #FIXED
      "social-security-number"
   a-dtype CDATA #FIXED
      "pubdate date
       binding length">
```

The attrNames are the names of attributes declared in the same attribute-list declaration for the element type. The dtypeName that follows is associated with the attribute of that name.

The e-dtype attribute can only be declared if the associated element type's content allows data but no sub-elements. The dtypeName is associated with the data content of elements of that type.

> **NOTE:** The "dtype" attributes are based on notations instead of XML Namespaces because they are meaningful in the DTD and not in the document instance. It must be possible for DTD-reading and writing applications to recognize the attributes in an entity containing DTD declarations without parsing a document for namespace declarations. The set of namespace declarations that apply to an element type can vary according to context and cannot in general be recognized in a DTD.

A dtypeName must be the name of a notation declared in the DTD; the notation declaration must include a system identifier that is a URI reference. However, if the datatype's name is somehow known to the information system because of the processing context, these notation declarations may be omitted. Notation declarations for datatypes native to the W3C XML Schema specifiation may always be omitted; every information system must natively recognize them from their dtypeName.

2. User-defined datatypes

When explicit notations are provided for dtypeNames, their associated system URIs must refer to one of the following.

2.1 XML Schema datatypes

The referent of the associated URI reference may be an XML Schema *datatypeDefn*. If so, a conforming processor may validate the attribute value or data content according to that specification.

2.2 XML-Data datatypes

The referent of the associated URI may be XML-Data datatypes as defined by these declarations, which may be included directly or through an entity reference.

```
<!NOTATION string
   SYSTEM "urn:schemas-microsoft-com:datatypes/
string">
<!NOTATION number
   SYSTEM "urn:schemas-microsoft-com:datatypes/
number">
<!NOTATION int
   SYSTEM "urn:schemas-microsoft-com:datatypes/int">
<!NOTATION float
   SYSTEM "urn:schemas-microsoft-com:datatypes/
float">
<!NOTATION fixed.14.4
   SYSTEM "urn:schemas-microsoft-com:datatypes/
fixed.14.4">
<!NOTATION boolean
   SYSTEM "urn:schemas-microsoft-com:datatypes/
boolean">
<!NOTATION dateTime.iso8601
   SYSTEM "urn:schemas-microsoft-com:datatypes/date.
type.iso8601">
<!NOTATION dateTime.iso8601tz
   SYSTEM "urn:schemas-microsoft-com:datatypes/
dateTime.iso8601tz">
<!NOTATION date.iso8601
   SYSTEM "urn:schemas-microsoft-com:datatypes/date.
iso8601">
<!NOTATION time.iso8601
   SYSTEM "urn:schemas-microsoft-com:datatypes/time.
iso8601">
<!NOTATION time.iso8601.tz
   SYSTEM "urn:schemas-microsoft-com:datatypes/time.
iso8601.tz">
<!NOTATION i1
   SYSTEM "urn:schemas-microsoft-com:datatypes/i1">
<!NOTATION i2
```

```
   SYSTEM "urn:schemas-microsoft-com:datatypes/i2">
<!NOTATION i4
   SYSTEM "urn:schemas-microsoft-com:datatypes/i4">
<!NOTATION i8
   SYSTEM "urn:schemas-microsoft-com:datatypes/i8">
<!NOTATION ui1
   SYSTEM "urn:schemas-microsoft-com:datatypes/ui1">
<!NOTATION ui2
   SYSTEM "urn:schemas-microsoft-com:datatypes/ui2">
<!NOTATION ui4
   SYSTEM "urn:schemas-microsoft-com:datatypes/ui4">
<!NOTATION ui8
   SYSTEM "urn:schemas-microsoft-com:datatypes/ui8">
<!NOTATION r4
   SYSTEM "urn:schemas-microsoft-com:datatypes/r4">
<!NOTATION r8
   SYSTEM "urn:schemas-microsoft-com:datatypes/r8">
<!NOTATION float.IEEE.754.32
   SYSTEM "urn:schemas-microsoft-com:datatypes/float.
IEEE.754.32">
<!NOTATION float.IEEE.754.64
   SYSTEM "urn:schemas-microsoft-com:datatypes/float.
IEEE.754.64">
<!NOTATION uuid
   SYSTEM "urn:schemas-microsoft-com:datatypes/uuid">
<!NOTATION uri
   SYSTEM "urn:schemas-microsoft-com:datatypes/uri">
<!NOTATION bin.hex
   SYSTEM "urn:schemas-microsoft-com:datatypes/bin.
hex">
<!NOTATION char
   SYSTEM "urn:schemas-microsoft-com:datatypes/char">
<!NOTATION string.ansi
   SYSTEM "urn:schemas-microsoft-com:datatypes/strin.
ansi">
```

2.3 Other datatypes

The referent of the associated URI may be some other notation established for use within some processing context. Optionally, such referents may be Java classes, XSL stylesheets, et. al. that are able to programatically test for validity.

3. Strict Conformance

This specification has certain optional features intended to support an existing installed base and also to allow customization of the specification for closed information systems. This section defines a subset called the "Strict Conformance Subset" that should be used for blind (not pre-arranged) information interchange on the World Wide Web.

A DTD that conforms to the strict subset will have a notation declaration for every dtypeName used in a datatype declaration except datatypes declared in the W3C XML Schema specification. It will aslo have a single explicit conformance declaration.

3.1 Conformance Declaration

The conformance declaration must only be used in DTDs that otherwise conform to the strict subset. This declaration is a notation declaration with the URI "http://www.w3.org/TR/1999/dt4dtd".

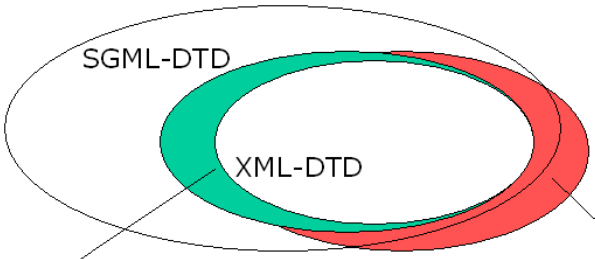
If present, the conformance notation declaration must precede the first attribute list declaration in a set of DTD declarations. For the convenience of human readers, we suggest that the conformance notation be named "dt4dtd".

4. Information Set Contribution

> **NOTE:** The principle of Information Set Contributions is established in the W3C XML Schema specification.

In an information set supporting this specification:

- every element information item must have associated "data type name" and "data type URI" properties.
- The value of the data type name property is the value of the dtypeName associated with the element type through the e-dtype attribute. If no such attribute exists then the value is null.
- If the associated dtypeName is a notation name then the data type URI property value is the absolute URI that can be generated from the system identifier. Otherwise it is null.
- every attribute information item must have associated data type name and data type URI properties.
- The data type name property value is the dtypeName associated with the attribute through a matching attrName in the a-dtype attribute. If no such declaration exists then the value is null.
- If the associated dtypeName is a notation name then the data type URI property value is the absolute URI that can be generated from the system identifier. Otherwise it is null.



Erweiterung des XML-DTD Mechanismus
um weitere Elemente der SGML-DTD:
(+) Ausdrucksmächtigkeit nähert sich (wieder) der von SGML an
(-) ...die Komplexität auch
(-) Ausdrucksmächtigkeit kann die von SGML niemals übertreffen

Erweiterung des XML-DTD Mechanismus um Elemente,
die *nicht* mit SGML-Mitteln ausdrückbar sind:
(+) Freiheitsgrad hinsichtlich beliebiger Erweiterungen
(-) XML-Grundforderung nach Untermengenbeziehung zu SGML entfällt
(-) immernoch zwei verschiedene Sprachen für Inhalt und Schema

Based on their expertise in the B2B market, Commerce One is confident of successfully integrating businesses in the Korean market. — **E-week**

Watch the Conductor Product Overview Click Here

News

- Commerce One and eFORCE Team to Deliver Next-Generation Integration Solutions to Global 1000 Clients
- Commerce One Reports Final First Quarter 2004 Results
- Commerce One Schedules 2004 Q1 Earnings Announcement and Conference Call

Customer Highlights

GHX Global Healthcare Exchange

Schema for Object-Oriented XML 2.0

W3C Note 30 July 1999

This version:
http://www.w3.org/1999/07/NOTE-SOX-19990730
Latest version:
http://www.w3.org/TR/NOTE-SOX
Previous version:
http://www.w3.org/TR/1998/NOTE-SOX-19980930
Authors:
Andrew Davidson
Matthew Fuchs
Mette Hedin
Mudita Jain
Jari Koistinen
Chris Lloyd
Murray Maloney
Kelly Schwarzhof

Status of this document

This document is a NOTE made available by the W3 Consortium for discussion only. This indicates no endorsement of its content by the W3C, nor that the Consortium has allocated, is allocating, or will be allocating any resources to the issues addressed by the NOTE.

It replaces the previous version of the SOX language specification and represents the current, implemented version of the language.

This document is a submission to W3C from Commerce One, Inc.. Please see Acknowledged Submissions to W3C regarding its disposition.

Comments on this document should be sent to schema@commerceone.com.

---

1. Abstract

This document describes SOX 2.0, the second version of the Schema for Object-Oriented XML. SOX is a schema language (or metagrammar) for defining the syntactic structure and partial semantics of XML document types. As such, SOX is an alternative to XML DTDs and can be used to define the same class of document types (with the exception of external parsed entities). However, SOX extends the language of DTDs by supporting:

1. An extensive (and extensible) set of datatypes
2. Inheritance among element types
3. Namespaces
4. Polymorphic content
5. Embedded documentation
6. Features to enable robust distributed schema management.

All of these features are supported with strong type-checking and validation. A SOX schema is also a valid XML instance according to the SOX DTD, enabling the application of XML content management tools to schema management.

SOX was initially developed to support the development of large-scale, distributed electronic commerce applications but is applicable across the whole range of applications of markup. As compared to XML DTDs, SOX dramatically decreases the complexity of supporting interoperation among heterogenous applications by facilitating software mapping of XML data structures, expressing domain abstractions and common relationships directly and explicitly, enabling reuse at the document design and the application programming levels, and supporting the generation of common application components.

Although SOX 2.0 retains many of the features of SOX 1.0, it represents an additional year of actual implementation experience. Commerce One has a working implementation of this language and will be releasing products based on it. Our goals in releasing this second version are two-fold:

1. Ensure that the public record accurately reflects the evolution of SOX. Members of the community wishing to understand SOX, build tools using SOX, or interoperate with SOX applications, need access to the current version of the language.
2. Expose some of the fruits of our implementation experience to the general community, particularly to assist in the ongoing work at the W3C to develop an official XML Schema language.

From the markup world, the SOX proposal is informed by the XML 1.0 [XML] specification as well as the XML-Data submission [XML-Data] and the Document Content Description submission[DCD]. However many of SOX' requirements come from the distributed computing world and SOX features have been heavily influenced by the Java[JAVA] programming language.

1.1. Table of Contents

2. Soxtype processing instruction

As a SOX document is not defined by a DTD, the native XML 1.0 Doctype declaration is not appropriate for a document instance to declare SOX Schema information (note that this may not continue to be true once the W3C Schema WG defines an official mechanism). Therefore we have created the *soxtype declaration* , which mimics the XML doctype declaration, but using a processing instruction. The soxtype declaration must be the first statement in a document following the optional XML declaration and declares that the default namespace [XML-Namespaces] of this document is that of the given SOX Schema, just as a doctype declaration declares that an XML 1.0 document conforms to the given Document Type Declaration (DTD). The soxtype declaration is not currently intended to coexist in the same document with a doctype declaration, but there is no particular restriction prohibiting this.

The soxtype declaration includes two parts:

1. The PI target, which is *soxtype.*
2. The uri [URI] specifying a schema to be the default namespace of the instance.

An example declaration would look like:

```
<?soxtype urn:x-commerceone:document:com:commerceone:
schema1.sox$1.0?>
```

where the schema definition is located through resolving the urn urn:x-commerceone:document:com:commerceone:schema1:schema1.sox$1.0. The part following the "$" gives the version number.

A small sample document would look like:

```
<?soxtype urn:x-commerceone:document:com:commerceone:
schema1.sox$1.0?>
<Root>
   <Body/>
</Root>
```

The root element of the instance is not required to belong to the default namespace. This will be further elaborated in the discussion of the *import* PI below.

---

3. Import processing instruction

With the arrival of namespaces and polymorphism, it is no longer necessarily possible to indicate a single schema to which the entire instance conforms. Nevertheless, we require a mechanism to indicate a set of schemata containing definitions for all the element and datatypes appearing in an instance. This function is handled by the *import* processing instruction. The *import* PI contains one argument, an absolute URI indicating a schema.

An example import PI would look like:

```
<?import urn:x-commerceone:document:com:commerceone:
schema1.sox$1.0?>
```

where the schema definition is located through resolving the URI: `urn:x-commerceone:document:com:commerceone:schema1.sox$1.0`. The part following the "$" gives the version number.

A small sample document would look like:

```
<?soxtype urn:x-commerceone:document:com:commerceone:
schema1.sox$1.0?>
<?import urn:x-commerceone:document:com:commerceone:
schema2.sox$1.0?>
<Root
   <s2:Body  xmlns:s2="urn:x-commerceone:document:com:
commerceone:schema2.sox$1.0"/>
</Root>
```

In the following, the namespace attributes on *Root* overrides the default namespace declared in the soxtype PI, and *Body* is assumed to also be in *schema2.sox*.

```
<?soxtype urn:x-commerceone:document:com:commerceone:
schema1.sox$1.0?>
<?import urn:x-commerceone:document:com:commerceone:
schema2.sox$1.0?>
<Root xmlns="urn:x-commerceone:document:com:commerceone:
```

```
schema2.sox$.0">
   <Body/>
</Root>
```

3.1 Validity constraint

It is not necessary that for every element type appearing in the instance there be a corresponding *import* or *soxtype* processing instruction. Nevertheless, all schema information must be available before processing of the root element begins. We can define a transitive closure property over schemata to accomplish this.

We require each schema declared in a *soxtype* or *import* PI to be processed. Furthermore, if any definition in a schema being processed refers to a definition (*elementtype* or *datatype*) in another schema, that other schema must be processed. The set of imports must be sufficient so that starting with the *soxtype* and the *imports* , and processing all schemata transitively referenced by them, all *elementtypes* and *datatypes* found in the instance will have been processed.

---

4. Schema definitions

The definition of an XML schema is performed with the *schema* element. The corresponding DTD fragment is:

```
<!ELEMENT schema (intro?,
                  (datatype | elementtype |join
                  |comment | namespace)*)>
<!ATTLIST schema
                  prefix NMTOKEN #IMPLIED
                  uri CDATA #REQUIRED
                  soxlang-version NMTOKEN #FIXED "V2.0">
```

The following is a minimal valid instance of a schema:

```
<schema uri="urn:x-commerceone:sampleSchema"/>
```

However it is unlikely there will be many schemata containing no definitions at all.

A *schema* consists of any number of definitions of datatypes or element types (both of which will be explained subsequently). In other words, a schema is a set of definitions, not the set of files, database entries, etc., we use to store a representation of these definitions: the physical storage mechanism we use may change frequently, without the set of definitions being affected at all. A schema may start out as a single file, then be split among several files (which would currently be linked using the *join* mechanism described below), before being stored in a database. But despite being stored in three different ways, the set of definitions remains the same, and it is that set which comprises the schema.

Each schema has a unique name to identify it. This name is a URI, given in the *uri* attribute of the *schema* element in a fragment. In essence, it proclaims the set of definitions in this schema element to be a subset of the definitions forming the schema identified by the uri attribute.

The *join* construct allows definitions from externally defined fragments belonging to the same schema to be pulled in.

There are three main classes of symbols created during the construction of a Schema: *elementtype* names, *datatype* names, and *namespace* prefixes. In the current version of the language, both *elementtype* names and *datatype* names are maintained in a single set; it is illegal to use the same name to represent both an element type and a datatype. Prefixes for namespaces, however, are kept separately. It is entirely legal, although potentially confusing, to use the same name for a *namespace* prefix as for an *elementtype* or *datatype* .

NOTE: It is our intention to separate the datatype and elementtype namespaces in a future version of SOX. When we do so, this will be entirely backwards compatible. The expressive power of SOX, however, is unchanged whether the namespaces are separated or coallesced.

The *prefix* attribute specifies a prefix available for referencing names created in this schema. As the current schema is the default namespace for dereferencing names, this is not strictly necessary, however it can be useful when definitions from several namespaces are mixed in close proximity.

The *uri* attribute provides a URI establishing the namespace of this schema fragment. This attribute has become required to prevent accidental name capture through the join mechanism described below. This must be an absolute URI.

The *namespace* element is considered a *declaration,* not a definition. The namespace being declared is defined elsewhere (probably in a Schema file)

N.B. SOX does not deal with a number of linking issues related to the organization of schemata into multiple files.

The *intro* element is available to provide an introduction to the schema as a whole. It consists of a number of HTML elements. The exact allowable contents of *intro* is available in the *htmltext.ent* file reproduced at the end of this document.

4.1 Validity issues

The following are the constraints placed on Schema files, aside from strict structural conformance to the DTD.

A Schema is about types. Types are both defined and referenced. Referencing is done using names. Once a type (or namespace) is given a name (or prefix) in a fragment, that name that name is *bound* in that fragment. Names that are only

referenced are *free* in that fragment.

A Schema fragment is processed in an *environment.* That environment extends beyond the fragment to include other schemata and the SOX definition itself. This document defines how the environment of a fragment is defined, but does not discuss how it is physically constructed.

A Schema fragment cannot be successfully processed unless all the names that are free in it are bound somewhere in its environment. It is an error for a name to be bound twice in a fragment's environment.

A fragment specifies its environment by providing bindings for all the free names it contains. There are four pieces to its environment that a fragment must specify:

1. The fragment itself. This includes those names defined in the fragment. The fragment specifies this by its own existence.
2. The rest of the Schema this fragment is defined in. This is specified by the *uri* attribute on the *schema* element.
3. Those schemata referenced directly or indirectly from this fragment, as explained below. These are specified in the *namespace* declaration element.
4. The SOX definition. This includes all names defined in this specification or its successors. The version is specified by the *version* attribute of the *schema* element.

Names are either qualified or unqualified. Unqualified names must be defined either in the current fragment, in the rest of the schema, or in the SOX definition. It can only be defined in one of the three. Qualified names must be defined in the schema declared for that name with a namespace declaration. For each qualified name in a fragment there must be a corresponding namespace declaration in the same fragment. The *schema* element itself is considered a namespace declaration for the current namespace, so qualified names using the value in the *prefix* attribute of the *schema* element must resolve to the current schema (although not necessarily in the same fragment). In this version of the language it is an error to create a definition whose local name is either one of the intrinsic datatypes.

---

5. Namespace declarations

```
<!ELEMENT namespace (explain?)>
<!ATTLIST namespace
                    prefix NMTOKEN #REQUIRED
                    namespace CDATA #REQUIRED >
```

All of the names defined in a single Schema belong to the same namespace, and can be used without qualifier. Schemata, however, frequently need to refer to definitions in other namespaces. A namespace declaration allows access to definitions in the referenced schema when appearing with an appropriate prefix attribute. A namespace declaration is scoped to the current schema fragment only. Namespace declarations

made in one schema fragment are not visible in other fragments belonging to the same schema, even when referenced through a *join* .

NOTE: We will use a prefix attribute for the prefix, instead of using colonized names, in accordance with the XSDL spec [XSDL]. The prefix attribute shows up on various elements, including attdef, scalar , and extends, all of which include a reference to a definition.

XML requires the use of qualified names to make such references in document instances. A qualified name in an instance consists of two parts separated by a colon:

- A prefix, which is the value of a prefix attribute specified in a namespace declaration.
- A local name, which corresponds to some name defined in the target namespace.

SOX implements qualified names through the use of two attributes, one for the name, and one for the prefix. Qualified names can be used wherever a reference to a definition is allowed - a schema cannot define a name in another schema, but it can extend an elementtype from another schema, or use a datatype from another Schema.

For example, the following fragment declares the urn:foo namespace and associates it with prefix bar:

<namespace prefix="bar" namespace="urn:foo"/>

If we later need to include a foobar element from the urn:foo namespace in an element type that would be done using the following fragment:

```
<elementtype name="et">
   <model>
         <element prefix="bar" type="foobar"
name="whatever"/>
         <element prefix="bar" type="foobar"/>
   </model>
</elementtype>
```

A valid instance of this would be:

```
<et><whatever><bar:foobar xmlns:bar="urn:foo"/></whatever>
    <bar:foobar xmlns:bar="urn:foo"/></et>
```

5.1 Validity issues

Each prefix must be unique within a schema fragment. While it is not a fatal error to declare a non-existent namespace, it is a fatal error to reference an element in a non-existent namespace, or to reference a non-existent element in an existing namespace. Both of these are semantic errors. It is a fatal runtime error if the

processor is unable to retrieve a definition during processing. A processor should distinguish among these cases. This specification does not address the issue of how to retrieve definitions.

The value of the namespace attribute of the namespace declaration must be the URI of a schema, as described above. In other words, it must be the same as the value appearing in the uri attribute of the schema element of files which define that schema.

---

6. Explain element

The *explain* element exists inside several different SOX constructs. It provides a hook for including documentation within a schema and exploits commonly known HTML [HTML-4] constructs.

```
<!ELEMENT explain (title?, synopsis?, (%html.block;)+) >
```

The *title* and *html.block* elements are common HTML constructs whose exact definitions are specified in the *htmltext.ent* file included at the end of this document. The *synopsis* is used to give a purpose or synopsis to the thing being *explain* ed. It is a single paragraph of text.

6.1 Validity constraint

Because it is HTML embedded in XML, all the HTML constructs must be used in a well-formed manner. It is common to take advantage of SGML tag minimization in writing HTML documents, but that would result in well-formedness errors in a SOX schema.

---

7. Element type definitions

In XML Schema documents, element type definitions reproduce the expressiveness of XML element type declarations using explicit element and attribute markup. An element type may be defined by using the *elementtype* element with the required *name* attribute, and a subordinate *model* or *extends* element (both of which will be described subsequently).

The corresponding DTD fragment is:

```
<!ELEMENT elementtype (explain?, (extends | ((empty|
model), (attdef)*)))>
<!ATTLIST elementtype
                        name NMTOKEN #REQUIRED >
```

The following example defines an element type of name *inline* :

```
<elementtype name="inline">
   <explain>
        <synopsis>This defines the <em>inline</em>
element</synopsis>
   </explain>
   <model>
         <string/>
   </model>
</elementtype>
```

A mechanism for attaching attributes to an element type is described later.

A valid instance for this fragment would be:

```
<inline>This is a string</inline>
```

7.1 Element type name

The name of an element type may be any valid unqualified XML element type name corresponding to the *Name* production in the XML 1.0 language definition. The name must be unique among the names of element types and datatypes defined in the current Schema, which includes the current document or other documents belonging to the same Schema processed through the *join* mechanism or other resolution mechanism.

An element type may be referenced by the *element* and *extends* elements.

It is a fatal error to re-assign an element name, or to reference an element type which is not defined.

The value of the *name* attribute must be unique across all *elementtype* names defined in this schema.

7.2 Content model

The content model of an element type defines the structure and composition of an element of that type in an XML instance. The definition of a content model in XML Schema documents extends the expressiveness of XML DTDs by providing greater specificity of the minimum and maximum number of times some content model atom may be repeated. This allows an XML Schema designer with more precise control than is offered by XML's *, ? and + occurrence indicators.

The DTD fragment corresponding to an content model definition is:

```
<!ELEMENT model (string|element|choice|sequence)>
```

7.2.1 Empty content specification

An *empty* atom is used to indicate that an element may not contain any content, as in the case of the *BR* element below:

```
<elementtype name="BR">
   <empty/>
</elementtype>
```

In order to properly support extensibility (explained below) an *empty* content model is considered to be an empty *sequence* .

Valid instances are:

```
<BR/>
```

or:

```
<BR></BR>
```

7.2.2 String content model atom

The *string* atom indicates that a content model is simply string content and is an evolution of *#PCDATA* . It can be used as in the example above. In addition, the string value may be constrained to be of a particular datatype defined by the optional *datatype* attribute (and *prefix* , if the datatype is in another schema) as can be seen in the DTD fragment:

```
<!ELEMENT string EMPTY>
<!ATTLIST string
                 prefix NMTOKEN #IMPLIED
                 datatype NMTOKEN "string" >
```

Any element type with *string* in its content model is considered to be a choice group with an *occurs* value of "*". This is consistent with the XML 1.0 spec which requires that any content model containing *#PCDATA* be in a choice with a Kleene star. It is unclear if this restriction will be maintained in the Schema world.

In the example below, the *size* element type's content model is string content constrained to be an *int* :

```
<elementtype name="size">
   <model>
           <string datatype="int" />
   </model>
</elementtype>
```

A valid example of this would be:

```
<size>12345</size>
```

However the following would not be valid:

```
<size>12r34</size>
```

7.2.3 Element content model atom

A content model may also comprise zero or more repetitions of another element. The DTD fragment for this definition is:

```
<!ELEMENT element EMPTY>
<!ATTLIST element
                  prefix NMTOKEN #IMPLIED
                  type   NMTOKEN #REQUIRED
                  name   NMTOKEN #IMPLIED
                  occurs CDATA #IMPLIED >
```

The defined *element* is an instance of either a previously defined datatype or element type, which is refered to by the required *type* attribute. As before, it is a fatal error to reference a datatype or element type that is not defined.

For purposes of extensibility, a content model with just one *element* is considered a *sequence* of length one.

The *name* attribute may be used to assign a name to the defined *element* when it appears in an instance. As datatypes are not also element types, the *name* attribute must have a value when *type* references a datatype. When *name* is specified, this shows up as an additional element wrapping an element of the referenced type.

The following fragment demonstrates the use of *name* . The type *int* refers to the built-in integer datatype:

```
<elementtype name="paragraph">
   <model>
         <string/>
   </model>
</elementtype>

<elementtype name="block">
   <model>
       <sequence>
           <element name="p" type="paragraph"/>
           <element name="position" type="int"/>
       </sequence>
   </model>
</elementtype>
```

A valid instance would look like this:

```
<block><p><paragraph>this is the paragraph</paragraph></
p><position>12345</position></block>
```

The following would not be valid:

```
<block><paragraph>you must use the name</paragraph><int>1</
int></block>
```

The *occurs* attribute indicates the number of repetitions of the instanced *element* . It can take on the values of:

- one of the Kleene operators, " * " (0 or more repetitions), " *?* " (0 or 1 repetitions) or " + " (1 or more repetitions)
- a value of the form " *N1,N2* " (a value between a minimum of *N1* and a maximum of *N2* occurrences) where *N1* and *N2* are non-negative integers (N1 <=N2)
- a value of the form "N1,*" where N1 is a non-negative integer. This indicates at least N1 occurrences but no upper maximum

We will call an occurs where N1 is not equal to N2 an *indefinite occurs.* The degenerate case of "0,0" is allowed and means exactly 0 repetitions, which is treated the same as if the declaration did not occur.

NOTE: Values of the form " N1,N2 " and " N1, " are not currently supported. They will be treated as an occurs of " + " if N1 is greater than 0, or as a " * " otherwise.*

In the following example, the definition of the content model for a *list* element type specifies that it contains a minimum of 2 and a maximum of 9 *item* elements.

```
<elementtype name="list">
   <model>
         <element type="item" occurs="2,9"/>
   </model>
</elementtype>
```

A valid instance of the above would be:

```
<list><item/><item/></list>
```

7.2.4 Choice content model atom

The *choice* atom defines a content model to comprise one of a set of choices of *element* , *choice* or *sequence* content models. The relevant DTD fragment is:

```
<!ELEMENT choice ((element|choice|sequence),
                  (element|choice|sequence)+) >
```

```
<!ATTLIST choice
                name NMTOKEN #IMPLIED
                occurs CDATA #IMPLIED >
```

As with *element* , the *occurs* attribute specifies the number of repetitions, and it can take the same values as defined earlier.

In the following example, the *dl* element type's content model specifies that either a single *dt* or a single *dd* element is allowed.

```
<elementtype name="dl">
   <model>
       <choice>
           <element type="dt"/>
           <element type="dd"/>
       </choice>
   </model>
</elementtype>
```

A valid instance would be:

```
<dl><dt/></dl>
```

or

```
<dl><dd/></dl>
```

but not:

```
<dl><dd/><dt/></dl>
```

7.2.5 Sequence content model atom

The sequence atom defines a content model to consist of the specified *element* , *choice* or *sequence* content models appended together in the order specified. The relevant DTD fragment is:

```
<!ELEMENT sequence ((element|choice|sequence),
                    (element|choice|sequence)+) >
<!ATTLIST sequence
                  name NMTOKEN #IMPLIED
                  occurs CDATA #IMPLIED >
```

As before the *occurs* attribute specifies the number of repetitions of the entire *sequence* , and it can take the same values as defined earlier.

In the following example, the *dl* element type's content model specifies that a single *dt* followed by a single *dd* is allowed.

```
<elementtype name="dl">
   <model>
       <sequence>
           <element type="dt"/>
           <element type="dd"/>
       </sequence>
   </model>
</elementtype>
```

A valid instance would be:

```
<dl><dt/><dd/></dl>
```

7.2.6 Combining content model atoms

The various content model atoms defined above may be combined to allow the definition of complex content models.

For example, the *dl* element type's content model below specifies that a *dh* is followed by two or more *dt* or *dd* elements.

```
<elementtype name="dl">
   <model>
       <sequence>
           <element type="dh"/>
              <choice occurs="2,*">
                  <element type="dt"/>
                  <element type="dd"/>
              </choice>
       </sequence>
   </model>
</elementtype>
```

A valid instance would be:

```
<dl><dh/><dt/><dd/><dd/><dt/></dl>
```

7.3 Validity Issues

The *occurs* attribute may not occur on the outermost *sequence* or *choice* in an *elementtype* definition. This is the *sequence* or *choice* immediately contained within *model* . The outermost *sequence* or *choice* within the *model* must occur exactly once.

The value of the name attribute (if any) given to an element, choice, or sequence, must be unique within the innermost enclosing construct. For example, the following is legal:

```
<sequence>
```

```
   <element name="a" type="string"/>
   <element name="b" type="int"/>
</sequence>
```

while the following is not:

```
<sequence>
   <element name="a" type="string"/>
   <element name="a" type="int"/>
</sequence>
```

Likewise, the following is valid:

```
<sequence>
 <element name="a" type="string"/>
 <sequence name="c">
     <element name="a" type="string"/>
     <element name="c" type="int"/>
 </sequence>
 <element name="b" type="string"/>
</sequence>
```

---

8. Elementtype inheritance

As in object-oriented inheritance, an element may specialize (or subclass) from another element by inheriting its structure and then adding on to its content model. Inheritance is specified using the *extends* construct, and the relevant DTD fragment is:

```
<!ELEMENT extends (append?, attdef*)>
<!ATTLIST extends
                  prefix NMTOKEN #IMPLIED
                  type     NMTOKEN #REQUIRED >
<!ELEMENT append (element|choice|sequence)+>
```

The *type* attribute refers to the base element type that is being extended, and the structure of the *append* atom has the same contents as that of *model* . The base type must be already defined. The contents of the *append* element are added to the end of the parent's content model (the outermost *sequence*). Note that the *append* element has been made optional. This makes it possible to declare semantically distinct element types whose structures remain the same as that of some common parent.

In the following example, the element type *datednote* has the content model of the element type it extends (*note*) with an appended date (using the intrinsic date datatype). The *multinote* element type polymorphyically can use either.

```
<elementtype name="note">
```

```
   <model><element type="p" occurs="+"></model>
</elementtype>

<elementtype name="datednote">
     <extends type="note">
         <append>
             <element type="date" name="adate">
             <element type="time" name="atime" occurs="?"/>
           </append>
     </extends>
</elementtype>

<elementtype name="multinote">
    <element type="note" occurs="+"/>
</elementtype>
```

The following is a valid instance of *multinote:*

```
<multinote>
    <note><p>This is a plain note</p></note>
    <datednote>
        <p>This is a dated note</p>
        <adate>19981209</adate>
        <atime>10:23:32</atime>
    </datednote>
</multinote>
```

8.1 Interaction of namespaces and extension in document instances

SOX permits an elementtype in one namespace to extend an elementtype in another Schema. In order to work correctly with the current draft of the W3C Namespace Recommendation, each element in an instance belongs to the namespace in which it was declared. In the above example, if *note* were declared in schema *Foo.sox* and both *datednote* and *multinote* in *Bar.sox* , then the following would be an appropriately prefixed version of the above example:

```
<bar:multinote xmlns:foo="Foo.sox" xmlns:bar="Bar.sox">
    <foo:note><foo:p>This is a plain note</foo:p></foo:
note>
    <bar:datednote>
        <foo:p>This is a dated note</foo:p>
        <bar:adate>19981209</bar:adate>
        <bar:atime>10:23:32</bar:atime>
    </bar:datednote>
</bar:multinote>
```

A prefered solution would be to consider local names as being in the namespace of the elementtype they are declared in (or its subtypes) and therefore need not be

prefixed. This would be similar to the treatment of attributes.

8.2 Validity issues

In order to support extensibility, each elementtype must be either a choice or a sequence. By default:

1. an *empty* elementtype is considered to be an empty sequence.
2. An elementtype with a model of just *element* is considered to be a sequence.
3. An elementtype with a model of *string* is considered to be a choice.

There are some constraints placed on the form of content models to avoid ambiguous models and interminable documents. These are an extension of similar restriction in XML 1.0.

For any possible path in the parse tree from an element to a descendant of itself there must be an intervening optional node (?, *, or indefinite occurs) or intevening choice node with at least two children. This assures that infinite documents are not required.

For any optional node in the parse tree (?, *, +, indefinite occurs, or choice), none of the descendants of elements in its first set may be in its follow set.

These conditions continue to hold when extending an elementtype.

Due to the desire to maintain substitutability of extended elementtypes with their base type, SOX 2.0 does not allow the extension of elementtypes whose content model is choice. In order to maintain substitutability, any element extending a choice would need to be a subtype of something already in the choice, so it would already be valid in the parent type.

When extending a sequence with a occurs of indefinite extent, the resulting content model must be checked for ambiguity with itself. In other words, if the parent model was *x** , where *x* is some content model, then the content model *xx* (*x* followed immediately by *x*) must have been unambiguous. If the child model is now *x*+*e* (i.e., *x* plus some additional element), then the content model *(x+e)(x+e)* must still be unambiguous.

Within an *element,* the value of the *name* attribute is scoped to the surrounding *elementtype.* It neither creates nor prevents the creation of a top level definition using the same name. However, in order to maintain backwards compatibility with XML 1.0, the binding between *name* and *type* must be global. In other words, once a name has been used with a particular type, it cannot be used with another type. We expect this restriction to be relaxed in a future version.

---

9. Datatypes

Along the lines of being able to define element types, XML Schema provides the means to define and refer to datatypes. XML Schema defines a set of intrinsic datatypes, listed below, from which user-defined datatypes may be derived. This list is derived from existing ISO standards [ISO-31, ISO-8601, DATETIME] and common programming language practice. Some of these have been refered to elsewhere in this document. It also (currently) provides the *datatype* element for actually defining new datatypes. The appropriate DTD fragment is:

```
<!ELEMENT datatype (explain?, (enumeration|scalar|
varchar)) >
<!ATTLIST datatype
                    name NMTOKEN #REQUIRED >
```

The value of the name attribute specifies the name of the new datatype. This must be unique across all the datatypes defined for this schema. The three operators, *enumeration* , *scalar* , and *varchar* (all further described below), each derive a new datatype from an existing datatype. The existing datatype can be either one of the intrisic types, or some other user-defined datatype, whether in this schema or another. Both *scalar* and *varchar* have some restrictions on which datatypes they can extend, as described below.

9.1 Intrinsic datatypes

The intrinsic datatypes define the domains of the atomic data units in XML Schema documents. The list below includes those datatypes defined intrinsic to this version of the Schema language.

boolean
: A value of either "true" or "false"

string
: A sequence of characters.
Format: X*

URI
: A sequence of characters forming a valid URI:

number
: An infinite precision number.
Format: [+-][0-9]*[`.'[0-9]*]?

float
: A single precision floating point number in the range -3.40282347 * 10E38 to 3.40282347 * 10E38
Format: [+-][0-9]*['.'[0-9]*]?

double
: A double precision floating point number in the range -1.17549435 * 10E308 to 1.17549435 * 10E308
Format: [+-][0-9]*['.'?[0-9]*]?

int
: An integer in the range -2,147,483,648 to 2,147,483,647
Format: [+-]?[0-9]*

long

An integer in the range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Format: [+-]?[0-9]*

byte

An integer in the range -128 to 127
Format: [+-]?[0-9]*

ID

An ID is a name which must be unique within the instance and names the node for which it is defined. The value of an ID cannot be reused within the instance.
Format: NMTOKEN production of XML 1.0 specification.

IDREF

An IDREF must contain the value of an ID string declared elsewhere in the document.

IDREFS

A list of whitespace delimited IDREF values.

NMTOKEN

A string corresponding to the values of the NMTOKEN production of the XML 1.0 specification.

NMTOKENS

A list of whitespace delimited NMTOKEN values.

date

A date including month, day, and year
Format: YYYYMMDD

time

A time accurate to the nearest second with optional offset from GMT
Format: hh:mm:ss[[+-]hh:mm]?

datetime

A combination Date and Time. Note the presence of a "T" character between the date and time portions, and the use of colons to separate hours, minutes, and seconds. These are as per ISO 8601.
Format: YYYYMMDDThh:mm:ss[[+-]hh:mm]?

9.2 Enumeration datatypes

XML Schema documents provide a mechanism for defining enumerations to constrain attribute or element string content. An *enumeration* datatype is a finite set of values enumerated by the *option* elements inside the *enumeration* element.

```
<!ELEMENT enumeration (explain?, option)+ >
<!ATTLIST enumeration
                      prefix   NMTOKEN #IMPLIED
                      datatype NMTOKEN #REQUIRED >

<!ELEMENT option (#PCDATA)* >
```

The *datatype* attribute of the enumeration specifies the instrinsic (see list above) or user-defined datatype (i.e., other *enumeration* , *varchar* , or *scalar*) being refined. If the datatype is not defined in this Schema, then the prefix of the appropriate schema must be specified.

Each *option* has a value representing a valid value for the *datatype* being extended.

The following example demonstrates the definition and use of an enumeration datatype:

```
<datatype name="colortype">
     <enumeration datatype="NMTOKEN">
          <option>Red</option>
          <option>Blue</option>
          <option>Green</option>
     </enumeration>
</datatype>

<elementtype name="car">
       <empty/>
       <attdef name="color"datatype="colortype"><required/
></attdef>
</elementtype>

<elementtype name="bus">
   <model>
       <element name="color" type="colortype">
   </model>
</elementtype>
```

The following are valid instances:

```
<car color="Red"/>

<bus><color>Blue</color></bus>
```

9.3 Scalar datatypes

Scalar datatypes are used for creating subtypes of *number* . The relevant DTD fragment is:

```
<!ELEMENT scalar EMPTY >
<!ATTLIST scalar
                 prefix NMTOKEN #IMPLIED
                 datatype NMTOKEN "number"
                 digits CDATA #IMPLIED
                 decimals CDATA #IMPLIED
                 minvalue CDATA #IMPLIED
                 maxvalue CDATA #IMPLIED
                 minexclusive (true | false) "false"
                 maxexclusive (true | false) "false" >
```

Digits specifies the maximum number of digits for the integral part of the number, *decimals* specifies the maximum number of digits for the fraction part of the number.

The following constraints must hold:

- If the datatype is a subtype of *int* , *long* , or *byte* , then *decimals* must be unspecified or 0.
- *Minvalue* specifies the low end of the range of possible values. *Maxvalue* specifies the high end. *Minvalue* must be less than or equal to *maxvalue* .
- If *minvalue* (respectively, *maxvalue*) is unspecified, then it is either the minimum (resp. maximum) possible value given the specified values of digits and decimals, or it is the value specified by the parent datatype Both *minvalue* and *maxvalue* must be specifiable with the given values for *digits* and *decimals* , or must be expressible as proper values for the parent class.
- *Minexclusive* and *maxexclusive* determine whether the minimum number, respectively largest number, in the range is included.
- The values for *digits* and *decimals* must be non-negative integers. Both must be less than or equal to the values specified (if any) for the parent class.
- The value of the datatype must be either one of the intrinsic scalar datatypes (*byte* , *long* , *int* , *float* , *double* , *number*), or of a scalar datatype derived from one of these intrinsics.

An example scalar would be:

```
<scalar digits="4" decimals="3" minvalue="-9999"
maxvalue="8888" minexclusive="true"/>
```

The following are valid: -9998.999, 8887.999, 0.0, 8888.

The following are invalid: -9999, 8888.001.

The *number* datatype covers all rational numbers which can be finitely specified as a decimal (i.e., it does not cover infinitely repeating decimals, such as 1/9, or irrational reals, such as *p).* The other intrinsic classes are designed to be easily mappable to existing datatypes in common programming languages.

9.4 Varchar types

Varchar , adapted from SQL[SQL], is for specifying string types with fixed maximum length.

```
<!ELEMENT varchar EMPTY>
<!ATTLIST varchar
                  prefix NMTOKEN #IMPLIED
                  datatype NMTOKEN "string"
                  maxlength CDATA #REQUIRED>
```

The value of *maxlength* must be a non-negative integer.

An example varchar use is:

```
<datatype name="var">
    <varchar maxlength="4"/>
</datatype>

<elementtype name="wrap">
    <model>
        <string datatype ="var"/>
    </model>
</elementtype>
```

A valid instance would be:

```
<wrap>abc</wrap>
```

as would:

```
<wrap>abcd</wrap>
```

An invalid instance would be:

```
<wrap>abcde</wrap>
```

9.5 Validity issues

Float , *double* , *int* , *long* , and *byte* are all predefined scalar types. This means they can be used as base types for any new definition of a *scalar* and can be referenced as datatypes. The value of *maxlength* in varchar must be greater than or equal to 0. Note that a length of 0 implies an empty string. The datatype of a *varchar* must be either string or another *varchar* or one of *string, NMTOKEN*, *NMTOKENS*, *ID*, *IDREF*, or *IDREFS*.

The names of the intrinsic types are reserved by SOX. It is an error to define a datatype or elementtype using the name of one of the intrinsic types.

---

10. Attribute definitions

Attribute definitions in XML Schema documents may be defined as part of the element type definition. An attribute definition has a name and a type, and must include a presence element. The relevant fragment of the DTD is:

```
<!ELEMENT attdef (explain?,(enumeration | scalar |
varchar)?,
                        (required|implied|default|fixed)?)>
<!ATTLIST attdef
                  name     NMTOKEN #REQUIRED
                  prefix   NMTOKEN #IMPLIED
```

```
                    datatype NMTOKEN #IMPLIED>
```

The name of the attribute is defined by the value of *name* , and it may be of a certain *datatype* . It must be unique among the attributes for this elementtype. If a value is not given for the *datatype* attribute and the *attdef* does not contain an *enumeration* , *scalar* , or *varchar* , then a datatype of *string* is assumed.

An attribute value's presence in an instance may be specified as *default* , *fixed* , *required* or *implied,* as in the XML specification. If no value is given for the presence, then it defaults to *implied* . In all cases, the value of an attribute must be valid for its datatype. The DTD fragment for the presence elements is:

```
<!ELEMENT default (#PCDATA) >
<!ELEMENT fixed   (#PCDATA) >
<!ELEMENT required EMPTY >
<!ELEMENT implied  EMPTY >
```

A *default* presence indicates that the value of the attribute is automatically set to the default value if none is specified. In an instance, if the attribute is defined to have another value, the default is ignored.

A *fixed* presence indicates that the value of the attribute is assumed to be the fixed value if no value is specified. The attribute may also be explicitly assigned exactly this fixed value. In an instance, if the attribute is defined to have a different value, this signals a fatal error.

An empty value can be specified for *default* or *fixed.*

A *required* presence indicates that in an instance, whenever the parent element appears, this attribute must be assigned a value.

And finally, an *implied* presence indicates that the application is given the responsibility of filling in a default value if no attribute value is defined in an instance.

In XML Schema documents, unlike XML DTDs, enumerations may be specified for any attribute type. This information will be lost if an XML DTD is generated from an XML Schema document, except for attributes of type *NMTOKEN* , indicating a name token. If there is an *enumeration* specified for an attribute and also a *fixed* or *default* value, then that *fixed* or *default* value must be a member of the *enumeration* .

Thus, example attribute definitions (within the definition of an *elementtype* , of course) might be:

```
<elementtype name="car">
    <empty/>
    <attdef name="owner" datatype="string"/>
    <attdef name="color">
        <enumeration datatype="NMTOKEN">
            <option>Red</option>
```

```
                <option>Blue</option>
                <option>Green</option>
            </enumeration>
            <required/>
        </attdef>
    </elementtype>
```

An instance corresponding to this definition would look like:

```
<car color="Blue" owner="John Smith"/>
```

10.1 Validity constraints

It is a fatal error for the *datatype* attribute of the *attdef* element to have a value if there is an enclosed *enumeration* , *scalar* , or *varchar.*

---

11. Including schema files

An externally-defined schema file whose definitions belong to the same namespace may be pulled in and parsed with the current schema definition. This is accomplished using the *join* construct. The relevant fragment of the DTD is:

```
<!ELEMENT join (explain?)>
<!ATTLIST join
                datatype NMTOKEN #FIXED "schema"
                public CDATA #IMPLIED
                system CDATA #REQUIRED>
```

The *datatype* attribute is fixed as only schemas may be included for now. The *public* attribute is the public identifier of the file as defined in the XML 1.0 specification. The *system* attribute is the URI of the file containing the schema definition to be lexically included. The entity manager must resolve this URI.

A *join* ed file is read only once by the parser. The parser determines identity between files by comparison of the values of the *system* attribute. It is not illegal for two *join* elements to reference the same URI, but one will be ignored. It is a user error to use two different URIs which ultimately map to the same file. Note that reading a fragment twice will cause an error as the schema processor will create all its definitions twice.

11.1 Validity issue

The joined file must belong to the same namespace as the joining one, as identified by the URI attribute of the root elements.

In the current implementation, both *namespace* and *join* elements use URIs to point to external files, and processing a namespace involves retrieving the physical file

referenced by the *namespace* element. It is not an error for a *join* element to reference this file again, however it is a runtime error for that file to actually be retrieved a second time. In other words, no fragment for a schema is to be processed more than once.

Appendix A: XML Schema DTD

```
<!-- ********************************************************* -->

    <!-- XML Schema DTD -->
    <!-- PUBLIC "-//Commerce One Inc.//DTD XML Schema 2.0//EN"
    -->
    <!-- SYSTEM "schema.dtd" -->

    <!-- Copyright: Commerce One Inc., 1997, 1998, 1999

     Date created: 17 Dec 1997
     Date revised: 03 June 1999
     Version: 2.0
    -->

    <!--
    ************************************************************
    -->

    <!--
    ************************************************************
    -->

    <!-- XML Schema
    ********************************************** -->

    <!--
    ************************************************************
    -->

    <!ENTITY % htmltext SYSTEM "htmltext.ent">
    %htmltext;

    <!ELEMENT schema (intro?, (datatype | elementtype | join |
    comment | namespace)*) >
    <!ATTLIST schema
                    prefix NMTOKEN #IMPLIED
                    uri CDATA #REQUIRED
                    soxlang-version NMTOKEN #FIXED "V0.2.2">

    <!--
```

```
********************************************************
-->
<!-- ELEMENTS
******************************************************* -->
<!--
********************************************************
-->

<!-- An Element Type definition requires a name.
  It is defined to extend a named element,
  as an instance of a named element,
  as an EMPTY or ANY element with optional attribute
definitions,
  or with a content model with optional attribute
definitions. -->

<!ELEMENT elementtype (explain?, (extends | ((empty|
model), (attdef)*)))>
<!ATTLIST elementtype
                name NMTOKEN #REQUIRED >

<!ELEMENT empty EMPTY >

<!--
********************************************************
-->
<!-- MODEL
******************************************************** -->
<!--
********************************************************
-->

<!ELEMENT model (string|element|choice|sequence)>

<!ELEMENT extends (append?, attdef*)>
<!ATTLIST extends
                prefix NMTOKEN #IMPLIED
                type NMTOKEN #REQUIRED >

<!ELEMENT append (element|choice|sequence)+>
<!ELEMENT element EMPTY >
<!ATTLIST element
                prefix NMTOKEN #IMPLIED
                type NMTOKEN #REQUIRED
                name NMTOKEN #IMPLIED
                occurs CDATA #IMPLIED >

<!ELEMENT string EMPTY >
<!ATTLIST string
                prefix NMTOKEN #IMPLIED
```

```
                datatype NMTOKEN "string" >
<!ELEMENT choice ((element|choice|sequence),
                  (element|choice|sequence)+) >
<!ATTLIST choice
                name NMTOKEN #IMPLIED
                occurs CDATA #IMPLIED >

<!ELEMENT sequence ((element|choice|sequence),
                    (element|choice|sequence)+) >
<!ATTLIST sequence
                name NMTOKEN #IMPLIED
                occurs CDATA #IMPLIED >

<!-- replacement for "include" -->
<!ELEMENT join (explain?)>
<!ATTLIST join
                datatype NMTOKEN #FIXED "schema"
                public CDATA #IMPLIED
                system CDATA #REQUIRED>

<!--
************************************************************
-->
<!-- ATTRIBUTES
********************************************** -->
<!--
************************************************************
-->

<!-- An attribute definition has a name and datatype, and
must have a
 presence element "required|implied|default|fixed"
included.
 It may have a namespace associated with it, or inherit
 enumeration is supposed to define the domain of
acceptable value -->

<!ELEMENT attdef (explain?, (enumeration | scalar |
varchar)?,
                (required|implied|default|fixed)?)>
<!ATTLIST attdef
                name NMTOKEN #REQUIRED
                prefix NMTOKEN #IMPLIED
                datatype NMTOKEN #IMPLIED >

<!ELEMENT default (#PCDATA) >
<!ELEMENT fixed   (#PCDATA) >
<!ELEMENT required EMPTY >
<!ELEMENT implied  EMPTY >
```

```
<!--
*********************************************************
-->
<!-- DATATYPE
****************************************************** -->
<!--
*********************************************************
-->

<!ELEMENT datatype (explain?, (enumeration|scalar|
varchar)) >
<!ATTLIST datatype
                name NMTOKEN #REQUIRED >

<!ELEMENT enumeration (explain?, option)+ >
<!ATTLIST enumeration
                prefix NMTOKEN #IMPLIED
                datatype NMTOKEN #REQUIRED >

<!ELEMENT option (#PCDATA)* >

<!ELEMENT scalar EMPTY >
<!ATTLIST scalar
                prefix NMTOKEN #IMPLIED
                datatype NMTOKEN "number"
                digits CDATA #IMPLIED
                decimals CDATA #IMPLIED
                minvalue CDATA #IMPLIED
                maxvalue CDATA #IMPLIED
                minexclusive (true | false) "false"
                maxexclusive (true | false) "false" >

<!ELEMENT varchar EMPTY>
<!ATTLIST varchar
                prefix NMTOKEN #IMPLIED
                datatype NMTOKEN "string"
                maxlength CDATA #REQUIRED>

<!--
*********************************************************
-->
<!-- COMMENT
****************************************************** -->
<!--
*********************************************************
-->

<!ELEMENT comment (#PCDATA)>

<!-- Namespaces -->
```

```
<!ELEMENT namespace (explain?) >
<!ATTLIST namespace
                prefix NMTOKEN #REQUIRED
                namespace CDATA #REQUIRED >
```

Appendix B: htmltext.ent - HTML element types for explain

```
<!--
************************************************************
-->
<!-- HTML Text: SOX uses HTML element types for
convenience.       -->
<!--
************************************************************
-->
<!-- Copyright: Commerce One Systems Inc., 1997, 1998
     Date created: 17 Dec 1997
     Date revised: 01 Mar 1999
     Version: 1.0 -->
<!--
************************************************************
-->

<!ENTITY % html.nonheading " table | p | bq | pre | ol |
ul | dl" >

<!ENTITY % html.text "#PCDATA| a | abbr | b | big | br
                      | cite | code | em | i | img
                      | q | small | span | strike | strong
                      | sub | sup | tt | u " >

<!ENTITY % html.heading.text
                     "#PCDATA| a | abbr | b | big | br
                      | cite | code | em | i | img
                      | q | small | span | strike | strong
                      | sub | sup | tt | u " >


<!ENTITY % html.block "h1|h2|h3|h4|h5|DT|%html.
nonheading;" >

<!--
************************************************************
-->
<!-- The intro element type is used to introduce a schema.
It contains a
```

```
 general description of the purpose and use of the
schema's document
 type or components. -->

<!ELEMENT intro ((%html.block;)+) >

<!-- The explain element is use to document a component
within a schema. -->

<!ELEMENT explain (title?, synopsis?, (%html.block;)+) >

<!-- The title is used to give a human readable title to
some type name. -->

<!ELEMENT title (%html.text;)* >

<!-- The synopsis is used to give a purpose or synopsis to
the thing being
 explained. It is a single paragraph of text. -->

<!ELEMENT synopsis (%html.text;)* >

<!--
************************************************************
-->

<!ELEMENT h1 (%html.heading.text;)* >
<!ELEMENT h2 (%html.heading.text;)* >
<!ELEMENT h3 (%html.heading.text;)* >
<!ELEMENT h4 (%html.heading.text;)* >
<!ELEMENT h5 (%html.heading.text;)* >
<!ELEMENT DT (%html.heading.text;)* >

<!--
************************************************************
-->

<!ELEMENT b      (#PCDATA)* >
<!ELEMENT br     EMPTY >
<!ELEMENT big    (#PCDATA)* >
<!ELEMENT i      (#PCDATA)* >
<!ELEMENT small  (#PCDATA)* >
<!ELEMENT sub    (#PCDATA)* >
<!ELEMENT sup    (#PCDATA)* >
<!ELEMENT strike (#PCDATA)* >
<!ELEMENT tt     (#PCDATA)* >
<!ELEMENT u      (#PCDATA)* >

<!ELEMENT abbr   (#PCDATA)* >
<!ELEMENT cite   (#PCDATA)* >
```

```
<!ELEMENT code   (#PCDATA)* >
<!ELEMENT em     (#PCDATA)* >
<!ELEMENT q      (#PCDATA)* >
<!ELEMENT span   (#PCDATA)* >
<!ELEMENT strong (#PCDATA)* >

<!--
**************************************************************
-->

<!ELEMENT a (%html.text;)* >
<!ATTLIST a
                name  CDATA #IMPLIED
                href  CDATA #IMPLIED
                title CDATA #IMPLIED >

<!--
**************************************************************
-->
<!ELEMENT img (explain?) >
<!ATTLIST img
                src      CDATA #REQUIRED
                alt      CDATA #REQUIRED
                longdesc CDATA #IMPLIED
                usemap   CDATA #IMPLIED >

<!--
**************************************************************
-->
<!ELEMENT pre (%html.text;)* >
<!ATTLIST pre
                xml:space (preserve) #REQUIRED >

<!--
**************************************************************
-->

<!ELEMENT p (%html.text;)* >
<!ELEMENT bq (%html.text;)* >

<!ELEMENT ol (lh?, li+) >
<!ELEMENT ul (lh?, li+) >
<!ELEMENT lh (%html.heading.text;)* >
<!ELEMENT li (%html.text;|%html.block;)* >

<!ELEMENT dl (dh?,(dt,dd)+) >
<!ELEMENT dh (%html.heading.text;)* >
<!ELEMENT dt (%html.text;|%html.block;)* >
<!ELEMENT dd (%html.text;|%html.block;)* >
```

```
<!--
**********************************************************
-->

<!ELEMENT table
                (thead?, tbody) >
<!ATTLIST table
                cols   CDATA #IMPLIED
                width  CDATA #IMPLIED
                height CDATA #IMPLIED
                align  (left|center|right|justify) #IMPLIED
                valign (top | middle | bottom | baseline)
#IMPLIED
                vspace CDATA #IMPLIED
                hspace CDATA #IMPLIED
                cellpadding CDATA #IMPLIED
                cellspacing CDATA #IMPLIED
                border  CDATA #IMPLIED
                frame   (box|void|above|below|hsides|
vsides|lhs|rhs) #IMPLIED
                rules (none|groups|rows|cols|all) #IMPLIED
>

<!ELEMENT thead
                (tr)+ >
<!ATTLIST thead
                align  (left|center|right|justify) #IMPLIED
                valign (top|middle|bottom|baseline)
#IMPLIED >

<!ELEMENT tbody
                (tr)+ >
<!ATTLIST tbody
                align  (left|center|right|justify) #IMPLIED
                valign (top|middle|bottom|baseline)
#IMPLIED >

<!ELEMENT tr (th | td)+ >
<!ATTLIST tr
                align  (left|center|right|justify) #IMPLIED
                valign (top | middle | bottom | baseline)
#IMPLIED >

<!ELEMENT th (%html.text;|%html.block;)* >
<!ATTLIST th
                colspan CDATA #IMPLIED
                rowspan CDATA #IMPLIED
                width   CDATA #IMPLIED
                height  CDATA #IMPLIED
                align   (left|center|right|justify)
```

```
    #IMPLIED
                    valign  (top | middle | bottom | baseline)
    #IMPLIED >

    <!ELEMENT td (%html.text;|%html.block;)* >
    <!ATTLIST td
                    colspan CDATA #IMPLIED
                    rowspan CDATA #IMPLIED
                    width   CDATA #IMPLIED
                    height  CDATA #IMPLIED
                    align   (left|center|right|justify)
    #IMPLIED
                    valign  (top | middle | bottom | baseline)
    #IMPLIED >

    <!--
    ********************************************************************
    -->
```

Appendix C: References

[DATETIME]
Date and Time Formats, Misha Wolf and Charles Wicksteed. W3C, 15 September 1997.
See http://www.w3.org/TR/NOTE-datetime-970915

[DCD]
Document Content Description for XML (DCD), Tim Bray et al. W3C, 10 August 1998
See http://www.w3.org/TR/NOTE-dcd

[HTML-4]
HTML 4.0 Specification, Dave Raggett et al. W3C, 1998
See http://www.w3.org/TR/REC-html40

[ISO-31]
ISO 31 -- Quantities and units. International Organization for Standardization

[ISO-8601]
ISO 8601 -- Date and Time. International Organization for Standardization

[ISO-10646]
ISO10646 -- Information Technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane, ISO/IEC 10646-1:1993. The current specification also takes into consideration the first five amendments to ISO/IEC 10646-1:1993.

[JAVA]
The Java Language Specification, James Gosling et al., Addison-Wesley, Reading, Mass., 1996
See also http://java.sun.com/docs/books/jls/index.html

[RFC-1808]
RFC 1808, Relative Uniform Resource Locators. Internet Engineering Task

Force.
See http://www.ietf.org/rfc/rfc1808.txt

[SQL]
ANSI/NCITS X3.135-1992(R1998), Information Systems - Database Language - SQL (includes ANSI X3.168-1989), National Committee for Information Technology Standards, 1992

[URI]
ID, Uniform Resource Identifiers (URI): Generic Syntax and Semantics
See http://www.ietf.org/rfc/rfc2396.txt

[URL]
RFC 1738, Uniform Resource Locators (URL). Internet Engineering Task Force.
See http://www.ietf.org/rfc/rfc1738.txt

[URN]
RFC 2141, URN Syntax. Internet Engineering Task Force.
See http://www.ietf.org/rfc/rfc2141.txt

[XML]
Extensible Markup Language (XML) 1.0, Tim Bray, et al. W3C, 10 February 1998
See http://www.w3.org/TR/REC-xml

[XML-Data]
XML-Data, Andrew Layman, et al. W3C, 05 January 1998.
See http://www.w3.org/TR/1998/NOTE-XML-data-0105

[XML-Namespaces]
Namespaces in XML, Tim Bray et al. W3C, 1998
See http://www.w3.org/TR/WD-xml-names

[XSDL]
XML Schema Part 1: Structures, David Beech et al.
See http://www.w3.org/TR/xmlschema-1/

Books

The Java Language Specification

Printable Page

The Java Language Specification, Second Edition - Written by the inventors of the technology, this book is the definitive technical reference for the Java programming language. If you want to know the precise meaning of the language's constructs, this is the source for you.

The book provides complete, accurate, and detailed coverage of the syntax and semantics of the Java programming language. It describes all aspects of the language, including the semantics of all types, statements, and expressions, as well as threads and binary compatibility.

James Gosling is a Fellow and Vice President at Sun Microsystems, the creator of the Java programming language, and one of the computer industry's most noted programmers. He is the 1996 recipient of Software Development's "Programming Excellence Award." He previously developed NeWS, Sun's network-extensible window system, and was a principal in the Andrew project at Carnegie Mellon University, where he earned a Ph.D. in Computer Science.

Bill Joy is founder and Vice President of Research at Sun Microsystems, where he has led the company's technical strategy, working on both hardware and software architecture. He is well known as the creator of the Berkeley version of the UNIX operating system, for which he received a lifetime achievement award from the USENIX Association in 1993. He received the ACM Grace Murray Hopper Award in 1986. Joy has had a central role in shaping the Java programming language.

Guy L. Steele Jr. is a Distinguished Engineer at Sun Microsystems, where he is responsible for the specification of the Java programming language as well as research in parallel algorithms. He is well known as the co-creator of the Scheme programming language and for his reference books for the C programming language (with Samuel Harbison) and for the Common Lisp programming language. Steele received the ACM Grace Murray Hopper Award in 1988 and was named an ACM Fellow in 1994.

Gilad Bracha is Computational Theologist at Sun Microsystems, and a researcher in the area of object-oriented programming languages. Prior to joining Sun, he worked on Strongtalk, the Animorphic Smalltalk System. He holds a B.Sc. in Mathematics and Computer Science from Ben Gurion University in Israel and a Ph.D. in Computer Science from the University of Utah.

- View HTML
- Download HTML FTP | HTTP (tar.Z, *~573K*)
- Download HTML FTP | HTTP (zip, *~409K*)
- Download PDF (zip, *~4420K*)
- Order this book through
  - DigitalGuru
  - amazon.com

Please send comments and errata to the authors using our feedback form.
For the latest updates and proposals, see the Java Language Specification maintenance information.

The Java Language Specification (First Edition) - With the publication of this book, James Gosling, Bill Joy, and Guy Steele provide the definitive technical reference for the Java programming language. It provides complete, accurate, and detailed coverage of the entire language and its syntax. If you want to know the precise meaning of Java's constructs, this is the source. Published in 1996 by Addison-Wesley.

- Book Description
- Table of Contents
- Order this book through
  - DigitalGuru
  - amazon.com

View HTML
Download HTML (tar.Z, *~865K*)
Download HTML (zip, *~600K*)
Download PDF (*~3706K*)
Download PostScript (tar.Z, *~1765K*)
Download PostScript (zip, *~1267K*)

Changes for Java 1.1 - Originally published as Appendix D from *The Java Programming Language* by Ken Arnold.

Clarifications and Amendments to *The Java Language Specification* (First Edition).

ENDORSED BY

SAP Markets

XML COMMON BUSINESS LIBRARY

xCBL 4.0 Final Release Now Available

xCBL 4.0 is now available as a final release. Click here to find out more and download xCBL 4.0 schemas and resources.

What is xCBL?

Welcome to xCBL.org - your source for the latest information about the XML Common Business Library (xCBL). xCBL is the pre-eminent XML component library for business-to-business e-commerce. This standard is created, maintained, and supported for use - free of charge - by anyone needing document definitions for their e-commerce applications.

xCBL is a set of XML business documents and their components, distributed freely here on xCBL.org. In addition to using the business documents provided, you can also use the component library to build your own documents. Either way, using xCBL promotes interoperability between applications.

xCBL is made available in numerous formats: xCBL 4.0 is available as W3C XML Schema only. But previous versions are supplied as a set of SOX schemas (SOX is the schema language Commerce One created and used in its own products), as W3C XML Schema, in XML DTD form, and as a set of XDR schemas (with a separate set designed for use within the BizTalk environment, see the XDR page for details). Regardless of which type of structure validation syntax you use inside your applications, the business documents - with very minor modifications - remain the same. One of the hardest parts of achieving interoperability is enabling the exchange of business documents, and xCBL is your solution to this problem.

xCBL 4.0

The newest version of xCBL - xCBL 4.0 is the first release of xCBL using XSDL schema language as the canonical form. This will be the standard for future releases of xCBL. Previously, all xCBL releases were based on SOX schema. xCBL 4.0 also represents an initial alignment with the OASIS Universal Business Language (UBL) initiative. Some of the UBL recommendations have been adopted in the design of xCBL 4.0. As UBL continues to evolve and mature, additional recommendations and standards will be adopted by xCBL. With a few exceptions, business documents that existed in xCBL 3.5 have been carried over to xCBL 4.0. New documents have also been added to xCBL 4.0 to support integration of applications with backend ERP systems.

xCBL 3.5

xCBL 3.5 contains all of the documents found in xCBL 3.0 as well as 9 new xCBL documents. Some of the xCBL 3.0 documents have been

Latest Updates

June 2003

Order Management Choreographies	All versions

March 2003

xCBL 4.0 Final Release	xCBL 4.0
xCBL 4.0 release notes	xCBL 4.0

September 2002

xCBL 3.5 XSDL namespace change	xCBL 3.5 XSD
xCBL 3.0 XSDL namespace change	xCBL 3.0 XSD

August 2002

xCBL 3.5 For Content: ProductCatalog Specification	xCBL 3.5 Documentation

May 2002

SOX-XDR datatype mapping	Development Resources

modified in response to new requirements and correction of bugs. To maintain interoperability between xCBL3.5 and xCBL3.0, the only modifications allowed to an xCBL 3.0 document were the addition of new optional elements and additions to code lists. An xCBL 3.0 instance of a document is also a valid instance in xCBL 3.5. New documents for xCBL 3.5 are in the areas of: Sourcing, Informational, Order Management and Material Management.

xCBL 3.0

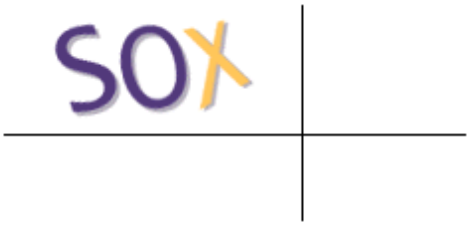
xCBL 3.0 was announced at the eLink conference in Hong Kong on November 29, 2000. It covers a wide variety of B2B e-commerce scenarios, in both the MRO/Indirect Procurement and Direct Goods arenas.

xCBL 2.0

xCBL 2.0 is made up of MRO documents used today in many applications.

HOME | ABOUT xCBL | xCBL4.0 | xCBL3.5 | EARLIER VERSIONS | DEVELOPMENT RESOURCES | FAQ | LINKS | CONTACT US

For problems or questions regarding this site, contact xcblwebmaster@commerceone.com.



ENDORSED BY

SAP Markets

Schema for Object-Oriented XML (SOX)

NOTE: *these tools are for use with early versions of xCBL and are provided here merely for users of those older versions. If you are new to xCBL it is recommended that you use the later W3C XML Schema based versions with the DocSOAPXDK toolkit (Click here for details).*

SOX was created by Commerce One in anticipation of the coming W3C standard XML Schema Language. SOX leverages the object-relationships between data structures to allow for the easy management of a component library, and the use of type relationships within schema-based e-commerce applications. It also provides strong datatyping capabilities, like other XML Schema languages.

To learn how to use SOX, please see the SOX Tutorial. (For more high-level information, please see the "About" page below.)

About SOX

FAQ

Distribution

Documentation

For problems or questions regarding this site, contact xcblwebmaster@commerceone.com.

NOTE-dcd-19980731

Document Content Description for XML

Submission to the World Wide Web Consortium 31-July-1998

This version:
http://www.w3.org/TR/1998/NOTE-dcd-19980731
http://www.w3.org/TR/1998/NOTE-dcd-19980731.html
Latest version:
http://www.w3.org/TR/NOTE-dcd
Editors:
Tim Bray (Textuality) <tbray@textuality.com>
Charles Frankston (Microsoft) <cfranks@microsoft.com>
Ashok Malhotra (IBM) <petsa@us.ibm.com>

Status of this document

This document is a submission to the World Wide Web Consortium (see Submission Request, W3C Staff Comment). It is the initial draft of the specification of the DCD facility. It is intended for review and comment by W3C members and is subject to change.

This document is a NOTE made available by the W3 Consortium for discussion only. This indicates no endorsement of its content, nor that the Consortium has, is, or will be allocating any resources to the issues addressed by the NOTE.

Abstract

This document proposes a structural schema facility, Document Content Description (DCD), for specifying rules covering the structure and content of XML documents. The DCD proposal incorporates a subset of the XML-Data Submission [XML-Data] and expresses it in a way which is consistent with the ongoing W3C RDF (Resource Description Framework) [RDF] effort; in particular, DCD is an RDF vocabulary. DCD is intended to define document constraints in an XML syntax; these constraints may be used in the same fashion as traditional XML DTDs. DCD also provides additional properties, such as basic datatypes.

Document Content Description for XML

Version 1.0

Table of Contents

Appendices

---

1. Introduction

The Document Content Description facility for XML (abbreviated DCD) is an RDF vocabulary designed for describing constraints to be applied to the structure and content of XML documents. The abbreviation "DCD" is used to describe both the general facility described in this document and individual schema instances that conform to it.

1.1 Motivating Examples

The following example is a DCD which describes the important characteristics of the `DL` element from HTML:

```
<DCD>
  <ElementDef Type="DL" Model="Elements" Content="Closed">
    <Description>A simple 'definition list' construct, which contains paired
       'DT' (DL Term) and 'DD' (DL Definition) elements</Description>
    <Group Occurs="OneOrMore" RDF:Order="Seq">
      <Element>DT</Element>
      <Group Occurs="Optional"><Element>DD</Element></Group>
    </Group>
  </ElementDef>
  <ElementDef Type="DT" Model ="Data" Content="Closed">
    <Description>The term being defined in a DL list item</Description>
  </ElementDef>
  <ElementDef Type="DD" Model ="Mixed" Content="Open">
    <Description>A term's definition in a DL list item</Description>
    <!-- Open because lots of markup can be in a DL -->
  </ElementDef>
</DCD>
```

The example above is very document-oriented and in many respects isomorphic to what can be done with an XML DTD. The following example, less document-oriented, provides constraints for an airline booking:

```
<DCD>
 <ElementDef Type="Booking" Model="Elements" Content="Closed">
   <Description>Describes an airline reservation</Description>
   <Group RDF:Order="Seq">
     <Element>LastName</Element> <Element>FirstInitial</Element>
     <Element>SeatRow</Element> <Element>SeatLetter</Element>
     <Element>Departure</Element> <Element>Class</Element>
   </Group>
 </ElementDef>

 <!-- example omits boring field declarations -->
 <ElementDef Type="SeatRow" Model="Data" Datatype="i1" Min="1" Max="72" />
 <ElementDef Type="SeatLetter" Model="Data" Datatype="char" Min="A" Max="K"/>
 <ElementDef Type="Class" Model="Data" Datatype="char" Default="1"/>
</DCD>
```

Here is a booking record that conforms to the schema:

```
<Booking>
  <LastName>Bray</LastName><FirstInitial>T</FirstInitial>
  <SeatRow>33</SeatRow><SeatLetter>B</SeatLetter>
  <Departure>1997-05-24T07:55:00+1</Departure>
</Booking>
```

1.2 Design Principles

DCD is based on the following design principles:

1. DCD semantics shall be a superset of those provided by XML DTDs.
2. The DCD data model and syntax shall be conformant with that of RDF.
3. The constraints in a DCD shall be straightforwardly usable by authoring tools and other applications which wish to retrieve information about a document's content and structure.
4. DCD shall use mechanisms from other W3C working groups wherever they are appropriate and efficient.
5. DCDs should be human-readable and reasonably clear.

1.3 Future Work

It is anticipated that for DCD to realize its full potential, several types of constraint are required beyond those described in this note. These include:

Subclassing and Inheritance
: The creation and maintenance of document schemas is a complex and demanding task, similar in many respects to that of software engineering. Software engineering has made great progress based on object-oriented design principles, which allow the efficient re-use and customization of proven pieces of work. The same techniques should be available to the designers and maintainers of document schemas. A proposal for subclassing and inheritance is contained in Appendix "B. Inheritance and Subclassing".

Database Interface
: DCD is expected to find use in constraining XML documents that contain extracts from databases. To meet these needs, it will be necessary to add properties which describe database constraints such as the uniqueness of values, key fields and referential integrity. It will also be necessary to define datatypes that faithfully mirror database datatypes such as fixed length strings. Section "4. Datatypes" is a proposal for a specific datatype repertoire. It is anticipated that a future version of the DCD specification will include other facilities to support database interaction and that they will be conformant to applicable industry and international standards such as [SQL]

The &-Connector
: There was a request from the database community to allow `Bag` as another legal value for the `RDF:Order` property. This would support the concept that a Relational database table is an unordered collection of columns. But this would bring back the SGML &-connector and so, on the balance, it was decided (for this release) to disallow `Bag` as a legal value for the `RDF:Order` property. This decision may need to be revisited in future.

2. The DCD Framework

A Document Content Description (DCD) is a set of properties used to constrain the types of elements and names of attributes that may appear in an XML document, the contents of the elements, and the values of the attributes.

2.1 A Note on Syntax

2.1.1 Proposed Simplification of RDF Syntax

As stated earlier, it is intended that DCD be conformant to the RDF Model and Syntax Specification [RDF]. However, it assumes certain simplifications in the RDF syntax which we intend to propose to the RDF working group. This syntax will be adopted only if ratified by the RDF working group. These syntactic simplifications are:

RDF:li
: The RDF:li should not be required if typed nodes are being inserted into a collection.

Collection type
: The collection type for properties can be specified as an attribute of the node.

2.1.2 Interchangeability of Elements and Attributes

The RDF syntax document allows non-repeatable properties to be expressed as attributes of the parent element. Thus, properties such as Name, Content and Model can be expressed either as elements or as attributes. The following are, therefore, equivalent:

```
<DCD>
  <ElementDef>
    <Type>DL</Type>
    <Description>A simple 'definition list' construct, which contains paired
        "DT" (DL Term) and "DD" (DL Definition) Elements</Description>
    <Model>Elements</Model>
    ...
  </ElementDef>
</DCD>
```

```
<DCD>
  <?DCD syntax="explicit"?>
  <ElementDef Type="DL" Model="Elements">
     <Description>A simple 'definition list' construct, which contains paired
         "DT" (DL Term) and "DD" (DL Definition) Elements</Description>
      ...
  </ElementDef>
</DCD>
```

As shown in the above example, a optional processing instruction (PI) may be added to a DCD to specify the alternative "explicit" syntax form. The examples are equivalent and legal even without the PI. When the DCD PI is present with `syntax="explicit"` specified, then throughout the schema, the following properties *must* be specified using attribute syntax as shown below:

`Type`
`Model`
`Occurs`
`RDF:Order`
`Content`
`Root`

`Fixed`
`Datatype`

and all other properties *must* be specified using element syntax.

```
<?DCD syntax="explicit"?>
```

The examples in this document, for the most part, use the attribute form for properties.

2.2 DCD Nodes and Resource Types

The namespace which describes DCD properties and resources is identified by the URI `http://w3.org/Schemas/DCD`. It contains the following types: `DCD`, `ElementDef`, `Group`, `AttributeDef`, `ExternalEntityDef` and `InternalEntityDef`.

In the XML form of a DCD, the types of the elements correspond to RDF's *property types*. In the interests of brevity, we refer, for example, to "objects of type Namespace", which in the XML syntax are elements whose type is "Namespace" representing RDF properties where the property type is "Namespace".

A resource of type DCD is a document structure description that constrains the structure and contents of any document that identifies itself as falling under that DCD's constraints. An XML document can be identified as falling under the constraints of more than one DCD, in which event the properties applying to each such DCD are taken as constraints on the XML document. This provides two benefits: first, a single DCD can be used to provide constraints for large numbers of separate documents. Second, the DCD object provides a convenient level of granularity for applying namespace mechanisms.

The resources of type ElementDef and AttributeDef are more detailed structure descriptors. The properties of these resources provide constraints governing elements and attributes in the XML document. Implicitly, any node which is the value of an `ElementDef` or `AttributeDef` property is of respective type ElementDef or AttributeDef; however, there is typically no value in indicating this explicitly with an `RDF:InstanceOf` property.

2.3 Referring to Elements and Attributes

Most DCD declarations constrain the content and attributes of elements in document instances. This is done by assigning properties to objects of type `ElementDef`. These assignments may be seen as element type declarations. Element definitions declare that elements may have other elements as children, or may have attributes provided with certain names and properties. Child elements must be collected together into `Groups` which have `Order` and `Occurs` properties. See "3.2.6 Groups, Occurs and Order". Each `ElementDef` must have a `Type` property. This must be unique within the DCD. But, see Appendix "A. Local Element Definitions".

The attributes and the elements referred to in a particular DCD may come from the same DCD or from other DCDs identified by namespaces. Element definitions from within the same DCD are referred to by their `Type` property. If the element definition comes from another namespace, the value of the `Type` property may be a qualified name, where the prefix identifies the namespace.

For example, in the following, `FirstName`, `MI` and `LastName` are defined elsewhere in the DCD but `Address` comes from a namespace declared with the `common` prefix.

```
<ElementDef Type="person" Model="Elements">
  <Group RDF:Order="Seq">
    <Element>FirstName</Element>
    <Group Occurs="Optional">
      <Element>MI</Element>
    </Group>
    <Element>LastName</Element>
    <Element>common:Address</Element>
  </Group>
</ElementDef>
```

Attributes are declared in DCDs using objects of type `AttributeDef`. An attribute definition may occur on its own, as a property of the DCD, or it may occur within an element definition. In either case it may have a `Global` property whose value may be `True` or `False`. The default is `False`. Every attribute definition must have a `Name` property. If the value of the `Global` property is `True` the `Name` property must be unique in the DCD.

Global attributes can referred to by their names in any element definition within the DCD. Global attributes in other namespaces can be referred to by the use of qualified names.

In the following example, `Hidden` is a global attribute in the DCD, while `schemas:CLASS` is a global attribute from another namespace.

```
<DCD>
  <AttributeDef Name="Hidden" Default="False" Global="True" />
  <ElementDef Type="MyType" />
    <Attribute>Hidden</Attribute>
    <Attribute>schemas:CLASS</Attribute>
  </ElementDef>
</DCD>
```

In the following, the `SRC` attribute is defined locally within the `IMG` element definition.

```
<DCD>
  <AttributeDef Name="Border" Global="True">
    <!-- facts about the Border attribute -->
  </AttributeDef>
  <ElementDef Type="IMG" />
    <AttributeDef Name="SRC" Datatype="uri">
      <Description>The URI where the image may be retrieved" </Description>
    <Attribute Name="Border"/>
  </ElementDef>
</DCD>
```

Attributes defined with `Global="False"` can be referred to in other element definitions in the DCD by a resource identifier. For example:

```
<DCD>
  <AttributeDef Global="False" Name="size" Datatype="int" id="sizeAtt" />
  <ElementDef>
    <AttributeDef resource="#sizeAtt" />
  </ElementDef>
</DCD>
```

3. The DCD Vocabulary

The (roughly alphabetical) order in which the property descriptions appear is not intended to have any significance.

3.1 Properties which apply to DCDs

In the following descriptions, the phrase "such documents" signifies documents which have been identified as falling under the constraints of the DCD.

3.1.1 AttributeDef

Declares an attribute type which may be provided for one or more elements in such documents. This property does not assert that the attribute is provided for any individual element type; this can only be done with `Attribute` and `AttributeDef` properties of ElementDef. However, this property can be used to create an AttributeDef node which can serve as the value of `Attribute` properties. See discussion above.

An example of the use of `AttributeDef`:

```
<DCD>
  <?DCD syntax="explicit"?>
  ...
  <AttributeDef Name="Class" Datatype="string"/>
  ...
</DCD>
```

3.1.2 Description

Provides a, presumably human-readable, description of the semantics and usage of this DCD. The value of this property must match the production labeled `Content` in the XML specification; that is to say, it may contain markup, and is well-formed.

3.1.3 InternalEntityDef and ExternalEntityDef

Identify an entity which may be invoked via reference within such documents. The value of these properties must be a Node (in RDF terms), provided in the RDF syntax with subelement or URI. The resource which is the property value must be identified by the class mechanism as an InternalEntityDef or ExternalEntityDef.

An example of the use of `InternalEntityDef` and `ExternalEntityDef`:

```
<InternalEntityDef
  Name="W3C" Value="World Wide Web Consortium" />
<ExternalEntityDef resource='#copyrightNotice' />
```

3.1.4 Contents

Signals whether elements of types not explicitly declared via `ElementDef` properties may appear in such documents. The value of this property must be a string whose value is `Open` or `Closed`. `Closed` means that such documents may contain only elements whose types have been declared via `ElementDef` properties. `Open` means that such documents may contain elements which have not been so declared.

3.1.5 Namespace

Provides the namespace of this DCD. The value of this property must be a URI which identifies a namespace. *This property is required to exist for every DCD.*

The namespace of a DCD applies to all elements and attributes attached by properties to this DCD. The idea is that in an instance, the prefix part of a qualified name is used to locate the namespace and schema, and the local name part used to locate the applicable properties in the schema.

An example of the use of `Namespace`:

```
<DCD>
  <?DCD syntax="explicit"?>
  <Description>about HTML</Description>
  <Namespace>http://www.w3.org/TR/REC-html40</Namespace>
  <ElementDef Type="B" Model="Data"/>
</DCD>
```

This declares the namespace for this DCD to be `http://www.w3.org/TR/REC-html40`. If some XML document indicates that the prefix `H` refers to the namespace whose namespace name is `http://www.w3.org/TR/REC-html40`, then references to an element H:B in that document refer to the element defined in the above example using the local name `B`.

3.2 Properties Which Apply to Element Definitions

[Definition:] In the descriptions, the phrase **this type** signifies the element definition to which the properties apply.

3.2.1 Attribute and AttributeDef

Identify attributes which may be provided for elements of this type. No element definition may have two `Attribute` or `AttributeDef` properties referencing attributes that have the same name.

An example of the use of `Attribute` and `AttributeDef`:

```
<ElementDef Type="IMG">
  <AttributeDef Name="SRC" Datatype="uri"/>
    <Description>The URI where the image may be retrieved</Description>
  <Attribute>BORDER</Attribute>
  <Attribute>SiteMap:HUE</Attribute>
</ElementDef>
```

In this example, the properties of the Attribute whose name is `SRC` are declared within the declaration of the `IMG` element. This would make sense if `IMG` is the only element for which the `SRC` attribute applies.

The second attribute, `BORDER`, has a declaration stored separately, referenced by its name. This declaration style is suitable when such an attribute is applicable to multiple elements; it allows maintaining the declaration in one location.

Finally, the declaration for the third attribute, `HUE` uses a qualified name and refers to a declaration found in another DCD, whose namespace is identified by the prefix `SiteMap`. `BORDER` and `HUE` must be defined as global attributes in their respective DCDs.

3.2.2 Contents

Signals whether elements of types not explicitly declared via the `Group` property may appear as children of elements of this type. The value of this property must be a string whose value is `Open` or `Closed`. `Closed` means that this element type is allowed to have children only of types which are declared via the `Group` property. `Open` means that this element type may have children of types not declared via the `Group` property.

Examples of the use of `Content`:

```
<ElementDef Type="DT" Model="Data" Content="Closed"/>
  <Description>The term being defined in a DL list item</Description>
<ElementDef Type="DD" Model="Mixed" Content="Open"/>
  <Description>A term's definition in a DL list item</Description>
```

3.2.3 Datatype

Identifies a specific datatype (in the [XML-Data] sense) which constrains the content of elements of this type. The value of this property must be a string which matches one of an enumerated list of datatypes. See section "4. Datatypes".

The `Datatype` property is only meaningful if the value of the `Model` property is `Data`. That is to say, it is not meaningful to provide a lexical datatype for content which contains substructures.

Examples of the use of `Datatype`:

```
<ElementDef Type="Loan">
  <Description>A Bank Loan</Description>
  <Group RDF:Order="Seq">
    <Element>InterestRate</Element>
    <Element>Amount</Element>
    <Element>Maturity</Element>
  </Group>
</ElementDef>
<ElementDef Type="InterestRate" Datatype="float"/>
<ElementDef Type="Amount" Datatype="int"/>
<ElementDef Type="Maturity" Model="Data" Datatype="dateTime"/>
```

3.2.4 Default and Fixed

Provides default values for the content of elements of this type, and signals whether any value other than the default is allowed. The value of the `Default` property must be a string which provides a default value. The only allowed values of the `Fixed` property are the strings `True` and `False`.

The `Default` value is used in the case that this element type appears as the value of an `Element` property of some other element type, but an element of that type fails to contain a child of this type.

The `Default` property is only meaningful if the value of the `Model` property is `Data`. That is to say, it is not meaningful to provide a default value for content which contains substructures.

When the `Default` property is used to give an element type a default value, the presence of the `Fixed` property with a value of `True` means that the default value is the only one allowed for this element type. If the `Fixed` property is not specified it is assumed to have a value of `False`.

An example of the use of `Default`:

```
<ElementDef Type="AirTicketClass" Model="Data" Datatype="char">
   <Default>Y</Default>
</ElementDef>
```

An example of the use of `Fixed`:

```
<ElementDef Type="Namespace" Model="Data" Fixed="True">
  <Default>http://www.w3.org/TR/REC-xml</Default>
</ElementDef>
```

3.2.5 Description

Provides a, presumably human-readable, description of the semantics and usage of elements of this type. The value of this property must match the production labeled `Content` in the XML specification; that is to say, it may contain markup, and is well-formed.

An example of the use of `Description`:

```
<ElementDef Type="BLINK">
  <Description>A mis-feature which should <em>never</em> be used.</
Description>
</ElementDef>
```

3.2.6 Groups, Occurs and Order

An `ElementDef` whose `Model` property has the value `Elements` must also have a single property named `Group`, containing a specification of the elements and groups which can appear as children of elements of this type. `Groups` in turn may have an `Occurs` property. This can take one of four values.

Required
: occurs exactly once

Optional
: occurs zero or one times

OneOrMore
: occurs one or more times

ZeroOrMore
: occurs zero or more times

The default is `Required`.

A group declares individual elements and other groups which may occur as children of groups of this type. The order of occurrence of the children is declared using the RDF collection ordering facility via the proposed `RDF:Collection` attribute. Legal values are `Seq`, in which case children must occur in the specified order, or `Alt` in which case only one of the specified children may appear. The default is `Seq`. See section "1.3 Future Work".

An example of a simple element declaration:

```
<ElementDef Type="person" Model="Elements" >
  <Group RDF:Order="Seq">
    <Element>FirstName</Element>
    <Group Occurs="Optional"><Element>MI</Element></Group>
    <Element>LastName</Element>
  </Group>
</ElementDef>
```

Here is a more complete example with attribute and element specifications:

```
<ElementDef Type="employee" Model="Elements" Content="Closed">
  <AttributeDef Name="employment" Occurs="Required" Datatype="enumeration">
     <Values>Temporary Permanent Retired</Values>
  </AttributeDef>
  <Group RDF:Order="Seq">
    <Element>FirstName</Element>
    <Group Occurs="Optional"><Element>MI</Element></Group>
      <Element>LastName</Element>
      <Group Occurs="OneOrMore" RDF:Order="Alt">
         <Element>Street</Element><Element>PO-Box</Element>
      </Group>
      <Group RDF:Order="Seq">
        <Element>Telephone</Element>
        <Element>Salary</Element>
      </Group>
    </Group>
  </Group>
</ElementDef>
```

3.2.7 Max, Min, MaxExclusive, MinExclusive

Provide, respectively, upper and lower bounds on the content of elements of this type. `Max` and `Min` allow values upto and including the bound while `MaxExclusive` and `MinExclusive` allow values less than and greater than the bound, respectively, The semantics of upper and lower bounding are highly dependent on the element's `Datatype`; for some datatypes (e.g. `uri`), this property has no meaning.

If an element has no `Datatype`, then `Max, Min, MaxExclusive` and `MinExclusive` values are treated as strings, and tests for upper and lower bounding are performed according to the language specification collation rules defined in Chapter 5.15 of the Unicode standard.[Unicode].

The `Max, Min, MaxExclusive` and `MinExclusive` properties are only meaningful if the value of the `Model` property is `Data`. That is to say, it is not meaningful to provide upper or lower bounds for content which contains substructures.

Examples of the use of `Max` and `Min`:

```
<ElementDef Type="MonthOfYear" Model="Data" Datatype="int"
  Max="12" Min="1" />
```

3.2.8 Model

Indicates which of five broad classes of constraints apply to the content of elements of this type. The value of this property must be a string whose value is one of `Empty`, `Any`, `Data`, `Elements`, or `Mixed`. The meanings are:

Empty
: Elements of this type must have no content.

Any
: Elements of this type may contain text and child elements of any declared type.

Data

Elements of this type contain text, but must not contain any child elements.

Elements

Elements of this type contain only child elements, optionally separated by white space. The types of the child elements that may appear are controlled by the `Group` and `Element` properties.

Mixed

Elements of this type may contain text and embedded child elements. The types of the child elements that may appear are controlled by the `Element` property.

The default is `Data`.

Examples of the use of `Model`:

```
<ElementDef Type='IMG' Model='Empty' />
<ElementDef Type='BODY' Model='Any '/>
<ElementDef Type='DT' Model='Data' />
<ElementDef Type='DL' Model='Elements' />
<ElementDef Type='P' Model='Mixed' />
```

3.2.9 Root

Element definitions can have a `Root` property that indicates whether an element of that type can serve as the root of a conforming document. Allowed values are `True` and `False`. The default is `False`.

If no element definition in a DCD has a `Root="True"` property, then an element of any type that is allowed to appear in such documents may serve as the root element. If multiple element definitions have `Root="True"` then any element of one of those types can appear as the root of a conforming document.

An example of the use of `Root`:

```
<DCD>
  <?DCD syntax="explicit"?>
  <Description>DCD for an email message</Description>
  <ElementDef Type="EMail" Root="True">
   ... declarations ...
  </ElementDef>
  <ElementDef Type="Head">
   ... declarations for Head ...
  </ElementDef>
  <ElementDef Type="Body" Model="Data"/>
</DCD>
```

3.2.10 Type

Gives the type of the element. This property is required to be present for every Element resource in DCD. The value of this property must be a `Name` in the XML sense. Furthermore, it must be an `NCName` as defined in [XML Namespaces]; that is to say, it may not contain a prefix or a colon.

As discussed earlier, the `Type` property for element definitions must be unique within the DCD. But, see Appendix "A. Local Element Definitions".

3.3 Properties Which Apply to Attribute Definitions

The following properties which apply to attribute definitions or attribute types have the same names as, and are identical in effect to, the corresponding properties of element types: `Datatype`, `Default`, `Description`, `Max`, `Min`, `MaxExclusive`, `MinExclusive` and `Fixed`.

3.3.1 Global

Indicates whether the `Name` property of this attribute must be unique in the DCD, and thus can serve as an address for this attribute definition. The possible values are `True` and `False`. The default is `False`.

An example of the use of `Global`:

```
<DCD>
  <?DCD syntax="explicit"?>
  <AttributeDef Name="CLASS" Global="True">
    <!-- facts about the CLASS attribute -->
  </AttributeDef>
</DCD>
```

3.3.2 ID-Role

Signals that the attribute has unique identifier or unique ID pointer semantics. The value of this property must be a string whose value is one of `ID`, `IDREF`, or `IDREFS`. The effect of each of these values is the same as if the attribute had been declared, in an XML DTD, with the attribute type of the same name.

An example of the use of `ID-Role`:

```
<ElementDef Type="A">
  <AttributeDef>
    <Name>NAME</Name>
    <ID-Role>ID</ID-Role>
  </AttributeDef>
</ElementDef>
```

3.3.3 Name

Gives the name of the attribute. This property is required to be present for every Attribute resource in DCD. The value of this property must be a `Name` in the XML sense. Furthermore, it must be an `NCName` as defined in [XML Namespaces]; that is to say, it may not contain a prefix or a colon.

As discussed earlier, the `Name` property for attribute definitions that have `Global="True"` must be unique within the DCD.

3.3.4 Occurs

Indicates whether the presence of the Attribute is required. This can take one of two values.

Required
occurs exactly once
Optional
occurs zero or one times

The default is Optional.

3.4 Properties Which Apply to Internal Entity Definitions

3.4.1 Name

Gives the name by which the entity may be invoked. This property is required to be present for every InternalEntity definition resource in DCD. The value of this property must be a `Name` in the XML sense. Furthermore, it must be an `NCName` as defined in [XML Namespaces]; that is to say, it may not contain a prefix or a colon.

3.4.2 Value

Provides the replacement text for the internal entity. The value of this property must match the production labeled `Content` in the XML specification; that is to say, it may contain markup, and is well-formed.

An example of the use of `Value`:

```
<InternalEntityDef>
  <Name>Warning</Name>
  <Value>Entity text <em>can</em> contain markup; references (e.g. &copy;)
      will in general be expanded unless protected, e.g. &amp;copy;</Value>
</InternalEntityDef>
```

3.5 Properties Which Apply to External Entity Definitions

3.5.1 Name

Gives the name by which the entity may be invoked. This property is required to be present for every ExternalEntity definition resource in DCD. The value of this property must be a `Name` in the XML sense. Furthermore, it must be an `NCName` as defined in [XML Namespaces]; that is to say, it may not contain a prefix or a colon.

3.5.2 PublicID

Provides a public identifier for the entity. This is a string whose syntax (see `PublicID`) and semantics are exactly as described in the XML specification.

3.5.3 SystemID

Provides a system identifier for the entity. This is a string whose syntax and semantics are exactly as described in the XML specification.

The `SystemID` property must be provided for every ExternalEntity resource in DCD.

4. Datatypes

4.1 Datatype Specifications

A number of datatypes are specified in this section. These are modeled after the datatypes supported by [SQL] and modern programming languages. Attributes and element types whose `Model` property has the value `Data` can constrain their values/contents to be instances of a particular datatype. XML 1.0 defines about 10 datatypes, which may only be used to constrain attribute values, and essentially one datatype, PCDATA, that can be used for element content. Here we propose a much richer set of datatypes, applicable equally to attribute and element content.

The specifications in this section serve a number of purposes:

- They specify maximum values on certain datatypes. For example, numbers can be up to 31 characters long.
- They specify syntax to constrain the value of a particular element/attribute within these maximum values.
- They specify acceptable formats for the specification of such datatypes.

Datatypes are referenced from the datatype namespace. In order to use this namespace in a schema, it must be declared. Some datataypes require that additional properties be specified. For example, `length` and `precision` for `decimal`, `length` for `char` and legal `values` for `enumeration`. These should be specified as additional properties of the element or attribute being defined. See the final example in "3.2.6 Groups, Occurs and Order".

The DCD primitive datatypes are tabulated below.

Name	Examples	Parse type
id	X	XML ID
idref	X	XML IDREF
idrefs	X Y Z	XML IDREFS
entity	Foo	XML ENTITY
entities	Foo Bar	XML ENTITIES
nmtoken	Name	XML NMTOKEN
nmtokens	Name1 Name2	XML NMTOKENS
enumeration Legal values must be specified.	Red Blue Green	XML ENUMERATION
notation	GIF	XML NOTATION
string	Give me liberty or give me death!	pcdata
number	15, 3.14, -123.456E+10	A number, with up to 31 digits. May optionally have a leading sign, fractional digits, and exponent. Punctuation as in US English. Leading and trailing blanks are removed before converting a number specified as as string. Similarly, leading and trailing zeroes are removed.
int	1, 58502, -13	A number, with optional sign, no fractions, no exponent.

fixed or decimal Precision and scale must be specified.	12.0044	Precision is the total number of digits. It may range from 1 to 31. Scale is the number of digits to the right of the decimal point and must be less than or equal to the precision.
boolean	0, 1 (1=="true")	"1" or "0"
dateTime	2088-04-07T18:39:09	A date in a subset of ISO 8601 format, with optional time and no optional zone. Fractional seconds may be as precise as nanoseconds.
dateTime.tz	2088-04-07T18:39:09-08:00	A date in a subset ISO 8601 format, with optional time and optional zone. Fractional seconds may be as precise as nanoseconds.
date	2094-11-05	A date in a subset ISO 8601 format. (no time)
time	08:15:27	A time in a subset ISO 8601 format, with no date and no time zone. Fractional seconds may be as precise as nanoseconds.
time.tz	08:1527-05:00	A time in a subset ISO 8601 format, with no date but optional time zone. Fractional seconds may be as precise as nanoseconds.
interval	2088-04-07T18:39:09	A time interval which may have year, month, day, hour, minute and second fields. Fractional seconds may be as precise as nanoseconds.
i1, byte 1-byte integer	1, 127, -128	A number, with optional sign, no fractions, no exponent.
i2 2-byte integer	1, 703, -32768	"
i4, int 4-byte integer	1, 703, -32768, 148343, -1000000000	"
i8 8-byte integer	1, 703, -32768, 1483433434334, -1000000000000000	"
ui1 unsigned 1-byte integer	1, 255	A number, unsigned, no fractions, no exponent.
ui2 unsigned 2-byte integer	1, 255, 65535	"
ui4 unsigned 4-byte integer	1, 703, 3000000000	"
ui8 unsigned 4-byte integer	1483433434334	"

r4	.31415E+1	Real number ranging from -3.402E+38 to -1.175E-37 or from 1.175E-37 to 3.402E+38
r8	.314159265358979E+1	Real number ranging from -1.79769E+308 to -2.225E-307 or from 2.225E-307 to 1.79769E+308
fixed.14.4	1.95	A number with 14 digits to the left of the decimal point and 4 digits to the right of the decimal point. Convenient for representing monetary values.
uuid	333C4-460F-11D0-BC04-0080CA83	Hexadecimal digits representing octets. Optional embedded hyphens are allowed but ignored during conversion.
uri	urn:schemas-microsoft-com:Office9 http://www.ics.uci.edu/pub/ietf/uri/	Universal Resource Identifier
bin.hex Length may be specified. Default is unlimited.		Hexadecimal digits representing octets
bin.base64 Length may be specified. Default is unlimited.		MIME style Base64 encoded binary blob.
char Length may be specified. Default is 1.	char	Character string, n characters long
picture Picture must be specified.	999-99-9999	Constraint for validating strings. See note below.

4.2 Datatypes in instances

The datatypes defined in "4. Datatypes" can also be used in instance datatype specifications as described in XML-Data [XML-Data]. For example:

```
<conversionRate DCD:dt="float">1.4172</conversionRate>
```

This provides the benefit of datatype support to well-formed documents that may not have an associated DTD or DCD. It is expected that XML parsers would provide assistance in encoding and decoding these datatypes.

4.3 Picture Constraints

"Pictures", similar to those in [COBOL] picture clauses, can be used to constrain the format of strings and in some cases control their conversion to numbers. A picture is an alphanumeric string consisting of character symbols. Each symbol, which is usually one character but may be two characters, is a placeholder that stands for a set of characters. For example, the picture "A" stands for a single alphabetic character.

The following is a list of picture symbols and their meanings.

A
: A single alphabetic character.

B
: A single blank character.

E
: The character E, used to indicate floating point numbers.

S
: The leftmost character of a picture indicating a signed number. The characters "+" or "-" *may* appear in the S position.

V
: An implied decimal sign. The input 1234 validated by a picture 99V99 is converted into 12.34.

X
: Any character.

Z
: The leftmost leading numeric character that can be replaced by a space character when the content of that content position is a zero.

9
: Any numeric character.

1
: Any boolean character (0 or 1).

0,/,-,., and ,
: represent themselves.

cs
: The currency symbol.

Here are some examples of picture constraints

```
$123,45.90 satisfies picture $999,99.99
$123,45.90 satisfies picture XXXX,XX.XX
123-45-5678 satisfies picture 999-99-9999 (Social Security Number)
24E80 satisfies picture 99E99 (floating point)
23.45 satisfies picture 99.99
2345 satisfies picture 99V99 (translates to 23.45)
```

Material in the appendices represents issues that are still under discussion. This material should be considered for inclusion in a later version of DCD.

Appendices

A. Local Element Definitions

The specifications in this document only allow elements to be defined as properties of the DCD. A useful future direction may be to allow element definitions within the context of another element definition. Element definitions may be local or global. Global element definitions must have a `Type` property that is unique in the DCD and can be referred to by name in other definitions. Local element definitions can be used within the containing definition and can be referred to in other definitions by a resource identifier as described for attribute definitions in "2.3 Referring to Elements and Attributes".

For example, in the following, `FirstName`, `MI` and `LastName` are defined elsewhere in the DTD but `Address` comes from a namespace declared with the `common` prefix. The `Telephone` element is defined locally within the `person` definition.

```
<ElementDef Type="person" Model="Elements" >
  <Group RDF:Order="Seq">
    <Element>FirstName</Element>
    <Group Occurs="Optional"><Element>MI</Element></Group>
    <Element>LastName</Element>
    <Element>common:Address</Element>
    <ElementDef Type="Telephone" Datatype="string"/>
  </Group>
</ElementDef>
```

B. Inheritance and Subclassing

An element type may be declared to re-use the content model declarations of other element types through the use of the `extends` property. This property effectively replaces itself with the entire content model of the element type it names. For example:

```
<ElementDef Type="polygon" Model="Elements">
  <AttributeDef Name="n" Occurs="Required"/>
  <AttributeDef Name="Regularity"/>
  <Element>diagonals</Element>
</ElementDef>

<ElementDef Type="regularPolygon" Model="Element">
  <AttributeDef Name="regularity" Occurs="Required">
     <Default>regular</Default>
  </AttributeDef>
  <Element>side</Element>
  <Extends Type="polygon"/>
</ElementDef>
```

A legal instance of regularPolygon (in this case an empty equilateral triangle 3mm on a side) might be:

```
<regularPolygon n="3">
  <side><dimension unit='mm'>3</dimension></side>
  <diagonals/>
</regularPolygon>
```

Using `extends` also allows instances of the extending element type to occur anywhere the extended type is allowed. In the above example this means that any content model that allows polygon will also now allow regularPolygon. Furthermore, attributes declared on the extended element type may also occur on the extending element type, so in the example `n` can, in fact must, now appear on regularPolygon. For example, if in addition to the above example we have:

```
<ElementDef Type="picture">
  <Group Occurs="OneOrMore">
    <Element>polygon</Element>
  </Group>
</ElementDef>
```

then the following is a valid schema:

```
<picture>
  <polygon n="3" regularity="irregular">...</polygon>
  <regularPolygon n="3">...</regularPolygon>
</picture>
```

Note that in the above examples, `Element` declarations occur directly within an `ElementDef` without an enclosing `Group`. We allow this to facilitate inheritance. The `Element` declaration opens a default `Group`. In fact, `Element` extends `Group` and inherits its properties.

We restrict the use of `extends` to cases where the merger of the two content models involved is straightforward.

1. Either the extended element type must have `Content="Open"` or the extending element type must have no content at all, either explicit or inherited.
2. If the extending element type has explicit content, the values of the `order` attribute must be consistent. The following table shows all the allowed values (if the extended element type has order with value `Alt`, no extension is possible):

Extended	Extending
Seq	Seq
Bag	Bag; Seq
Alt	Alt

3. The values of the content attribute must be consistent, as follows:

Extended	Extending
Empty	Empty
Data	Data; Empty
Elements	Elements
Any; Mixed	Any; Mixed; Data; Elements

4. Allowed attributes and datatype constraints (see "4. Datatypes") are cumulative, that is, all apply. Attributes of the same name are merged: the only difference allowed is that an attribute in the extending declaration may provide and/or require a default where the extended declaration does not. Multiple datatype constraints, whether for content or for an attribute, must be intelligibly combinable, (see "4. Datatypes").

Consistent with the above remark about the extending element type being allowed anywhere the extended one is, the guiding principle is that anything allowed by the extending declaration would also be allowed by the extended one if the tag was changed. That is, the extending type is polymorphic to the extended type. Thus, if we rename regularPolygon to polygon in the first example above, we get a schema-valid polygon:

```
<polygon n="3">
  <side><dimension unit='mm'>3</dimension></side>
  <diagonals/>
</polygon>
```

It's legal as a polygon, because it has everything a polygon requires (`n` attribute, `diagonals` sub-element), and the `side` sub-element is permissible because polygon has, by default, open Content.

Note that a single ElementDef can contain multiple extends. This does not cause ambiguity -- effectively, the extended content model is dropped in as a group in the relevant place in the extending model.

C. Null Values

For several situations, especially in mapping data from a database into XML, we need to handle the case where the value is not specified. This is different from a numeric value being zero or a string being empty.

If the element or attribute is not `Required` then it can just be omitted. If it is `Required` or if it has a default value then it is desirable to be able to indicate that its value in the database was undefined. This can be done by defining a special attribute to signal this condition. If an element is involved then the special attribute is an attribute of the element. If an attribute is involved it is another attribute of the parent element. In either case, the special attribute takes one of two values "True" or "False".

Consider the case of a required `Salary` element. A missing `Salary` element would be appear as:

```
<Employee>
    ...
  <Salary DCD:null="True"/>
</Employee>
```

If `Salary` was a required attribute on, say, an `Employee` element then we would need to define another attribute on `Employee` called, say, `Salary_null`.

If the element or attribute had a default value the value would appear along with the `null` attribute with a `True` value.

Similarly, special attributes can be defined to indicate errors in data conversion

D. Unique Values

In current XML, the `ID` attribute type is unique within a document. Unique attribute and element types are very important and should be extended to any named attribute and element type with the ability to specify the scope of the uniqueness. For elements, uniqueness specification applies only if the model type is `Data` i.e. it does not apply to elements that have structure. Particular implementations can use unique element and attribute types to define keys to speed up searches.

Essentially, when defining an attribute type we can specify that it's value is unique within a particular element type.

```
<AttributeDef UniqueIn="Company" Global="True"
  <Name>SerialNumber</Name>
  <Datatype>int</Datatype
</AttributeDef>
```

`Company` is the name of an element type defined within the DTD. This specifies that the `SerialNumber` attribute is unique within `Company` elements in documents conformant with this DTD. The default value of the `UniqueIn` Attribute is "null" which signifies the entire document. Thus, the default behavior is the current XML behavior.

E. References

COBOL
: COBOL Standard. See http://www.dkuug.dk/jtc1/sc22/wg4/

SQL
: SQL Standard. See http://www.jcc.com/sql_stnd.html.

RDF
: RDF Model and Syntax. See http://www.w3.org/TR/WD-rdf-syntax.

Unicode
: Unicode Standard. See "The Unicode Standard, Version 2.0", Reading Mass., Addison-Wesley Developers Press, 1996

XML-Data
: XML-Data. See http://www.w3.org/TR/1998/NOTE-XML-data-0105/.

XML Namespaces
: Namespaces in XML. See http://www.w3.org/TR/WD-xml-names.

F. Acknowledgements

This work is totally dependent on the whole lineage of metadata thinking in the World Wide Web Consortium. This specification has benefited greatly as a result of input from David Fallside and David Singer, both of IBM, Andrew Layman and Jean Paoli both of Microsoft, and from Lauren Wood of SoftQuad. We also wish to thank Henry Thompson of the University of Edinburgh and all the authors of the XML-Data specification [XML-Data].

RDF Primer

W3C Recommendation 10 February 2004

This version:
http://www.w3.org/TR/2004/REC-rdf-primer-20040210/
Latest version:
http://www.w3.org/TR/rdf-primer/
Previous version:
http://www.w3.org/TR/2003/PR-rdf-primer-20031215/
Editors:
Frank Manola, fmanola@acm.org
Eric Miller, W3C, em@w3.org
Series Editor:
Brian McBride, Hewlett-Packard Laboratories, bwm@hplb.hpl.hp.com

Please refer to the **errata** for this document, which may include some normative corrections.

See also translations.

Abstract

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. This Primer is designed to provide the reader with the basic knowledge required to effectively use RDF. It introduces the basic concepts of RDF and describes its XML syntax. It describes how to define RDF vocabularies using the RDF Vocabulary Description Language, and gives an overview of some deployed RDF applications. It also describes the content and purpose of other RDF specification documents.

Status of this Document

This document has been reviewed by W3C Members and other interested parties, and it has been endorsed by the Director as a W3C Recommendation. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This is one document in a set of six (Primer, Concepts, Syntax, Semantics, Vocabulary, and Test Cases) intended to jointly replace the original Resource Description Framework specifications, RDF Model and Syntax (1999 Recommendation) and RDF Schema (2000 Candidate

Recommendation). It has been developed by the RDF Core Working Group as part of the W3C Semantic Web Activity (Activity Statement, Group Charter) for publication on 10 February 2004.

Changes to this document since the Proposed Recommendation Working Draft are detailed in the change log.

The public is invited to send comments to www-rdf-comments@w3.org (archive) and to participate in general discussion of related technology on www-rdf-interest@w3.org (archive).

A list of implementations is available.

The W3C maintains a list of any patent disclosures related to this work.

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Table of Contents

Appendices

---

1. Introduction

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be *identified* on the Web, even when they cannot be directly *retrieved* on the Web. Examples include information about items available from on-line shopping facilities (e.g., information about specifications, prices, and availability), or the description of a Web user's preferences for information delivery.

RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created.

RDF is based on the idea of identifying things using Web identifiers (called *Uniform Resource Identifiers*, or *URIs*), and describing resources in terms of simple properties and property values. This enables RDF to represent simple statements about resources as a *graph* of nodes and arcs representing the resources, and their properties and values. To make this discussion somewhat more concrete as soon as possible, the group of statements "there is a Person identified by `http://www.w3.org/People/EM/contact#me`, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr." could be represented as the RDF graph in Figure 1:

Figure 1: An RDF Graph Describing Eric Miller

Figure 1 illustrates that RDF uses URIs to identify:

- individuals, e.g., Eric Miller, identified by `http://www.w3.org/People/EM/contact#me`
- kinds of things, e.g., Person, identified by `http://www.w3.org/2000/10/swap/pim/contact#Person`
- properties of those things, e.g., mailbox, identified by `http://www.w3.org/2000/10/swap/pim/contact#mailbox`
- values of those properties, e.g. `mailto:em@w3.org` as the value of the mailbox property (RDF also uses character strings such as "Eric Miller", and values from other datatypes such as integers and dates, as the values of properties)

RDF also provides an XML-based syntax (called *RDF/XML*) for recording and exchanging these graphs. Example 1 is a small chunk of RDF in RDF/XML corresponding to the graph in Figure 1:

Example 1: RDF/XML Describing Eric Miller

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:contact="http://www.w3.org/2000/10/swap/pim/
contact#">

  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>

</rdf:RDF>
```

Note that this RDF/XML also contains URIs, as well as properties like `mailbox` and `fullName` (in an abbreviated form), and their respective values `em@w3.org`, and `Eric Miller`.

Like HTML, this RDF/XML is machine processable and, using URIs, can link pieces of information across the Web. However, unlike conventional hypertext, RDF URIs can refer to any identifiable thing, including things that may not be directly retrievable on the Web (such as the person Eric Miller). The result is that in addition to describing such things as Web pages, RDF can also describe cars, businesses, people, news events, etc. In addition, RDF properties themselves have URIs, to precisely identify the relationships that exist between the linked items.

The following documents contribute to the specification of RDF:

- RDF Concepts and Abstract Syntax [RDF-CONCEPTS]
- RDF/XML Syntax Specification [RDF-SYNTAX]
- RDF Vocabulary Description Language 1.0: RDF Schema [RDF-VOCABULARY]
- RDF Semantics [RDF-SEMANTICS]
- RDF Test Cases [RDF-TESTS]
- RDF Primer (this document)

This Primer is intended to provide an introduction to RDF and describe some existing RDF applications, to help information system designers and application developers understand the features of RDF and how to use them. In particular, the Primer is intended to answer such questions as:

- What does RDF look like?
- What information can RDF represent?
- How is RDF information created, accessed, and processed?
- How can existing information be combined with RDF?

The Primer is a *non-normative* document, which means that it does not provide a definitive specification of RDF. The examples and other explanatory material in the Primer are provided to help readers understand RDF, but they may not always provide definitive or fully-complete answers. In such cases, the relevant normative parts of the RDF specification should be consulted. To help in doing this, the Primer describes the roles these other documents play in the complete specification of RDF, and provides links pointing to the relevant parts of the normative specifications, at appropriate places in the discussion.

It should also be noted that these RDF documents update and clarify previously-published RDF specifications, the Resource Description Framework (RDF) Model and Syntax Specification [RDF-MS] and the Resource Description Framework (RDF) Schema Specification 1.0 [RDF-S]. As a result, there have been some changes in terminology, syntax, and concepts. This Primer reflects the newer set of RDF specifications given in the bulleted list of RDF documents cited above. Hence, readers familiar with the older specifications, and with earlier tutorial and introductory articles based on them, should be aware that there may be differences between the current specifications and those previous documents. The RDF Issue Tracking document [RDFISSUE] can be consulted for a list of issues raised concerning the previous RDF specifications, and their resolution in the current specifications.

2. Making Statements About Resources

RDF is intended to provide a simple way to make statements about Web resources, e.g., Web pages. This section describes the basic ideas behind the way RDF provides these capabilities (the normative specification describing these concepts is RDF Concepts and Abstract Syntax

[RDF-CONCEPTS]).

2.1 Basic Concepts

Imagine trying to state that someone named John Smith created a particular Web page. A straightforward way to state this in a natural language such as English would be in the form of a simple statement such as:

```
http://www.example.org/index.html has a creator whose value
is John Smith
```

Parts of this statement are emphasized to illustrate that, in order to describe the properties of something, there need to be ways to name, or identify, a number of things:

- the thing the statement describes (the Web page, in this case)
- a specific property (creator, in this case) of the thing the statement describes
- the thing the statement says is the value of this property (who the creator is), for the thing the statement describes

In this statement, the Web page's URL (Uniform Resource Locator) is used to identify it. In addition, the word "creator" is used to identify the property, and the two words "John Smith" to identify the thing (a person) that is the value of this property.

Other properties of this Web page could be described by writing additional English statements of the same general form, using the URL to identify the page, and words (or other expressions) to identify the properties and their values. For example, the date the page was created, and the language in which the page is written, could be described using the additional statements:

```
http://www.example.org/index.html has a creation-date whose
value is August 16, 1999
http://www.example.org/index.html has a language whose
value is English
```

RDF is based on the idea that the things being described have properties which have values, and that resources can be described by making statements, similar to those above, that specify those properties and values. RDF uses a particular terminology for talking about the various parts of statements. Specifically, the part that identifies the thing the statement is about (the Web page in this example) is called the subject. The part that identifies the property or characteristic of the subject that the statement specifies (creator, creation-date, or language in these examples) is called the predicate, and the part that identifies the value of that property is called the object. So, taking the English statement

```
http://www.example.org/index.html has a creator whose value
is John Smith
```

the RDF terms for the various parts of the statement are:

- the *subject* is the URL `http://www.example.org/index.html`
- the *predicate* is the word "creator"
- the *object* is the phrase "John Smith"

However, while English is good for communicating between (English-speaking) humans, RDF is about making *machine-processable* statements. To make these kinds of statements suitable for processing by machines, two things are needed:

- a system of machine-processable identifiers for identifying a subject, predicate, or object in a statement without any possibility of confusion with a similar-looking identifier that might be used by someone else on the Web.
- a machine-processable language for representing these statements and exchanging them between machines.

Fortunately, the existing Web architecture provides both these necessary facilities.

As illustrated earlier, the Web already provides one form of identifier, the *Uniform Resource Locator* (URL). A URL was used in the original example to identify the Web page that John Smith created. A URL is a character string that identifies a Web resource by representing its primary access mechanism (essentially, its network "location"). However, it is also important to be able to record information about many things that, unlike Web pages, do not have network locations or URLs.

The Web provides a more general form of identifier for these purposes, called the Uniform Resource Identifier (URI). URLs are a particular kind of URI. All URIs share the property that different persons or organizations can independently create them, and use them to identify things. However, URIs are not limited to identifying things that have network locations, or use other computer access mechanisms. In fact, a URI can be created to refer to anything that needs to be referred to in a statement, including

- network-accessible things, such as an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), or a group of other resources.
- things that are not network-accessible, such as human beings, corporations, and bound books in a library.
- abstract concepts that do not physically exist, such as the concept of a "creator".

Because of this generality, RDF uses URIs as the basis of its mechanism for identifying the subjects, predicates, and objects in statements. To be more precise, RDF uses URI references [URIS]. A URI reference (or *URIref*) is a URI, together with an optional *fragment identifier* at the end. For example, the URI reference `http://www.example.org/index.html#section2` consists of the URI `http://www.example.org/index.html` and (separated by the "#" character) the fragment identifier `Section2`. RDF URIrefs can contain Unicode [UNICODE] characters (see [RDF-CONCEPTS]), allowing many languages to be reflected in URIrefs. RDF defines a *resource* as anything that is identifiable by a URI reference, so using URIrefs allows RDF to describe practically anything, and to state relationships between such things as well. URIrefs and fragment identifiers are discussed further in Appendix A, and in [RDF-CONCEPTS].

To represent RDF statements in a machine-processable way, RDF uses the Extensible Markup Language [XML]. XML was designed to allow anyone to design their own document format and then write a document in that format. RDF defines a specific XML markup language, referred to as *RDF/XML*, for use in representing RDF information, and for exchanging it between machines. An example of RDF/XML was given in Section 1. That example (Example 1) used tags such as `<contact:fullName>` and `<contact:personalTitle>` to delimit the text content `Eric`

`Miller` and `Dr.`, respectively. Such tags allow programs written with an understanding of what the tags mean to properly interpret that content. Both XML content and (with certain exceptions) tags can contain Unicode [UNICODE] characters, allowing information from many languages to be directly represented. Appendix B provides further background on XML in general. The specific RDF/XML syntax used for RDF is described in more detail in Section 3, and is normatively defined in [RDF-SYNTAX]

2.2 The RDF Model

Section 2.1 has introduced RDF's basic statement concepts, the idea of using URI references to identify the things referred to in RDF statements, and RDF/XML as a machine-processable way to represent RDF statements. With that background, this section describes how RDF uses URIs to make statements about resources. The introduction said that RDF was based on the idea of expressing simple statements about resources, where each statement consists of a subject, a predicate, and an object. In RDF, the English statement:

```
http://www.example.org/index.html has a creator whose value
is John Smith
```

could be represented by an RDF statement having:

- a subject `http://www.example.org/index.html`
- a predicate `http://purl.org/dc/elements/1.1/creator`
- and an object `http://www.example.org/staffid/85740`

Note how URIrefs are used to identify not only the subject of the original statement, but also the predicate and object, instead of using the words "creator" and "John Smith", respectively (some of the effects of using URIrefs in this way will be discussed later in this section).

RDF models statements as nodes and arcs in a graph. RDF's graph model is defined in [RDF-CONCEPTS]. In this notation, a statement is represented by:

- a node for the subject
- a node for the object
- an arc for the predicate, directed from the subject node to the object node.

So the RDF statement above would be represented by the graph shown in Figure 2:

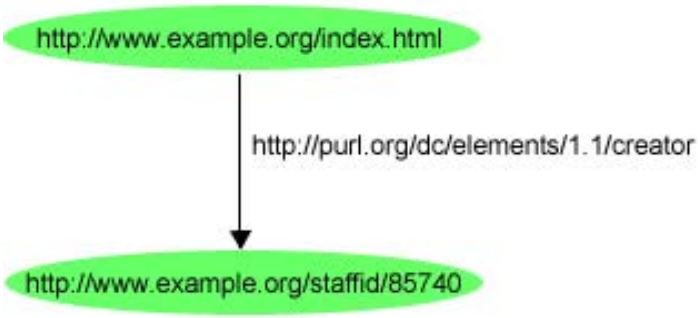


Figure 2: A Simple RDF Statement

Groups of statements are represented by corresponding groups of nodes and arcs. So, to reflect the additional English statements

```
http://www.example.org/index.html has a creation-date whose
value is August 16, 1999
http://www.example.org/index.html has a language whose
value is English
```

in the RDF graph, the graph shown in Figure 3 could be used (using suitable URIrefs to name the properties "creation-date" and "language"):

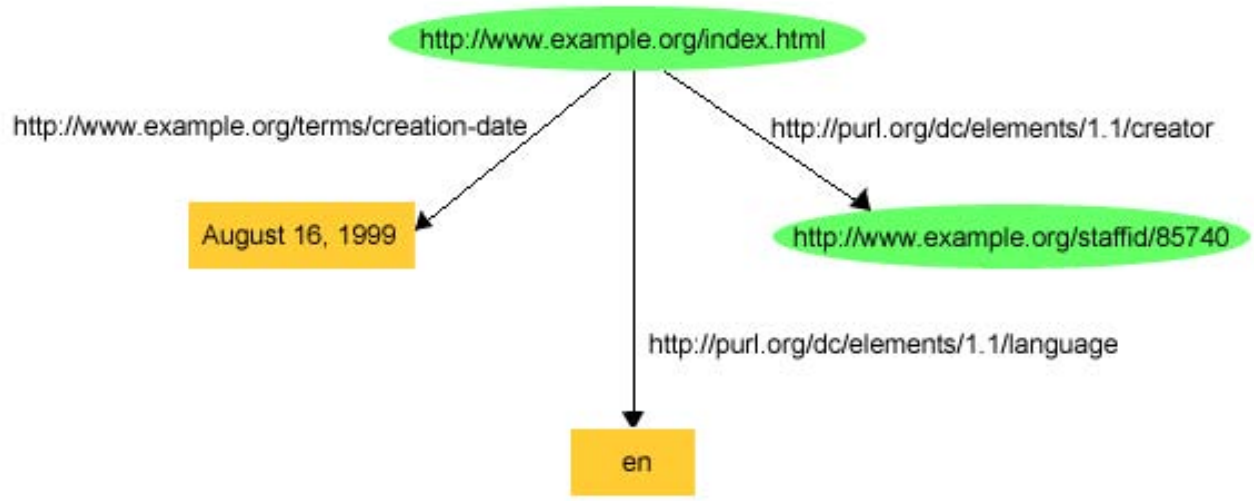


Figure 3: Several Statements About the Same Resource

Figure 3 illustrates that objects in RDF statements may be either URIrefs, or constant values (called literals) represented by character strings, in order to represent certain kinds of property values. (In the case of the predicate `http://purl.org/dc/elements/1.1/language` the literal is an international standard two-letter code for English.) Literals may not be used as subjects or predicates in RDF statements. In drawing RDF graphs, nodes that are URIrefs are shown as ellipses, while nodes that are literals are shown as boxes. (The simple character string literals used in these examples are called plain literals, to distinguish them from the typed literals to be introduced in Section 2.4. The various kinds of literals that can be used in RDF statements are defined in [RDF-CONCEPTS]. Both plain and typed literals can contain Unicode [UNICODE] characters, allowing information from many languages to be directly represented.)

Sometimes it is not convenient to draw graphs when discussing them, so an alternative way of writing down the statements, called triples, is also used. In the triples notation, each statement in the graph is written as a simple triple of subject, predicate, and object, in that order. For example, the three statements shown in Figure 3 would be written in the triples notation as:

```
<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/
creator> <http://www.example.org/staffid/85740> .

<http://www.example.org/index.html> <http://www.example.org/terms/
creation-date> "August 16, 1999" .

<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/
language> "en" .
```

Each triple corresponds to a single arc in the graph, complete with the arc's beginning and ending

nodes (the subject and object of the statement). Unlike the drawn graph (but like the original statements), the triples notation requires that a node be separately identified for each statement it appears in. So, for example, `http://www.example.org/index.html` appears three times (once in each triple) in the triples representation of the graph, but only once in the drawn graph. However, the triples represent exactly the same information as the drawn graph, and this is a key point: what is fundamental to RDF is the *graph model* of the statements. The notation used to represent or depict the graph is secondary.

The full triples notation requires that URI references be written out completely, in angle brackets, which, as the example above illustrates, can result in very long lines on a page. For convenience, the Primer uses a shorthand way of writing triples (the same shorthand is also used in other RDF specifications). This shorthand substitutes an XML *qualified name* (or *QName*) without angle brackets as an abbreviation for a full URI reference (QNames are discussed further in Appendix B). A QName contains a *prefix* that has been assigned to a namespace URI, followed by a colon, and then a *local name*. The full URIref is formed from the QName by appending the local name to the namespace URI assigned to the prefix. So, for example, if the QName prefix `foo` is assigned to the namespace URI `http://example.org/somewhere/`, then the QName `foo:bar` is shorthand for the URIref `http://example.org/somewhere/bar`. Primer examples will also use several "well-known" QName prefixes (without explicitly specifying them each time), defined as follows:

prefix `rdf:`, namespace URI: `http://www.w3.org/1999/02/22-rdf-syntax-ns#`
prefix `rdfs:`, namespace URI: `http://www.w3.org/2000/01/rdf-schema#`
prefix `dc:`, namespace URI: `http://purl.org/dc/elements/1.1/`
prefix `owl:`, namespace URI: `http://www.w3.org/2002/07/owl#`
prefix `ex:`, namespace URI: `http://www.example.org/` (or `http://www.example.com/`)
prefix `xsd:`, namespace URI: `http://www.w3.org/2001/XMLSchema#`

Obvious variations on the "example" prefix `ex:` will also be used as needed in the examples, for instance,

prefix `exterms:`, namespace URI: `http://www.example.org/terms/` (for terms used by an example organization),
prefix `exstaff:`, namespace URI: `http://www.example.org/staffid/` (for the example organization's staff identifiers),
prefix `ex2:`, namespace URI: `http://www.domain2.example.org/` (for a second example organization), and so on.

Using this new shorthand, the previous set of triples can be written as:

```
ex:index.html  dc:creator             exstaff:85740 .

ex:index.html  exterms:creation-date  "August 16, 1999" .

ex:index.html  dc:language            "en" .
```

Since RDF uses URIrefs instead of words to name things in statements, RDF refers to a set of URIrefs (particularly a set intended for a specific purpose) as a *vocabulary*. Often, the URIrefs in such vocabularies are organized so that they can be represented as a set of QNames using a common prefix. That is, a common namespace URIref will be chosen for all terms in a vocabulary, typically a URIref under the control of whoever is defining the vocabulary. URIrefs that are contained in the vocabulary are formed by appending individual local names to the end of the common URIref. This forms a set of URIrefs with a common prefix. For instance, as illustrated by the previous examples, an organization such as example.org might define a vocabulary consisting of URIrefs starting with the prefix `http://www.example.org/terms/`

for terms it uses in its business, such as "creation-date" or "product", and another vocabulary of URIrefs starting with `http://www.example.org/staffid/` to identify its employees. RDF uses this same approach to define its own vocabulary of terms with special meanings in RDF. The URIrefs in this RDF vocabulary all begin with `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, conventionally associated with the QName prefix `rdf:`. The RDF Vocabulary Description Language (described in Section 5) defines an additional set of terms having URIrefs that begin with `http://www.w3.org/2000/01/rdf-schema#`, conventionally associated with the QName prefix `rdfs:`. (Where a specific QName prefix is commonly used in connection with a given set of terms in this way, the QName prefix itself is sometimes used as the name of the vocabulary. For example, someone might refer to "the `rdfs:` vocabulary".)

Using common URI prefixes provides a convenient way to organize the URIrefs for a related set of terms. However, this is just a convention. The RDF model only recognizes full URIrefs; it does not "look inside" URIrefs or use any knowledge about their structure. In particular, RDF does not assume there is any relationship between URIrefs just because they have a common leading prefix (see Appendix A for further discussion). Moreover, there is nothing that says that URIrefs with different leading prefixes cannot be considered part of the same vocabulary. A particular organization, process, tool, etc. can define a vocabulary that is significant for it, using URIrefs from any number of other vocabularies as part of its vocabulary.

In addition, sometimes an organization will use a vocabulary's namespace URIref as the URL of a Web resource that provides further information about that vocabulary. For example, as noted earlier, the QName prefix `dc:` will be used in Primer examples, associated with the namespace URIref `http://purl.org/dc/elements/1.1/`. In fact, this refers to the Dublin Core vocabulary described in Section 6.1. Accessing this namespace URIref in a Web browser will retrieve additional information about the Dublin Core vocabulary (specifically, an RDF schema). However, this is also just a convention. RDF does not assume that a namespace URI identifies a retrievable Web resource (see Appendix B for further discussion).

In the rest of the Primer, the term *vocabulary* will be used when referring to a set of URIrefs defined for some specific purpose, such as the set of URIrefs defined by RDF for its own use, or the set of URIrefs defined by example.org to identify its employees. The term *namespace* will be used only when referring specifically to the syntactic concept of an XML namespace (or in describing the URI assigned to a prefix in a QName).

URIrefs from different vocabularies can be freely mixed in RDF graphs. For example, the graph in Figure 3 uses URIrefs from the `exterms:`, `exstaff:`, and `dc:` vocabularies. Also, RDF imposes no restrictions on how many statements using a given URIref as predicate can appear in a graph to describe the same resource. For example, if the resource `ex:index.html` had been created by the cooperative efforts of several staff members in addition to John Smith, example.org might have written the statements:

```
ex:index.html  dc:creator  exstaff:85740 .

ex:index.html  dc:creator  exstaff:27354 .

ex:index.html  dc:creator  exstaff:00816 .
```

These examples of RDF statements begin to illustrate some of the advantages of using URIrefs as RDF's basic way of identifying things. For instance, in the first statement, instead of identifying the creator of the Web page by the character string "John Smith", he has been assigned a URIref, in this case (using a URIref based on his employee number) `http://www.example.org/staffid/85740` . An advantage of using a URIref in this case is that the identification of the statement's subject can be more precise. That is, the creator of the page is not the character

string "John Smith", or any one of the thousands of people named John Smith, but the particular John Smith associated with that URIref (whoever created the URIref defines the association). Moreover, since there is a URIref to refer to John Smith, he is a full-fledged resource, and additional information can be recorded about him, simply by adding additional RDF statements with John's URIref as the subject. For example, Figure 4 shows some additional statements giving John's name and age.

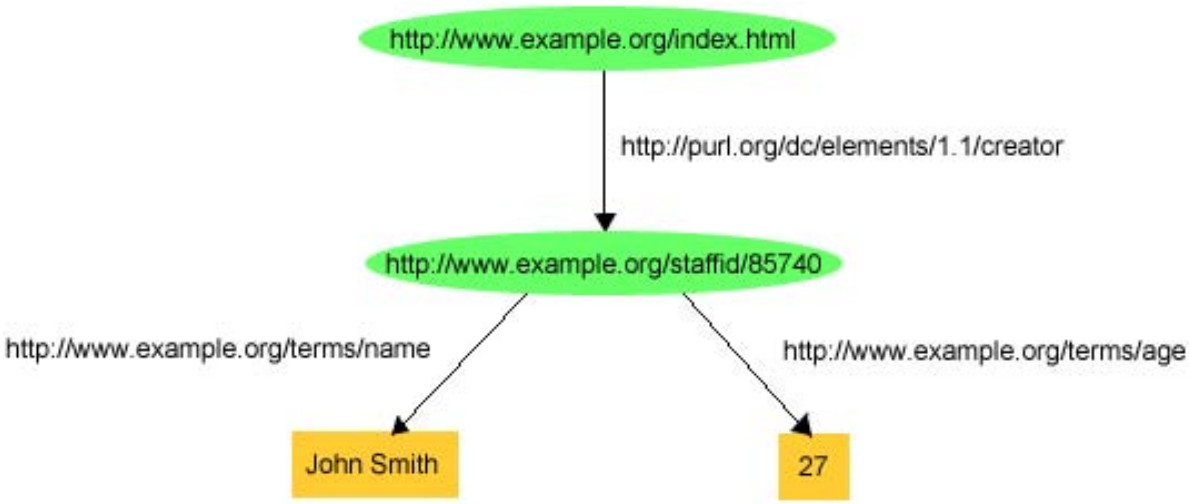


Figure 4: More Information About John Smith

These examples also illustrate that RDF uses URIrefs as *predicates* in RDF statements. That is, rather than using character strings (or words) such as "creator" or "name" to identify properties, RDF uses URIrefs. Using URIrefs to identify properties is important for a number of reasons. First, it distinguishes the properties one person may use from different properties someone else may use that would otherwise be identified by the same character string. For instance, in the example in Figure 4, example.org uses "name" to mean someone's full name written out as a character string literal (e.g., "John Smith"), but someone else may intend "name" to mean something different (e.g., the name of a variable in a piece of program text). A program encountering "name" as a property identifier on the Web (or merging data from multiple sources) would not necessarily be able to distinguish these uses. However, if example.org writes `http://www.example.org/terms/name` for its "name" property, and the other person writes `http://www.domain2.example.org/genealogy/terms/name` for hers, it is clear that there are distinct properties involved (even if a program cannot automatically determine the distinct meanings). Also, using URIrefs to identify properties enables the properties to be treated as resources themselves. Since properties are resources, additional information can be recorded about them (e.g., the English description of what example.org means by "name"), simply by adding additional RDF statements with the property's URIref as the subject.

Using URIrefs as subjects, predicates, and objects in RDF statements supports the development and use of shared vocabularies on the Web, since people can discover and begin using vocabularies already used by others to describe things, reflecting a shared understanding of those concepts. For example, in the triple

```
ex:index.html   dc:creator   exstaff:85740 .
```

the predicate `dc:creator`, when fully expanded as a URIref, is an unambiguous reference to the "creator" attribute in the Dublin Core metadata attribute set (discussed further in Section 6.1), a widely-used set of attributes (properties) for describing information of all kinds. The writer of this triple is effectively saying that the relationship between the Web page (identified by `http://www.example.org/index.html`) and the creator of the page (a distinct person, identified by `http://www.example.org/staffid/85740`) is exactly the concept identified by `http://`

purl.org/dc/elements/1.1/creator. Another person familiar with the Dublin Core vocabulary, or who finds out what dc:creator means (say by looking up its definition on the Web) will know what is meant by this relationship. In addition, based on this understanding, people can write programs to behave in accordance with that meaning when processing triples containing the predicate dc:creator.

Of course, this depends on increasing the general use of URIrefs to refer to things instead of using literals; e.g., using URIrefs like exstaff:85740 and dc:creator instead of character string literals like John Smith and creator. Even then, RDF's use of URIrefs does not solve all identification problems because, for example, people can still use different URIrefs to refer to the same thing. For this reason, it is a good idea to try to use terms from existing vocabularies (such as the Dublin Core) where possible, rather than making up new terms that might overlap with those of some other vocabulary. Appropriate vocabularies for use in specific application areas are being developed all the time, as illustrated by the applications described in Section 6. However, even when synonyms are created, the fact that these different URIrefs are used in the commonly-accessible "Web space" provides the opportunity both to identify equivalences among these different references, and to migrate toward the use of common references.

In addition, it is important to distinguish between any meaning that *RDF itself* associates with terms (such as dc:creator in the previous example) used in RDF statements and additional, *externally-defined* meaning that people (or programs written by those people) might associate with those terms. As a language, RDF directly defines only the graph syntax of subject, predicate, and object triples, certain meanings associated with URIrefs in the rdf: vocabulary, and certain other concepts to be described later. These things are normatively defined in [RDF-CONCEPTS] and [RDF-SEMANTICS]. However, RDF does not define the meanings of terms from other vocabularies, such as dc:creator, that might be used in RDF statements. Specific vocabularies will be created, with specific meanings assigned to the URIrefs defined in them, externally to RDF. RDF statements using URIrefs from these vocabularies may convey the specific meanings associated with those terms to people familiar with these vocabularies, or to RDF applications written to process these vocabularies, without conveying any of these meanings to an arbitrary RDF application *not* specifically written to process these vocabularies.

For example, people can associate meaning with a triple such as

```
ex:index.html  dc:creator  exstaff:85740 .
```

based on the meaning they associate with the appearance of the word "creator" as part of the URIref dc:creator, or based on their understanding of the specific definition of dc:creator in the Dublin Core vocabulary. However, as far as an arbitrary RDF application is concerned the triple might as well be something like

```
fy:joefy.iunm  ed:dsfbups  fytubgg:85740 .
```

as far as any built-in meaning is concerned. Similarly, any natural language text describing the meaning of dc:creator that might be found on the Web provides no additional meaning that an arbitrary RDF application can directly use.

Of course, URIrefs from a particular vocabulary can be used in RDF statements even though a given application may not be able to associate any special meanings with them. For example, generic RDF software would recognize that the above expression is an RDF statement, that ed:dsfbups is the predicate, and so on. It will simply not associate with the triple any special meaning that the vocabulary developer might have associated with a URIref like ed:dsfbups. Moreover, based on their understanding of a given vocabulary, people can write RDF

applications to behave in accordance with the special meanings assigned to URIrefs from that vocabulary, even though that meaning will not be accessible to RDF applications not written in that way.

The result of all this is that RDF provides a way to make statements that applications can more easily process. An application cannot actually "understand" such statements, as noted already, any more than a database system "understands" terms like "employee" or "salary" in processing a query like `SELECT NAME FROM EMPLOYEE WHERE SALARY > 35000`. However, if an application is appropriately written, it can deal with RDF statements in a way that makes it seem like it does understand them, just as a database system and its applications can do useful work in processing employee and payroll information without understanding "employee" and "payroll". For example, a user could search the Web for all book reviews and create an average rating for each book. Then, the user could put that information back on the Web. Another Web site could take that list of book rating averages and create a "Top Ten Highest Rated Books" page. Here, the availability and use of a shared vocabulary about ratings, and a shared group of URIrefs identifying the books they apply to, allows individuals to build a mutually-understood and increasingly-powerful (as additional contributions are made) "information base" about books on the Web. The same principle applies to the vast amounts of information that people create about thousands of subjects every day on the Web.

RDF statements are similar to a number of other formats for recording information, such as:

- entries in a simple record or catalog listing describing the resource in a data processing system.
- rows in a simple relational database.
- simple assertions in formal logic

and information in these formats can be treated as RDF statements, allowing RDF to be used to integrate data from many sources.

2.3 Structured Property Values and Blank Nodes

Things would be very simple if the only types of information to be recorded about things were obviously in the form of the simple RDF statements illustrated so far. However, most real-world data involves structures that are more complicated than that, at least on the surface. For instance, in the original example, the date the Web page was created is recorded as a single `exterms:creation-date` property, with a plain literal as its value. However, suppose the value of the `exterms:creation-date` property needed to record the month, day, and year as separate pieces of information? Or, in the case of John Smith's personal information, suppose John's address was being described. The whole address could be written out as a plain literal, as in the triple

```
exstaff:85740   exterms:address   "1501 Grant Avenue, Bedford,
Massachusetts 01730" .
```

However, suppose John's address needed to be recorded as a *structure* consisting of separate street, city, state, and postal code values? How would this be done in RDF?

Structured information like this is represented in RDF by considering the aggregate thing to be described (like John Smith's address) as a resource, and then making statements about that new resource. So, in the RDF graph, in order to break up John Smith's address into its component parts, a new node is created to represent the concept of John Smith's address, with a new URIref to identify it, say `http://www.example.org/addressid/85740` (abbreviated as `exaddressid:85740`). RDF statements (additional arcs and nodes) can then be written with

that node as the subject, to represent the additional information, producing the graph shown in Figure 5:

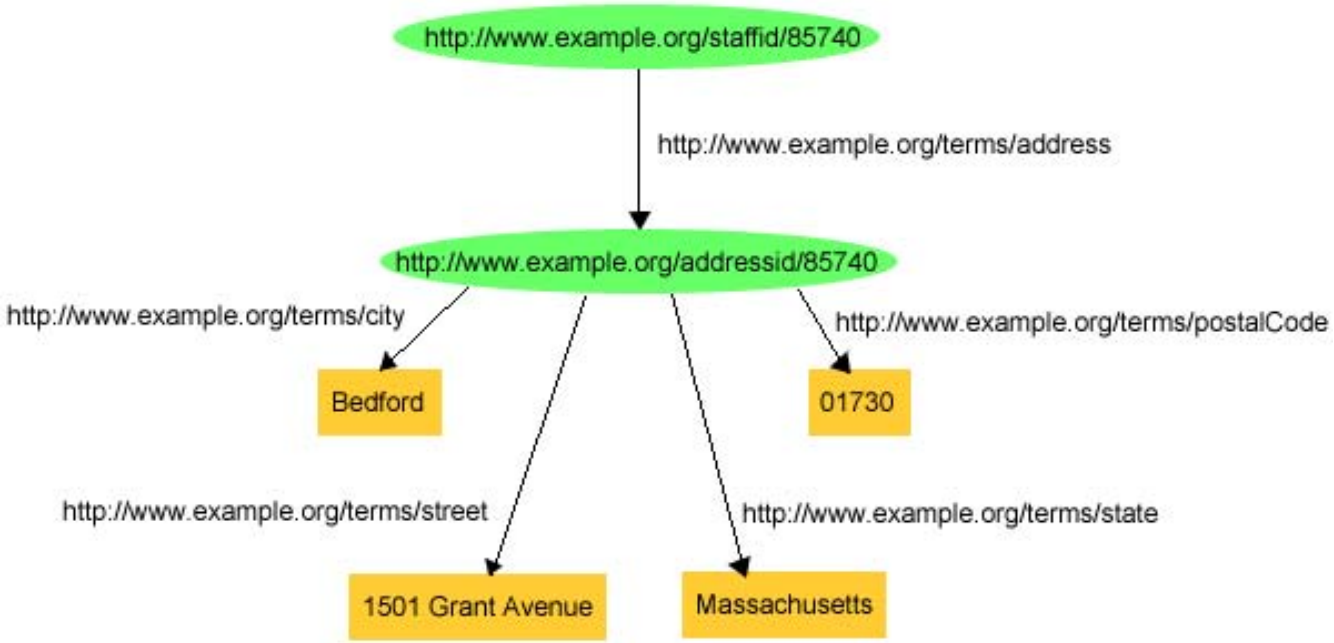


Figure 5: Breaking Up John's Address

or the triples:

```
exstaff:85740       exterms:address        exaddressid:85740 .
exaddressid:85740   exterms:street         "1501 Grant Avenue" .
exaddressid:85740   exterms:city           "Bedford" .
exaddressid:85740   exterms:state          "Massachusetts" .
exaddressid:85740   exterms:postalCode     "01730" .
```

This way of representing structured information in RDF can involve generating numerous "intermediate" URIrefs such as `exaddressid:85740` to represent aggregate concepts such as John's address. Such concepts may never need to be referred to directly from outside a particular graph, and hence may not require "universal" identifiers. In addition, in the *drawing* of the graph representing the group of statements shown in Figure 5, the URIref assigned to identify "John Smith's address" is not really needed, since the graph could just as easily have been drawn as in Figure 6:

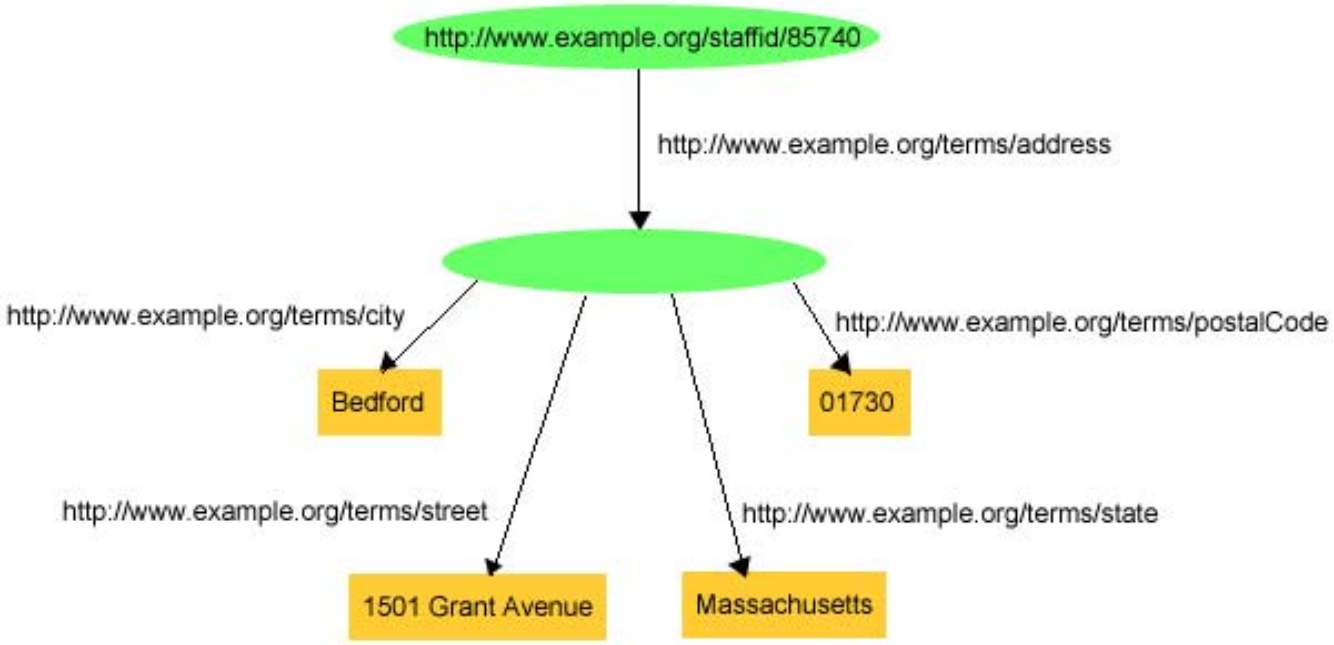


Figure 6: Using a Blank Node

Figure 6, which is a perfectly good RDF graph, uses a node without a URIref to stand for the concept of "John Smith's address". This blank node serves its purpose in the drawing without needing a URIref, since the node itself provides the necessary connectivity between the various other parts of the graph. (Blank nodes were called *anonymous resources* in [RDF-MS].) However, some form of explicit identifier for that node is needed in order to represent this graph as triples. To see this, trying to write the triples corresponding to what is shown in Figure 6 would produce something like:

```
exstaff:85740   exterms:address         ??? .
???             exterms:street          "1501 Grant Avenue" .
???             exterms:city            "Bedford" .
???             exterms:state           "Massachusetts" .
???             exterms:postalCode      "01730" .
```

where ??? stands for something that indicates the presence of the blank node. Since a complex graph might contain more than one blank node, there also needs to be a way to differentiate between these different blank nodes in a triples representation of the graph. As a result, triples use blank node identifiers, having the form `_:name`, to indicate the presence of blank nodes. For instance, in this example a blank node identifier `_:johnaddress` might be used to refer to the blank node, in which case the resulting triples might be:

```
exstaff:85740   exterms:address         _:johnaddress .
_:johnaddress   exterms:street          "1501 Grant Avenue" .
_:johnaddress   exterms:city            "Bedford" .
_:johnaddress   exterms:state           "Massachusetts" .
_:johnaddress   exterms:postalCode      "01730" .
```

In a triples representation of a graph, each distinct blank node in the graph is given a different blank node identifier. Unlike URIrefs and literals, blank node identifiers are not considered to be actual parts of the RDF graph (this can be seen by looking at the drawn graph in Figure 6 and noting that the blank node has no blank node identifier). Blank node identifiers are just a way of representing the blank nodes in a graph (and distinguishing one blank node from another) when the graph is written in triple form. Blank node identifiers also have significance only within the

triples representing a *single* graph (two different graphs with the same number of blank nodes might independently use the same blank node identifiers to distinguish them, and it would be incorrect to assume that blank nodes from different graphs having the same blank node identifiers are the same). If it is expected that a node in a graph will need to be referenced from outside the graph, a URIref should be assigned to identify it. Finally, because blank node identifiers represent (blank) *nodes*, rather than arcs, in the triple form of an RDF graph, blank node identifiers may only appear as subjects or objects in triples; blank node identifiers may not be used as predicates in triples.

The beginning of this section noted that aggregate structures, like John Smith's address, can be represented by considering the aggregate thing to be described as a separate resource, and then making statements about that new resource. This example illustrates an important aspect of RDF: RDF directly represents only *binary* relationships, e.g. the relationship between John Smith and the literal representing his address. Representing the relationship between John and the group of separate *components* of this address involves dealing with an *n-ary* (n-way) relationship (in this case, n=5) between John and the street, city, state, and postal code components. In order to represent such structures directly in RDF (e.g., considering the address as a group of street, city, state, and postal code components), this n-way relationship must be broken up into a group of separate binary relationships. Blank nodes provide one way to do this. For each n-ary relationship, one of the participants is chosen as the subject of the relationship (John in this case), and a blank node is created to represent the rest of the relationship (John's address in this case). The remaining participants in the relationship (such as the city in this example) are then represented as separate properties of the new resource represented by the blank node.

Blank nodes also provide a way to more accurately make statements about resources that may not have URIs, but that are described in terms of relationships with other resources that *do* have URIs. For example, when making statements about a person, say Jane Smith, it may seem natural to use a URI based on that person's email address as her URI, e.g., `mailto:jane@example.org`. However, this approach can cause problems. For example, it may be necessary to record information both about *Jane's mailbox* (e.g., the server it is on) as well as about *Jane herself* (e.g., her current physical address), and using a URIref for Jane based on her email address makes it difficult to know whether it is Jane or her mailbox that is being described. The same problem exists when a company's Web page URL, say `http://www.example.com/`, is used as the URI of the company itself. Once again, it may be necessary to record information about the Web page itself (e.g., who created it and when) as well as about the company, and using `http://www.example.com/` as an identifier for both makes it difficult to know which of these is the actual subject.

The fundamental problem is that using Jane's *mailbox* as a stand-in for *Jane* is not really accurate: Jane and her mailbox are not the same thing, and hence they should be identified differently. When Jane herself does not have a URI, a blank node provides a more accurate way of modeling this situation. Jane can be represented by a blank node, and that blank node used as the subject of a statement with `exterms:mailbox` as the property and the URIref `mailto:jane@example.org` as its value. The blank node could also be described with an `rdf:type` property having a value of `exterms:Person` (types are discussed in more detail in the following sections), an `exterms:name` property having a value of `"Jane Smith"`, and any other descriptive information that might be useful, as shown in the following triples:

```
_:jane   exterms:mailbox   <mailto:jane@example.org> .
_:jane   rdf:type          exterms:Person .
_:jane   exterms:name      "Jane Smith" .
_:jane   exterms:empID     "23748"  .
_:jane   exterms:age       "26" .
```

(Note that `mailto:jane@example.org` is written within angle brackets in the first triple. This is because `mailto:jane@example.org` is a full URIref in the `mailto` URI scheme, rather than a QName abbreviation, and full URIrefs must be enclosed in angle brackets in the triples notation.)

This says, accurately, that "there is a resource of type `exterms:Person`, whose electronic mailbox is identified by `mailto:jane@example.org`, whose name is `Jane Smith`, etc." That is, the blank node can be read as "there is a resource". Statements with that blank node as subject then provide information about the characteristics of that resource.

In practice, using blank nodes instead of URIrefs in these cases does not change the way this kind of information is handled very much. For example, if it is known that an email address uniquely identifies someone at example.org (particularly if the address is unlikely to be reused), that fact can still be used to associate information about that person from multiple sources, even though the email address is not the person's URI. In this case, if some RDF is found on the Web that describes a book, and gives the author's contact information as `mailto:jane@example.org`, it might be reasonable, combining this new information with the previous set of triples, to conclude that the author's name is Jane Smith. The point is that saying something like "the author of the book is `mailto:jane@example.org`" is typically a shorthand for "the author of the book is someone whose mailbox is `mailto:jane@example.org`". Using a blank node to represent this "someone" is just a more accurate way to represent the real world situation. (Incidentally, some RDF-based schema languages allow specifying that certain properties are *unique identifiers* of the resources they describe. This is discussed further in Section 5.5.)

Using blank nodes in this way can also help avoid the use of literals in what might be inappropriate situations. For example, in describing Jane's book, lacking a URIref to identify the author, the publisher might have written (using the publisher's own `ex2terms:` vocabulary):

```
ex2terms:book78354   rdf:type          ex2terms:Book .
ex2terms:book78354   ex2terms:author   "Jane Smith" .
```

However, the author of the book is not really the character string "Jane Smith", but a person whose *name* is Jane Smith. The same information might be more accurately given by the publisher using a blank node, as:

```
ex2terms:book78354   rdf:type          ex2terms:Book .
ex2terms:book78354   ex2terms:author   _:author78354 .
_:author78354        rdf:type          ex2terms:Person .
_:author78354        ex2terms:name     "Jane Smith" .
```

This essentially says "resource `ex2terms:book78354` is of type `ex2terms:Book`, and its author is a resource of type `ex2terms:Person`, whose name is `Jane Smith`." Of course, in this particular case the publisher might instead have assigned its own URIrefs to its authors instead of using blank nodes to identify them, in order to encourage external references to its authors.

Finally, the example above giving Jane's age as 26 illustrates the fact that sometimes the value of a property may appear to be simple, but actually may be more complex. In this case, Jane's age is actually 26 *years*, but the units information (years) is not explicitly given. Such information is often omitted in contexts where it can be safely assumed that anyone accessing the property value will understand the units being used. However, in the wider context of the Web, it is generally *not* safe to make this assumption. For example, a U.S. site might give a weight value in pounds, but someone accessing that data from outside the U.S. might assume that weights are given in kilograms. In general, careful consideration should be given to explicitly representing

units and similar information. This issue is discussed further in Section 4.4, which describes an RDF feature for representing such information as structured values, as well as some other techniques for representing such information.

2.4 Typed Literals

The last section described how to handle situations in which property values represented by plain literals had to be broken up into structured values to represent the individual parts of those literals. Using this approach, instead of, say, recording the date a Web page was created as a single `exterms:creation-date` property, with a single plain literal as its value, the value would be represented as a structure consisting of the month, day, and year as separate pieces of information, using separate plain literals to represent the corresponding values. However, so far, all constant values that serve as objects in RDF statements have been represented by these plain (untyped) literals, even when the intent is probably for the value of the property to be a number (e.g., the value of a `year` or `age` property) or some other kind of more specialized value.

For example, Figure 4 illustrated an RDF graph recording information about John Smith. That graph recorded the value of John Smith's `exterms:age` property as the plain literal "27", as shown in Figure 7:

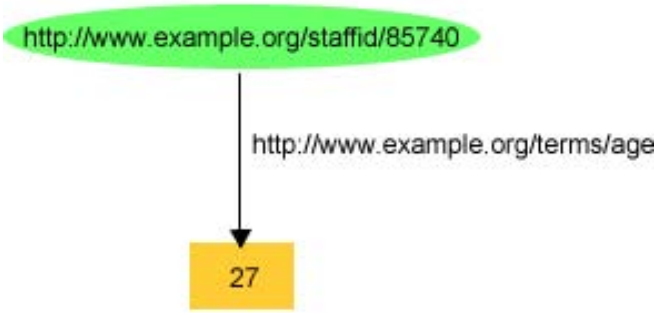


Figure 7: Representing John Smith's Age

In this case, the hypothetical organization example.org probably intends for "27" to be interpreted as a number, rather than as the string consisting of the character "2" followed by the character "7" (since the literal represents the value of an "age" property). However, there is no information in Figure 7's graph that explicitly indicates that "27" should be interpreted as a number. Similarly, example.org also probably intends for "27" to be interpreted as a *decimal* number, i.e., the value *twenty seven*, rather than, say, as an *octal* number, i.e., the value *twenty three*. However, once again there is no information in Figure 7's graph that explicitly indicates this. Specific applications might be written with the understanding that they should interpret values of the `exterms:age` property as decimal numbers, but this would mean that proper interpretation of this RDF would depend on information not explicitly provided in the RDF graph, and hence on information that would not necessarily be available to other applications that might need to interpret this RDF.

The common practice in programming languages or database systems is to provide this additional information about how to interpret a literal by associating a *datatype* with the literal, in this case, a datatype like `decimal` or `integer`. An application that understands the datatype then knows, for example, whether the literal "10" is intended to represent the number *ten*, the number *two*, or the string consisting of the character "1" followed by the character "0", depending on whether the specified datatype is `integer`, `binary`, or `string`. (More specialized datatypes could also be used to include the units information mentioned at the end of Section 2.3, e.g., a datatype `integerYears`, although the Primer will not elaborate on this idea.) In RDF, typed literals are used to provide this kind of information.

An RDF typed literal is formed by pairing a string with a URIref that identifies a particular

datatype. This results in a single literal node in the RDF graph with the pair as the literal. The value represented by the typed literal is the value that the specified datatype associates with the specified string. For example, using a typed literal, John Smith's age could be described as being the integer number *27* using the triple:

```
<http://www.example.org/staffid/85740>  <http://www.example.org/terms/
age> "27"^^<http://www.w3.org/2001/XMLSchema#integer> .
```

or, using the QName simplification for writing long URIs:

```
exstaff:85740  exterms:age  "27"^^xsd:integer .
```

or as shown in Figure 8:

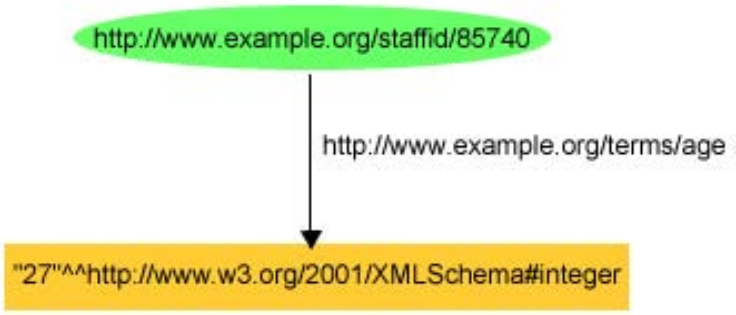


Figure 8: A Typed Literal for John Smith's Age

Similarly, in the graph shown in Figure 3 describing information about a Web page, the value of the page's `exterms:creation-date` property was written as the plain literal "August 16, 1999". However, using a typed literal, the creation date of the Web page could be explicitly described as being the date *August 16, 1999*, using the triple:

```
ex:index.html  exterms:creation-date  "1999-08-16"^^xsd:date .
```

or as shown in Figure 9:

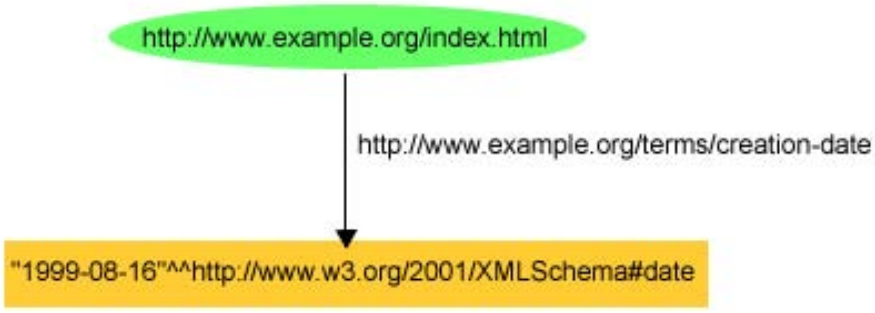


Figure 9: A Typed Literal for a Web Page's Creation Date

Unlike typical programming languages and database systems, RDF has no built-in set of datatypes of its own, such as datatypes for integers, reals, strings, or dates. Instead, RDF typed literals simply provide a way to explicitly indicate, for a given literal, what datatype should be used to interpret it. The datatypes used in typed literals are defined externally to RDF, and identified by their datatype URIs. (There is one exception: RDF defines a built-in datatype with the URIref `rdf:XMLLiteral` to represent XML content as a literal value. This datatype is defined in [RDF-CONCEPTS], and its use is described in Section 4.5.) For instance, the examples in Figure 8 and

Figure 9 use the datatypes `integer` and `date` from the XML Schema datatypes defined in XML Schema Part 2: Datatypes [XML-SCHEMA2]. An advantage of this approach is that it gives RDF the flexibility to directly represent information coming from different sources without the need to perform type conversions between these sources and a native set of RDF datatypes. (Type conversions would still be required when moving information between systems having different sets of datatypes, but RDF would impose no extra conversions into and out of a native set of RDF datatypes.)

RDF datatype concepts are based on a conceptual framework from XML Schema datatypes [XML-SCHEMA2], as described in RDF Concepts and Abstract Syntax [RDF-CONCEPTS]. This conceptual framework defines a datatype as consisting of:

- A set of values, called the *value space*, that literals of the datatype are intended to represent. For example, for the XML Schema datatype `xsd:date`, this set of values is a set of dates.
- A set of character strings, called the *lexical space*, that the datatype uses to represent its values. This set determines which character strings can legally be used to represent literals of this datatype. For example, the datatype `xsd:date` defines `1999-08-16` as being a legal way to write a literal of this type (as opposed, say, to `August 16, 1999`). As defined in [RDF-CONCEPTS], the lexical space of a datatype is a set of Unicode [UNICODE] strings, allowing information from many languages to be directly represented.
- A *lexical-to-value mapping* from the lexical space to the value space. This determines the value that a given character string from the lexical space represents for this particular datatype. For example, the lexical-to-value mapping for datatype `xsd:date` determines that, for this datatype, the string `1999-08-16` represents the date *August 16, 1999*. The lexical-to-value mapping is a factor because the same character string may represent different values for different datatypes.

Not all datatypes are suitable for use in RDF. For a datatype to be suitable for use in RDF, it must conform to the conceptual framework just described. This basically means that, given a character string, the datatype must unambiguously define whether or not the string is in its lexical space, and what value in its value space the string represents. For example, the basic XML Schema datatypes such as `xsd:string`, `xsd:boolean`, `xsd:date`, etc. are suitable for use in RDF. However, some of the built-in XML Schema datatypes are not suitable for use in RDF. For example, `xsd:duration` does not have a well-defined value space, and `xsd:QName` requires an enclosing XML document context. Lists of the XML Schema datatypes that are currently considered suitable and unsuitable for use in RDF are given in [RDF-SEMANTICS].

Since the value that a given typed literal denotes is defined by the typed literal's datatype, and, with the exception of `rdf:XMLLiteral`, RDF does not define any datatypes, the actual interpretation of a typed literal appearing in an RDF graph (e.g., determining the value it denotes) must be performed by software that is written to correctly process not only RDF, but the typed literal's datatype as well. Effectively, this software must be written to process an extended language that includes not only RDF, but also the datatype, as part of its built-in vocabulary. This raises the issue of which datatypes will be generally available in RDF software. Generally, the XML Schema datatypes that are listed as suitable for use in RDF in [RDF-SEMANTICS] have a "first among equals" status in RDF. As noted already, the examples in Figure 8 and Figure 9 used some of these XML Schema datatypes, and the Primer will be using these datatypes in most of its other examples of typed literals as well (for one thing, XML Schema datatypes already have assigned URIrefs that can be used to refer to them, specified in [XML-SCHEMA2]). These XML Schema datatypes are treated no differently than any other datatype, but they are expected to be the most widely used, and therefore the most likely to be interoperable among different software. As a result, it is expected that much RDF software will also be written to process these datatypes. However, RDF software could be written to process other sets of datatypes as well,

assuming they were determined to be suitable for use with RDF, as described already.

In general, RDF software may be called on to process RDF data that contains references to datatypes that the software has not been written to process, in which case there are some things the software will not be able to do. For one thing, with the exception of `rdf:XMLLiteral`, RDF itself does not define the URIrefs that identify datatypes. As a result, RDF software, unless it has been written to recognize specific URIrefs, will not be able to determine whether or not a URIref written in a typed literal actually identifies a datatype. Moreover, even when a URIref does identify a datatype, RDF itself does not define the validity of pairing that datatype with a particular literal. This validity can only be determined by software written to correctly process that particular datatype.

For example, the typed literal in the triple:

```
exstaff:85740  exterms:age  "pumpkin"^^xsd:integer .
```

or the graph shown in Figure 10:

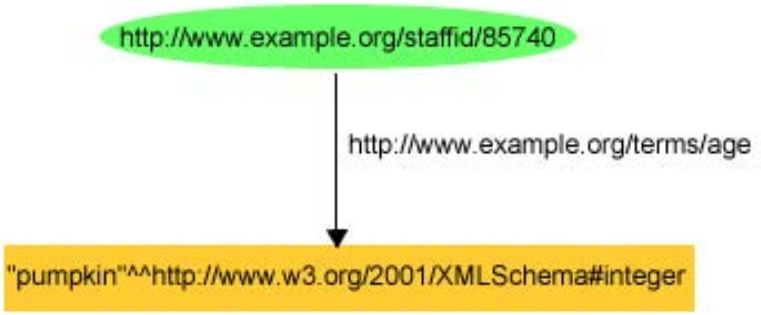


Figure 10: An Invalid Typed Literal for John Smith's Age

is valid RDF, but obviously an error as far as the `xsd:integer` datatype is concerned, since `"pumpkin"` is not defined as being in the lexical space of `xsd:integer`. RDF software not written to process the `xsd:integer` datatype would not be able to recognize this error.

However, proper use of RDF typed literals provides more information about the intended interpretation of literal values, and hence makes RDF statements a better means of information exchange among applications.

2.5 Concepts Summary

Taken as a whole, RDF is basically simple: nodes-and-arcs diagrams interpreted as statements about things identified by URIrefs. This section has presented an introduction to these concepts. As noted earlier, the normative (i.e., definitive) RDF specification describing these concepts is RDF Concepts and Abstract Syntax [RDF-CONCEPTS], which should be consulted for further information. The formal semantics (meaning) of these concepts is defined in the (normative) RDF Semantics [RDF-SEMANTICS] document.

However, in addition to the basic techniques for describing things using RDF statements discussed so far, it should be clear that people or organizations also need a way to describe the *vocabularies* (terms) they intend to use in those statements, specifically, vocabularies for:

- describing types of things (like `exterms:Person`)
- describing properties (like `exterms:age` and `exterms:creation-date`), and
- describing the types of things that can serve as the subjects or objects of statements

involving those properties (such as specifying that the value of an `exterms:age` property should always be an `xsd:integer`).

The basis for describing such vocabularies in RDF is the RDF Vocabulary Description Language 1.0: RDF Schema [RDF-VOCABULARY], which will be described in Section 5.

Additional background on the basic ideas underlying RDF, and its role in providing a general language for describing Web information, can be found in [WEBDATA]. RDF draws upon ideas from knowledge representation, artificial intelligence, and data management, including Conceptual Graphs, logic-based knowledge representation, frames, and relational databases. Some possible sources of background information on these subjects include [SOWA], [CG], [KIF], [HAYES], [LUGER], and [GRAY].

3. An XML Syntax for RDF: RDF/XML

As described in Section 2, RDF's conceptual model is a graph. RDF provides an XML syntax for writing down and exchanging RDF graphs, called *RDF/XML*. Unlike triples, which are intended as a shorthand notation, RDF/XML is the normative syntax for writing RDF. RDF/XML is defined in the RDF/XML Syntax Specification [RDF-SYNTAX]. This section describes this RDF/XML syntax.

3.1 Basic Principles

The basic ideas behind the RDF/XML syntax can be illustrated using some of the examples presented already. Take as an example the English statement:

```
http://www.example.org/index.html has a creation-date whose
value is August 16, 1999
```

The RDF graph for this single statement, after assigning a URIref to the `creation-date` property, is shown in Figure 11:

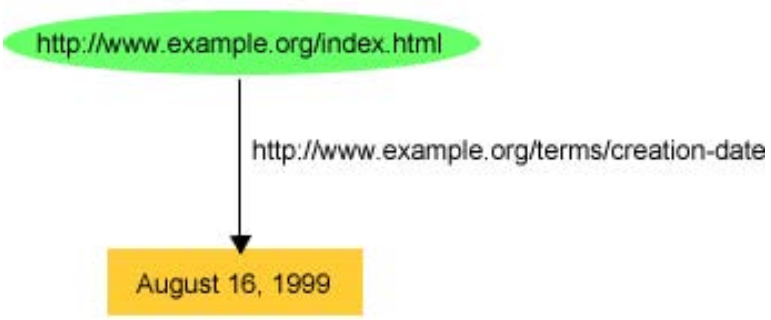


Figure 11: Describing a Web Page's Creation Date

with a triple representation of:

```
ex:index.html    exterms:creation-date    "August 16, 1999" .
```

(Note that a typed literal is not used for the date value in this example. Representing typed literals in RDF/XML will be described later in this section.)

Example 2 shows the RDF/XML syntax corresponding to the graph in Figure 11:

Example 2: RDF/XML for the Web Page's Creation Date

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.           xmlns:exterms="http://www.example.org/terms/">

4.   <rdf:Description rdf:about="http://www.example.org/index.html">
5.       <exterms:creation-date>August 16, 1999</exterms:creation-date>
6.   </rdf:Description>

7. </rdf:RDF>
```

(Line numbers are added to help in explaining the example.)

This seems like a lot of overhead. It is easier to understand what is going on by considering each part of this XML in turn (a brief introduction to XML is provided in Appendix B).

Line 1, `<?xml version="1.0"?>`, is the *XML declaration*, which indicates that the following content is XML, and what version of XML it is.

Line 2 begins an `rdf:RDF` element. This indicates that the following XML content (starting here and ending with the `</rdf:RDF>` in line 7) is intended to represent RDF. Following the `rdf:RDF` on this same line is an XML namespace declaration, represented as an `xmlns` attribute of the `rdf:RDF` start-tag. This declaration specifies that all tags in this content prefixed with `rdf:` are part of the namespace identified by the URIref `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. URIrefs beginning with the string `http://www.w3.org/1999/02/22-rdf-syntax-ns#` are used for terms from the RDF vocabulary.

Line 3 specifies another XML namespace declaration, this time for the prefix `exterms:`. This is expressed as another `xmlns` attribute of the `rdf:RDF` element, and specifies that the namespace URIref `http://www.example.org/terms/` is to be associated with the `exterms:` prefix. URIrefs beginning with the string `http://www.example.org/terms/` are used for terms from the vocabulary defined by the example organization, example.org. The ">" at the end of line 3 indicates the end of the `rdf:RDF` start-tag. Lines 1-3 are general "housekeeping" necessary to indicate that this is RDF/XML content, and to identify the namespaces being used within the RDF/XML content.

Lines 4-6 provide the RDF/XML for the specific statement shown in Figure 11. An obvious way to talk about any RDF statement is to say it is a *description*, and that it is *about* the subject of the statement (in this case, about http://www.example.org/index.html), and this is the way RDF/XML represents the statement. The `rdf:Description` start-tag in line 4 indicates the start of a *description* of a resource, and goes on to identify the resource the statement is *about* (the subject of the statement) using the `rdf:about` attribute to specify the URIref of the subject resource. Line 5 provides a *property element*, with the QName `exterms:creation-date` as its tag, to represent the predicate and object of the statement. The QName `exterms:creation-date` is chosen so that appending the local name `creation-date` to the URIref of the `exterms:` prefix (`http://www.example.org/terms/`) gives the statement's predicate URIref `http://www.example.org/terms/creation-date`. The content of this property element is the object of the statement, the plain literal `August 19, 1999` (the value of the creation-date property of the subject resource). The property element is nested within the containing `rdf:Description` element, indicating that this property applies to the resource specified in the `rdf:about` attribute of the `rdf:Description` element. Line 6 indicates the end of this particular `rdf:Description` element.

Finally, Line 7 indicates the end of the `rdf:RDF` element started on line 2. Using an `rdf:RDF` element to enclose RDF/XML content is optional in situations where the XML can be identified as RDF/XML by context. This is discussed further in [RDF-SYNTAX]. However, it does not hurt to provide the `rdf:RDF` element in any case, and Primer examples will generally (but not always) provide one.

Example 2 illustrates the basic ideas used by RDF/XML to encode an RDF graph as XML elements, attributes, element content, and attribute values. The URIrefs of predicates (as well as some nodes) are written as XML *QNames*, consisting of a short *prefix* denoting a namespace URI, together with a *local name* denoting a namespace-qualified element or attribute, as described in Appendix B. The (namespace URIref, local name) pair is chosen so that concatenating them forms the URIref of the original node or predicate. The URIrefs of subject nodes are written as XML attribute values (URIrefs of object nodes may sometimes be written as attribute values as well). Literal nodes (which are always object nodes) become element text content or attribute values. (Many of these options are described later in the Primer; all of these options are described in [RDF-SYNTAX].)

An RDF graph consisting of multiple statements can be represented in RDF/XML by using RDF/XML similar to Lines 4-6 in Example 2 to separately represent each statement. For example, to write the following two statements:

```
ex:index.html   exterms:creation-date   "August 16, 1999" .
ex:index.html   dc:language             "en" .
```

the RDF/XML in Example 3 could be used:

Example 3: RDF/XML for Two Statements

```
1.  <?xml version="1.0"?>
2.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.              xmlns:dc="http://purl.org/dc/elements/1.1/"
4.              xmlns:exterms="http://www.example.org/terms/">

5.    <rdf:Description rdf:about="http://www.example.org/index.html">
6.        <exterms:creation-date>August 16, 1999</exterms:creation-
date>
7.    </rdf:Description>

8.    <rdf:Description rdf:about="http://www.example.org/index.html">
9.        <dc:language>en</dc:language>
10.   </rdf:Description>

11. </rdf:RDF>
```

Example 3 is the same as Example 2, with the addition of a second `rdf:Description` element (in lines 8-10) to represent the second statement. (An additional namespace declaration is also given in line 3 to identify the additional namespace used in this statement.) An arbitrary number of additional statements could be written in the same way, using a separate `rdf:Description` element for each additional statement. As Example 3 illustrates, once the overhead of writing the XML and namespace declarations is dealt with, writing each additional RDF statement in RDF/XML is both straightforward and not too complicated.

The RDF/XML syntax provides a number of abbreviations to make common uses easier to write. For example, it is typical for the same resource to be described with several properties and

values at the same time, as in Example 3, where the resource `ex:index.html` is the subject of several statements. To handle such cases, RDF/XML allows multiple property elements representing those properties to be nested within the `rdf:Description` element that identifies the subject resource. For example, to represent the following group of statements about `http://www.example.org/index.html`:

```
ex:index.html   dc:creator              exstaff:85740 .
ex:index.html   exterms:creation-date   "August 16, 1999" .
ex:index.html   dc:language             "en" .
```

whose graph (the same as Figure 3) is shown in Figure 12:

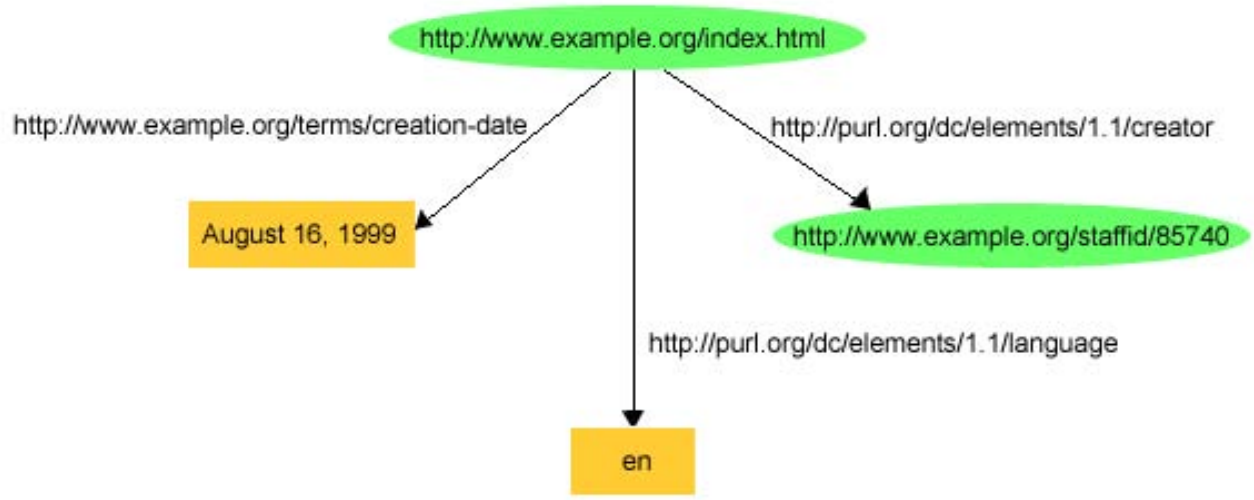


Figure 12: Several Statements About the Same Resource

the RDF/XML shown in Example 4 could be written:

Example 4: Abbreviating Multiple Properties

```
1.  <?xml version="1.0"?>
2.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.             xmlns:dc="http://purl.org/dc/elements/1.1/"
4.             xmlns:exterms="http://www.example.org/terms/">

5.    <rdf:Description rdf:about="http://www.example.org/index.html">
6.         <exterms:creation-date>August 16, 1999</exterms:creation-
date>
7.         <dc:language>en</dc:language>
8.         <dc:creator rdf:resource="http://www.example.org/
staffid/85740"/>
9.    </rdf:Description>

10. </rdf:RDF>
```

Compared with the previous two examples, Example 4 adds an additional `dc:creator` property element (in line 8). In addition, the property elements for the three properties whose subject is `http://www.example.org/index.html` are nested within a single `rdf:Description` element identifying that subject, rather than writing a separate `rdf:Description` element for each statement.

Line 8 also introduces a new form of property element. The `dc:language` element in line 7 is

similar to the `exterms:creation-date` element used in Example 2. Both these elements represent properties with plain literals as property values, and such elements are written by enclosing the literal within start- and end-tags corresponding to the property name. However, the `dc:creator` element on line 8 represents a property whose value is *another resource*, rather than a literal. If the URIref of this resource were written as a plain literal within start- and end-tags in the same way as the literal values of the other elements, this would say that the value of the `dc:creator` element was the *character string* `http://www.example.org/staffid/85740`, rather than the resource identified by that literal interpreted as a URIref. In order to indicate the difference, the `dc:creator` element is written using what XML calls an *empty-element tag* (it has no separate end-tag), and the property value is written using an `rdf:resource` attribute within that empty element. The `rdf:resource` attribute indicates that the property element's value is another resource, identified by its URIref. Because the URIref is being used as an attribute *value*, RDF/XML requires the URIref to be written out (as an absolute or relative URIref), rather than abbreviating it as a QName as was done in writing element and attribute *names* (absolute and relative URIrefs are discussed in Appendix A).

It is important to understand that the RDF/XML in Example 4 is an *abbreviation*. The RDF/XML in Example 5, in which each statement is written separately, describes exactly the same RDF graph (the graph of Figure 12):

Example 5: Writing Example 4 as Separate Statements

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:exterms="http://www.example.org/terms/">

  <rdf:Description rdf:about="http://www.example.org/index.html">
      <exterms:creation-date>August 16, 1999</exterms:creation-date>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.example.org/index.html">
      <dc:language>en</dc:language>
  </rdf:Description>

  <rdf:Description rdf:about="http://www.example.org/index.html">
      <dc:creator rdf:resource="http://www.example.org/staffid/85740"/
>
  </rdf:Description>

</rdf:RDF>
```

The following sections will describe a few additional RDF/XML abbreviations. [RDF-SYNTAX] provides a more thorough description of the abbreviations that are available.

RDF/XML can also represent graphs that include nodes that have no URIrefs, i.e., the *blank nodes* described in Section 2.3. For example, Figure 13 (taken from [RDF-SYNTAX]) shows a graph saying "the document 'http://www.w3.org/TR/rdf-syntax-grammar' has a title 'RDF/XML Syntax Specification (Revised)' and has an editor, the editor has a name 'Dave Beckett' and a home page 'http://purl.org/net/dajobe/' ".

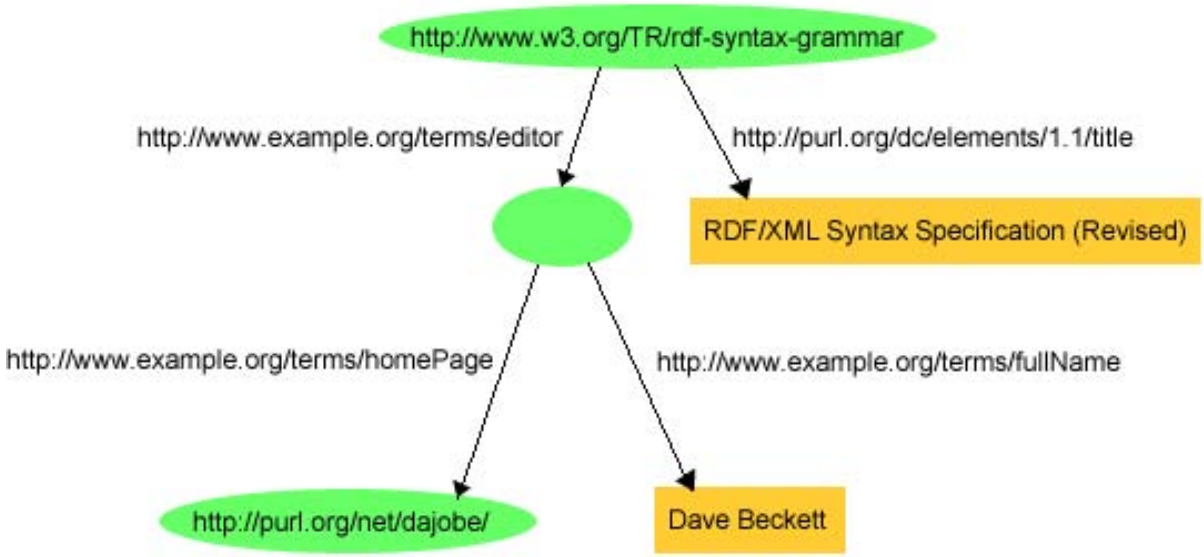


Figure 13: A Graph Containing a Blank Node

This illustrates an idea discussed in Section 2.3: the use of a blank node to represent something that does not have a URIref, but can be described in terms of other information. In this case, the blank node represents a person, the editor of the document, and the person is described by his name and home page.

RDF/XML provides several ways to represent graphs containing blank nodes. These are all described in [RDF-SYNTAX]. The approach illustrated here, which is the most direct approach, is to assign a *blank node identifier* to each blank node. A blank node identifier serves to identify a blank node within a particular RDF/XML document but, unlike a URIref, is unknown outside the document in which it is assigned. A blank node is referred to in RDF/XML using an `rdf:nodeID` attribute, with a blank node identifier as its value, in places where the URIref of a resource would otherwise appear. Specifically, a statement with a blank node as its *subject* can be written in RDF/XML using an `rdf:Description` element with an `rdf:nodeID` attribute instead of an `rdf:about` attribute. Similarly, a statement with a blank node as its *object* can be written using a property element with an `rdf:nodeID` attribute instead of an `rdf:resource` attribute. Using `rdf:nodeID`, Example 6 shows the RDF/XML corresponding to Figure 13:

Example 6: RDF/XML Describing a Blank Node

```
1.  <?xml version="1.0"?>
2.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.             xmlns:dc="http://purl.org/dc/elements/1.1/"
4.             xmlns:exterms="http://example.org/stuff/1.0/">

5.     <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-
grammar">
6.       <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
7.       <exterms:editor rdf:nodeID="abc"/>
8.     </rdf:Description>

9.     <rdf:Description rdf:nodeID="abc">
10.       <exterms:fullName>Dave Beckett</exterms:fullName>
11.       <exterms:homePage rdf:resource="http://purl.org/net/
dajobe/"/>
12.    </rdf:Description>

13. </rdf:RDF>
```

In Example 6, the blank node identifier `abc` is used in line 9 to identify the blank node as the subject of several statements, and is used in line 7 to indicate that the blank node is the value of a resource's `exterms:editor` property. The advantage of using a blank node identifier over some of the other approaches described in [RDF-SYNTAX] is that using a blank node identifier allows the same blank node to be referred to in more than one place in the same RDF/XML document.

Finally, the *typed literals* described in Section 2.4 may be used as property values instead of the plain literals used in the examples so far. A typed literal is represented in RDF/XML by adding an `rdf:datatype` attribute specifying a datatype URIref to the property element containing the literal.

For example, to change the statement in Example 2 to use a typed literal instead of a plain literal for the `exterms:creation-date` property, the triple representation would be:

```
ex:index.html   exterms:creation-date   "1999-08-16"^^xsd:date .
```

with corresponding RDF/XML syntax shown in Example 7:

Example 7: RDF/XML Using a Typed Literal

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.           xmlns:exterms="http://www.example.org/terms/">

4.   <rdf:Description rdf:about="http://www.example.org/index.html">
5.     <exterms:creation-date rdf:datatype=
         "http://www.w3.org/2001/XMLSchema#date">1999-08-16
       </exterms:creation-date>
6.   </rdf:Description>

7. </rdf:RDF>
```

In line 5 of Example 7, a typed literal is given as the value of the `exterms:creation-date` property element by adding an `rdf:datatype` attribute to the element's start-tag to specify the datatype. The value of this attribute is the URIref of the datatype, in this case, the URIref of the XML Schema `date` datatype. Since this is an attribute value, the URIref must be written out, rather than using the QName abbreviation `xsd:date` used in the triple. A literal appropriate to this datatype is then written as the element content, in this case, the literal `1999-08-16`, which is the literal representation for August 16, 1999 in the XML Schema `date` datatype.

In the rest of the Primer, the examples will use typed literals from appropriate datatypes rather than plain (untyped) literals, in order to emphasize the value of typed literals in conveying more information about the intended interpretation of literal values. (The exceptions will be that plain literals will continue to be used in examples taken from actual applications that do not currently use typed literals, in order to accurately reflect the usage in those applications.) In RDF/XML, both plain and typed literals (and, with certain exceptions, tags) can contain Unicode [UNICODE] characters, allowing information from many languages to be directly represented.

Example 7 illustrates that using typed literals requires writing an `rdf:datatype` attribute with a URIref identifying the datatype for each element whose value is a typed literal. As noted earlier, RDF/XML requires that URIrefs used as attribute values must be written out, rather than abbreviated as a QName. XML *entities* can be used in RDF/XML to improve readability in such cases, by providing an additional abbreviation facility for URIrefs. Essentially, an XML entity

declaration associates a name with a string of characters. When the entity name is referenced elsewhere within an XML document, XML processors replace the reference with the corresponding string. For example, the `ENTITY` declaration (specified as part of a `DOCTYPE` declaration at the beginning of the RDF/XML document):

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
```

defines the entity `xsd` to be the string representing the namespace URIref for XML Schema datatypes. This declaration allows the full namespace URIref to be abbreviated elsewhere in the XML document by the *entity reference* `&xsd;`. Using this abbreviation, Example 7 could also be written as shown in Example 8.

Example 8: RDF/XML Using a Typed Literal and an XML Entity

```
1. <?xml version="1.0"?>
2. <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
XMLSchema#">]>

3. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.           xmlns:exterms="http://www.example.org/terms/">

5.   <rdf:Description rdf:about="http://www.example.org/index.html">
6.     <exterms:creation-date rdf:datatype="&xsd;date">1999-08-16
       </exterms:creation-date>
7.   </rdf:Description>

8. </rdf:RDF>
```

The `DOCTYPE` declaration in line 2 defines the entity `xsd`, which is used in line 6.

The use of XML entities as an abbreviation mechanism is optional in RDF/XML, and hence the use of an XML `DOCTYPE` declaration is also optional in RDF/XML. (For readers familiar with XML, RDF/XML is only required to be "well-formed" XML. RDF/XML is not designed to be validated against a DTD by a validating XML processor. This is discussed more fully in Appendix B, which provides additional information about XML.)

For readability purposes, examples in the rest of the Primer will use the XML entity `xsd` as just described. XML entities are discussed further in Appendix B. As illustrated in Appendix B, other URIrefs (and, more generally, other strings) can also be abbreviated using XML entities. However, the URIrefs for XML Schema datatypes are the only ones that will be abbreviated in this way in Primer examples.

Although additional abbreviated forms for writing RDF/XML are available, the facilities illustrated so far provide a simple but general way to express graphs in RDF/XML. Using these facilities, an RDF graph is written in RDF/XML as follows:

- All blank nodes are assigned blank node identifiers.
- Each node is listed in turn as the subject of an un-nested `rdf:Description` element, using an `rdf:about` attribute if the node has a URIref, or an `rdf:nodeID` attribute if the node is blank.
  For each triple with this node as subject, an appropriate property element is created, with either literal content (possibly empty), an `rdf:resource` attribute specifying the object of the triple (if the object node has a URIref), or an `rdf:nodeID` attribute specifying the object of the triple (if the object node is blank).

Compared to some of the more abbreviated approaches described in [RDF-SYNTAX], this simple approach provides the most direct representation of the actual graph structure, and is particularly recommended for applications in which the output RDF/XML is to be used in further RDF processing.

3.2 Abbreviating and Organizing RDF URIrefs

So far, the examples have assumed that the resources being described have been given URIrefs already. For instance, the initial examples provided descriptive information about example.org's Web page, whose URIref was `http://www.example.org/index.html`. This resource was identified in RDF/XML using an `rdf:about` attribute citing its full URIref. Although RDF does not specify or control how URIrefs are assigned to resources, sometimes it is desirable to achieve the effect of assigning URIrefs to resources that are part of an organized group of resources. For example, suppose a sporting goods company, example.com, wanted to provide an RDF-based catalog of its products, such as tents, hiking boots, and so on, as an RDF/XML document, identified by (and located at) `http://www.example.com/2002/04/products`. In that resource, each product might be given a separate RDF description. This catalog, along with one of these descriptions, the catalog entry for a model of tent called the "Overnighter", might be written in RDF/XML as shown in Example 9:

Example 9: RDF/XML for example.com's Catalog

```
1.   <?xml version="1.0"?>
2.   <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
XMLSchema#">]>
3.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.               xmlns:exterms="http://www.example.com/terms/">

5.     <rdf:Description rdf:ID="item10245">
6.          <exterms:model rdf:datatype="&xsd;string">Overnighter</
exterms:model>
7.          <exterms:sleeps rdf:datatype="&xsd;integer">2</exterms:
sleeps>
8.          <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:
weight>
9.          <exterms:packedSize rdf:datatype="&xsd;integer">784</
exterms:packedSize>
10.    </rdf:Description>

  ...other product descriptions...

11.  </rdf:RDF>
```

Example 9 is similar to previous examples in the way it represents the properties (model, sleeping capacity, weight) of the resource (the tent) being described. (The surrounding xml, DOCTYPE, RDF, and namespace information is included in lines 1 through 4, and line 11, but this information would only need to be provided once for the whole catalog, not repeated for each entry in the catalog. Note also that although the *datatypes* associated with the various property values are given explicitly, the *units* associated with some of these property values are not, even though this information should be available to properly interpret the values. Representing units and similar information that may be associated with property values is discussed in Section 4.4. In this example, the value of `exterms:sleeps` is the number of persons the tent can sleep, the value of `exterms:weight` is given in kilograms, and the value of `exterms:packedSize` is given in square centimeters, the area the tent occupies on a backpack.)

An important *difference* from previous examples is that, in line 5, the `rdf:Description` element has an `rdf:ID` attribute instead of an `rdf:about` attribute. Using `rdf:ID` specifies a *fragment identifier*, given by the value of the `rdf:ID` attribute (`item10245` in this case, which might be the catalog number assigned by example.com), as an abbreviation of the complete URIref of the resource being described. The fragment identifier `item10245` will be interpreted relative to a *base URI*, in this case, the URI of the containing catalog document. The full URIref for the tent is formed by taking the base URI (of the catalog), and appending the character "#" (to indicate that what follows is a fragment identifier) and then `item10245` to it, giving the absolute URIref `http://www.example.com/2002/04/products#item10245`.

The `rdf:ID` attribute is somewhat similar to the `ID` attribute in XML and HTML, in that it defines a name which must be unique relative to the current base URI (in this example, that of the catalog). In this case, the `rdf:ID` attribute appears to be assigning a name (`item10245`) to this particular kind of tent. Any other RDF/XML within this catalog could refer to the tent by using either the absolute URIref `http://www.example.com/2002/04/products#item10245`, or the *relative URIref* `#item10245`. The relative URIref would be understood as being a URIref defined relative to the base URIref of the catalog. Using a similar abbreviation, the URIref of the tent could also be given by specifying `rdf:about="#item10245"` in the catalog entry (i.e., by specifying the relative URIref directly) instead of `rdf:ID="item10245"` . As an abbreviation mechanism, the two forms are essentially synonyms: the full URIref formed by RDF/XML is the same in either case: `http://www.example.com/2002/04/products#item10245`. However, using `rdf:ID` provides an additional check when assigning a set of distinct names, since a given value of the `rdf:ID` attribute can only appear once relative to the same base URI (the catalog document, in this example). Using either form, example.com would be giving the URIref for the tent in a two-stage process, first assigning the URIref for the whole catalog, and then using a relative URIref in the description of the tent in the catalog to indicate the URIref that has been assigned to this particular kind of tent. Moreover, this use of a relative URIref can be thought of either as being an abbreviation for a full URIref that has been assigned to the tent independently of the RDF, or as being the assignment of the URIref to the tent within the catalog.

RDF located *outside* the catalog could refer to this tent by using the full URIref, i.e., by concatenating the relative URIref `#item10245` of the tent to the base URI of the catalog, forming the absolute URIref `http://www.example.com/2002/04/products#item10245`. For example, an outdoor sports Web site exampleRatings.com might use RDF to provide ratings of various tents. The (5-star) rating given to the tent described in Example 9 might then be represented on exampleRatings.com's Web site as shown in Example 10:

Example 10: exampleRatings.com's Rating of the Tent

```
1.  <?xml version="1.0"?>
2.  <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
XMLSchema#">]>
3.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.              xmlns:sportex="http://www.exampleRatings.com/terms/">

5.    <rdf:Description rdf:about="http://www.example.com/2002/04/
products#item10245">
6.         <sportex:ratingBy rdf:datatype="&xsd;string">Richard Roe</
sportex:ratingBy>
7.         <sportex:numberStars rdf:datatype="&xsd;integer">5</sportex:
numberStars>
8.    </rdf:Description>
9.  </rdf:RDF>
```

In Example 10, line 5 uses an `rdf:Description` element with an `rdf:about` attribute whose

value is the full URIref of the tent. The use of this URIref allows the tent being referred to in the rating to be precisely identified.

These examples illustrate several points. First, even though RDF does not specify or control how URIrefs are assigned to resources (in this case, the various tents and other items in the catalog), the *effect* of assigning URIrefs to resources in RDF can be achieved by combining a process (external to RDF) that identifies a single document (the catalog in this case) as the source for descriptions of those resources, with the use of relative URIrefs in descriptions of those resources within that document. For instance, example.com could use this catalog as the central source where its products are described, with the understanding that if a product's item number is not in an entry in this catalog, it is not a product known to example.com. (Note that RDF does not assume any particular relationship exists between two resources just because their URIrefs have the same base, or are otherwise similar. This relationship may be known to example.com, but it is not directly defined by RDF.)

These examples also illustrate one of the basic architectural principles of the Web, which is that anyone should be able to freely add information about an existing resource, using any vocabulary they please [BERNERS-LEE98]. The examples further illustrate that the RDF describing a particular resource does not need to be located all in one place; instead, it may be distributed throughout the Web. This is true not only for situations like this one, in which one organization is rating or commenting on a resource defined by another, but also for situations in which the original definer of a resource (or anyone else) wishes to amplify the description of that resource by providing additional information about it. This may be done by modifying the RDF document in which the resource was originally described, to add the properties and values needed to describe the additional information. Or, as this example illustrates, a separate document could be created, providing the additional properties and values in `rdf:Description` elements that refer to the original resource via its URIref using `rdf:about`.

The discussion above indicated that relative URIrefs such as `#item10245` will be interpreted relative to a *base URI*. By default, this base URI would be the URI of the resource in which the relative URIref is used. However, in some cases it is desirable to be able to explicitly specify this base URI. For instance, suppose that in addition to the catalog located at `http://www.example.com/2002/04/products`, example.org wanted to provide a duplicate catalog on a mirror site, say at `http://mirror.example.com/2002/04/products`. This could create a problem, since if the catalog was accessed from the mirror site, the URIref for the example tent would be generated from the URI of the containing document, forming `http://mirror.example.com/2002/04/products#item10245`, rather than `http://www.example.com/2002/04/products#item10245`, and hence would apparently refer to a different resource than the one intended. Alternatively, example.org might want to assign a base URIref for its set of product URIrefs *without* publishing a single source document whose location defines the base.

To deal with such cases, RDF/XML supports XML Base [XML-BASE], which allows an XML document to specify a base URI other than the URI of the document itself. Example 11 shows how the catalog would be described using XML Base:

Example 11: Using XML Base in example.com's Catalog

```
1.   <?xml version="1.0"?>
2.   <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
XMLSchema#">]>
3.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.              xmlns:exterms="http://www.example.com/terms/"
5.              xml:base="http://www.example.com/2002/04/products">
```

```
6.     <rdf:Description rdf:ID="item10245">
7.          <exterms:model rdf:datatype="&xsd;string">Overnighter</
exterms:model>
8.          <exterms:sleeps rdf:datatype="&xsd;integer">2</exterms:
sleeps>
9.          <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:
weight>
10.         <exterms:packedSize rdf:datatype="&xsd;integer">784</
exterms:packedSize>
11.    </rdf:Description>

  ...other product descriptions...

12.  </rdf:RDF>
```

In Example 11, the `xml:base` declaration in line 5 specifies that the base URI for the content within the `rdf:RDF` element (until another `xml:base` attribute is specified) is `http://www.example.com/2002/04/products`, and all relative URIrefs cited within that content will be interpreted relative to that base, no matter what the URI of the containing document is. As a result, the relative URIref of the tent, `#item10245`, will be interpreted as the same absolute URIref, `http://www.example.com/2002/04/products#item10245`, no matter what the actual URI of the catalog document is, or whether the base URIref actually identifies a particular document at all.

So far, the examples have used a single product description, a particular model of tent, from example.com's catalog. However, example.com will probably offer several different models of tents, as well as multiple instances of other categories of products, such as backpacks, hiking boots, and so on. This idea of things being classified into different *kinds* or *categories* is similar to the programming language concept of objects having different *types* or *classes*. RDF supports this concept by providing a predefined property, `rdf:type`. When an RDF resource is described with an `rdf:type` property, the value of that property is considered to be a resource that represents a category or *class* of things, and the subject of that property is considered to be an *instance* of that category or class. Using `rdf:type`, Example 12 shows how example.com might indicate that the product description is that of a tent:

Example 12: Describing a Tent with `rdf:type`

```
1.   <?xml version="1.0"?>
2.   <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
XMLSchema#">]>
3.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.              xmlns:exterms="http://www.example.com/terms/"
5.              xml:base="http://www.example.com/2002/04/products">

6.     <rdf:Description rdf:ID="item10245">
7.          <rdf:type rdf:resource="http://www.example.com/terms/Tent"/
>
8.          <exterms:model rdf:datatype="&xsd;string">Overnighter</
exterms:model>
9.          <exterms:sleeps rdf:datatype="&xsd;integer">2</exterms:
sleeps>
10.         <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:
weight>
11.         <exterms:packedSize rdf:datatype="&xsd;integer">784</
exterms:packedSize>
12.    </rdf:Description>
```

```
  ...other product descriptions...

13.  </rdf:RDF>
```

In Example 12, the `rdf:type` property in line 7 indicates that the resource being described is an instance of the class identified by the URIref `http://www.example.com/terms/Tent`. This assumes that example.com has described its classes as part of the same vocabulary that it uses to describe its other terms (such as the property `exterms:weight`), so the absolute URIref of the class is used to refer to it. If example.com had described these classes as part of the product catalog itself, the relative URIref `#Tent` could have been used to refer to it.

RDF itself does not provide facilities for defining application-specific classes of things, such as `Tent` in this example, or their properties, such as `exterms:weight`. Instead, such classes would be described in an *RDF schema*, using the *RDF Schema* language discussed in Section 5. Other such facilities for describing classes can also be defined, such as the *DAML+OIL* and *OWL* languages described in Section 5.5.

It is fairly common in RDF for resources to have `rdf:type` properties that describe the resources as instances of specific types or classes. Such resources are called *typed nodes* in the graph, or *typed node elements* in the RDF/XML. RDF/XML provides a special abbreviation for describing these typed nodes. In this abbreviation, the `rdf:type` property and its value are removed, and the `rdf:Description` element for the node is replaced by an element whose name is the QName corresponding to the value of the removed `rdf:type` property (a URIref that names a class). Using this abbreviation, example.com's tent from Example 12 could also be described as shown in Example 13:

Example 13: Abbreviating the Tent's Type

```
1.   <?xml version="1.0"?>
2.   <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/
XMLSchema#">]>
3.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.              xmlns:exterms="http://www.example.com/terms/"
5.              xml:base="http://www.example.com/2002/04/products">

6.     <exterms:Tent rdf:ID="item10245">
7.          <exterms:model rdf:datatype="&xsd;string">Overnighter</
exterms:model>
8.          <exterms:sleeps rdf:datatype="&xsd;integer">2</exterms:
sleeps>
9.          <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:
weight>
10.         <exterms:packedSize rdf:datatype="&xsd;integer">784</
exterms:packedSize>
11.    </exterms:Tent>

  ...other product descriptions...

12.  </rdf:RDF>
```

Since a resource may be described as an instance of more than one class, a resource may have more than one `rdf:type` property. However, only one of these `rdf:type` properties can be abbreviated in this way. The others must be written out using `rdf:type` properties, in the

manner illustrated by the `rdf:type` property in Example 12.

In addition to its use in describing instances of user-defined classes such as `exterms:Tent`, the typed node abbreviation is also commonly used in RDF/XML when describing instances of the built-in RDF classes (such as `rdf:Bag`) to be described in Section 4, and the built-in RDF Schema classes (such as `rdfs:Class`) to be described in Section 5.

Both Example 12 and Example 13 illustrate that RDF statements can be written in RDF/XML in a way that closely resembles descriptions that might have been written directly in (non-RDF) XML. This is an important consideration, given the increasing use of XML in all kinds of applications, since it suggests that RDF could be used in these applications without requiring major changes in the way their information is structured.

3.3 RDF/XML Summary

The examples above have illustrated some of the basic ideas behind the RDF/XML syntax. These examples provide enough information to begin writing useful RDF/XML. A more thorough discussion of the principles behind the modeling of RDF statements in XML (known as *striping*), together with a presentation of the other RDF/XML abbreviations available, and other details and examples about writing RDF in XML, is given in the (normative) RDF/XML Syntax Specification [RDF-SYNTAX].

4. Other RDF Capabilities

RDF provides a number of additional capabilities, such as built-in types and properties for representing groups of resources and RDF statements, and capabilities for representing XML fragments as property values. These additional capabilities are described in the following sections.

4.1 RDF Containers

There is often a need to describe *groups* of things: for example, to say that a book was created by several authors, or to list the students in a course, or the software modules in a package. RDF provides several predefined (built-in) types and properties that can be used to describe such groups.

First, RDF provides a *container vocabulary* consisting of three predefined types (together with some associated predefined properties). A *container* is a resource that contains things. The contained things are called *members*. The members of a container may be resources (including blank nodes) or literals. RDF defines three types of containers:

- `rdf:Bag`
- `rdf:Seq`
- `rdf:Alt`

A *Bag* (a resource having type `rdf:Bag`) represents a group of resources or literals, possibly including duplicate members, where there is no significance in the order of the members. For example, a Bag might be used to describe a group of part numbers in which the order of entry or processing of the part numbers does not matter.

A *Sequence* or *Seq* (a resource having type `rdf:Seq`) represents a group of resources or literals, possibly including duplicate members, where the order of the members is significant. For example, a Sequence might be used to describe a group that must be maintained in alphabetical

order.

An *Alternative* or *Alt* (a resource having type `rdf:Alt`) represents a group of resources or literals that are *alternatives* (typically for a single value of a property). For example, an Alt might be used to describe alternative language translations for the title of a book, or to describe a list of alternative Internet sites at which a resource might be found. An application using a property whose value is an Alt container should be aware that it can choose any one of the members of the group as appropriate.

To describe a resource as being one of these types of containers, the resource is given an `rdf:type` property whose value is one of the predefined resources `rdf:Bag`, `rdf:Seq`, or `rdf:Alt` (whichever is appropriate). The container resource (which may either be a blank node or a resource with a URIref) denotes the group as a whole. The *members* of the container can be described by defining a *container membership property* for each member with the container resource as its subject and the member as its object. These container membership properties have names of the form `rdf:_`*n*, where *n* is a decimal integer greater than zero, with no leading zeros, e.g., `rdf:_1`, `rdf:_2`, `rdf:_3`, and so on, and are used specifically for describing the members of containers. Container resources may also have other properties that describe the container, in addition to the container membership properties and the `rdf:type` property.

It is important to understand that while these types of containers are described using predefined RDF types and properties, any special meanings associated with these containers, e.g., that the members of an Alt container are alternative values, are only *intended* meanings. These specific container types, and their definitions, are provided with the aim of establishing a shared convention among those who need to describe groups of things. All RDF does is provide the types and properties that can be used to construct the RDF graphs to describe each type of container. RDF has no more built-in understanding of what a resource of type `rdf:Bag` is than it has of what a resource of type `ex:Tent` (discussed in Section 3.2) is. In each case, applications must be written to behave according to the particular meaning involved for each type. This point will be expanded on in the following examples.

A typical use of a container is to indicate that the value of a property is a group of things. For example, to represent the sentence "Course 6.001 has the students Amy, Mohamed, Johann, Maria, and Phuong", the course could be described by giving it a `s:students` property (from an appropriate vocabulary) whose value is a container of type `rdf:Bag` (representing the group of students). Then, using the container membership properties, individual students could be identified as being members of that group, as in the RDF graph shown in Figure 14:

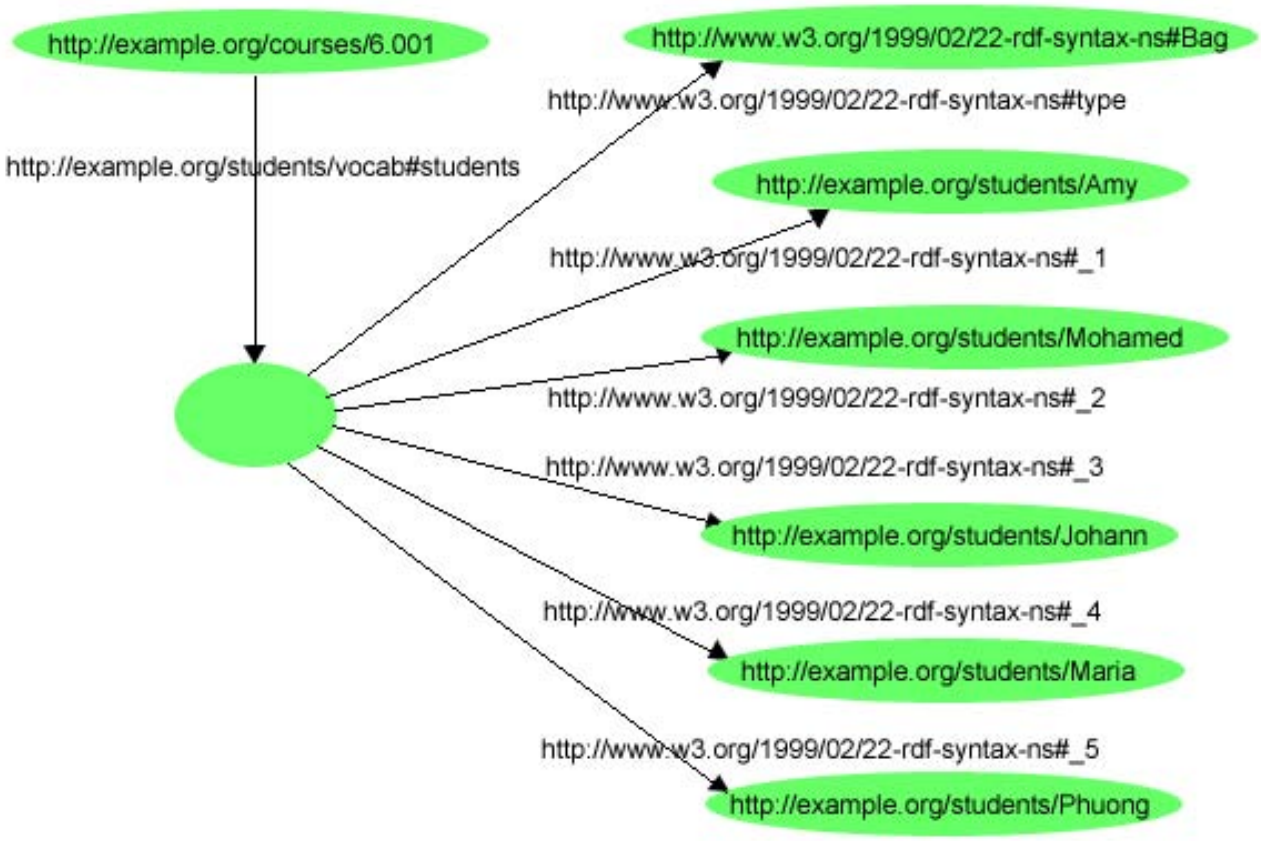


Figure 14: A Simple Bag Container Description

Since the value of the `s:students` property in this example is described as a Bag, there is no intended significance in the order given for the URIrefs of the students, even though the membership properties in the graph have integers in their names. It is up to applications creating and processing graphs that include `rdf:Bag` containers to ignore any (apparent) order in the names of the membership properties.

RDF/XML provides some special syntax and abbreviations to make it simpler to describe such containers. For example, Example 14 describes the graph shown in Figure 14:

Example 14: RDF/XML for a Bag of Students

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">

   <rdf:Description rdf:about="http://example.org/courses/6.001">
      <s:students>
         <rdf:Bag>
            <rdf:li rdf:resource="http://example.org/students/Amy"/>
            <rdf:li rdf:resource="http://example.org/students/Mohamed"/
>
            <rdf:li rdf:resource="http://example.org/students/Johann"/>
            <rdf:li rdf:resource="http://example.org/students/Maria"/>
            <rdf:li rdf:resource="http://example.org/students/Phuong"/>
         </rdf:Bag>
      </s:students>
   </rdf:Description>
</rdf:RDF>
```

Example 14 shows that RDF/XML provides `rdf:li` as a convenience element to avoid having to

explicitly number each membership property. The numbered properties `rdf:_1`, `rdf:_2`, and so on are generated from the `rdf:li` elements in forming the corresponding graph. The element name `rdf:li` was chosen to be mnemonic with the term "list item" from HTML. Note also the use of a `<rdf:Bag>` element nested within the `<s:students>` property element. The `<rdf:Bag>` element is another example of the abbreviation used in Example 13 that replaces both an `rdf:Description` element and an `rdf:type` element with a single element when describing an instance of a type (an instance of `rdf:Bag` in this case). Since no URIref is specified, the Bag is a blank node. Its nesting within the `<s:students>` property element is an abbreviated way of indicating that the blank node is the value of this property. These abbreviations are described further in [RDF-SYNTAX].

The graph structure for an `rdf:Seq` container, and the corresponding RDF/XML, are similar to those for an `rdf:Bag` (the only difference is in the type, `rdf:Seq`). Once again, although an `rdf:Seq` container is intended to describe a sequence, it is up to applications creating and processing the graph to appropriately interpret the sequence of integer-valued property names.

To illustrate an Alt container, the sentence "The source code for X11 may be found at ftp.example.org, ftp1.example.org, or ftp2.example.org" could be expressed in the RDF graph shown in Figure 15:

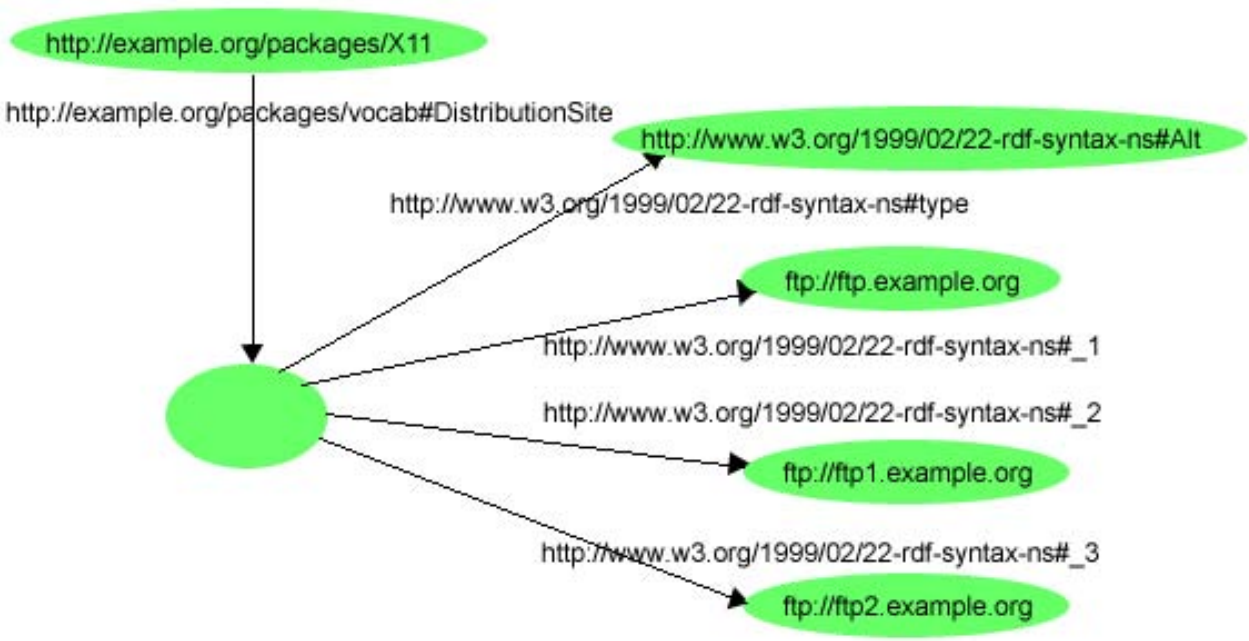


Figure 15: A Simple Alt Container Description

Example 15 shows how the graph in Figure 15 could be written in RDF/XML:

Example 15: RDF/XML for an Alt Container

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/packages/vocab#">

   <rdf:Description rdf:about="http://example.org/packages/X11">
      <s:DistributionSite>
         <rdf:Alt>
            <rdf:li rdf:resource="ftp://ftp.example.org"/>
            <rdf:li rdf:resource="ftp://ftp1.example.org"/>
            <rdf:li rdf:resource="ftp://ftp2.example.org"/>
         </rdf:Alt>
      </s:DistributionSite>
```

```
   </rdf:Description>
</rdf:RDF>
```

An Alt container is intended to have at least one member, identified by the property `rdf:_1`. This member is intended to be considered as the default or preferred value. Other than the member identified as `rdf:_1`, the order of the remaining elements is not significant.

The RDF in Figure 15 *as written* states simply that the value of the `s:DistributionSite` site property is the Alt container resource itself. Any additional meaning that is to be read into this graph, e.g., that one of the *members* of the Alt container is to be considered as the value of the `s:DistributionSite` site property, or that `ftp://ftp.example.org` is the default or preferred value, must be built into an application's understanding of the intended meaning of an Alt container, and/or into the meaning defined for the particular property (`s:DistributionSite` in this case), which also must be understood by the application.

Alt containers are frequently used in conjunction with language tagging. (RDF/XML permits the use of the `xml:lang` attribute defined in [XML] to indicate that the element content is in a specified language. The use of `xml:lang` is described in [RDF-SYNTAX], and illustrated later in Section 6.2.) For example, a work whose title has been translated into several languages might have its `title` property pointing to an Alt container holding literals representing the titles expressed in each of the language variants.

The distinction between the intended meanings of a Bag and an Alt can be further illustrated by considering the authorship of the book "Huckleberry Finn". The book has exactly one author, but the author has two names (Mark Twain and Samuel Clemens). Either name is sufficient to specify the author. Thus using an Alt container for the author's names more accurately represents the relationship than using a Bag (which might suggest there are two *different* authors).

Users are free to choose their own ways to describe groups of resources, rather than using the RDF container vocabulary. These RDF containers are merely provided as common definitions that, if generally used, could help make data involving groups of resources more interoperable.

Sometimes there are clear alternatives to using these RDF container types. For example, a relationship between a particular resource and a group of other resources could be indicated by making the first resource the subject of multiple statements using the same property. This is structurally different from the resource being the subject of a single statement whose object is a container containing multiple members. In some cases, these two structures may have equivalent meaning, but in other cases they may not. The choice of which to use in a given situation should be made with this in mind.

Consider as an example the relationship between a writer and her publications, as in the sentence:

> Sue has written "Anthology of Time", "Zoological Reasoning", and "Gravitational Reflections".

In this case, there are three resources each of which was written independently by the same writer. This could be expressed using repeated properties as:

```
exstaff:Sue   exterms:publication   ex:AnthologyOfTime .
exstaff:Sue   exterms:publication   ex:ZoologicalReasoning .
exstaff:Sue   exterms:publication   ex:GravitationalReflections .
```

In this example there is no stated relationship between the publications other than that they were written by the same person. Each of the statements is an independent fact, and so using repeated properties would be a reasonable choice. However, this could just as reasonably be represented as a statement about the group of resources written by Sue:

```
exstaff:Sue   exterms:publication   _:z .
_:z           rdf:type              rdf:Bag .
_:z           rdf:_1                ex:AnthologyOfTime .
_:z           rdf:_2                ex:ZoologicalReasoning .
_:z           rdf:_3                ex:GravitationalReflections .
```

On the other hand, the sentence:

> The resolution was approved by the Rules Committee, having members Fred, Wilma, and Dino.

says that the committee *as a whole* approved the resolution; it does not necessarily state that each committee member *individually* voted in favor of the resolution. In this case, it would be potentially misleading to model this sentence as three separate `exterms:approvedBy` statements, one for each committee member, as shown below:

```
ex:resolution   exterms:approvedBy   ex:Fred .
ex:resolution   exterms:approvedBy   ex:Wilma .
ex:resolution   exterms:approvedBy   ex:Dino .
```

since these statements say that each member individually approved the resolution.

In this case, it would be better to model the sentence as a single `exterms:approvedBy` statement whose subject is the resolution and whose object is the committee itself. The committee resource could then be described as a Bag whose members are the members of the committee, as in the following triples:

```
ex:resolution       exterms:approvedBy   ex:rulesCommittee .
ex:rulesCommittee   rdf:type             rdf:Bag .
ex:rulesCommittee   rdf:_1               ex:Fred .
ex:rulesCommittee   rdf:_2               ex:Wilma .
ex:rulesCommittee   rdf:_3               ex:Dino .
```

When using RDF containers, it is important to understand that the statements are not *constructing* containers, as in a programming language data structure. Instead, the statements are *describing* containers (groups of things) that presumably exist. For instance, in the Rules Committee example just given, the Rules Committee is an unordered group of people, whether it is described in RDF that way or not. Saying that the resource `ex:rulesCommittee` has type `rdf:Bag` is not saying that the Rules Committee is a data structure, or constructing a particular data structure to hold the members of the group (the Rules Committee could be described as a Bag without describing any members at all). Instead, it is describing the Rules Committee as having characteristics corresponding to those associated with a Bag container, namely that it has members, and their order of description is not significant. Similarly, using the container membership properties simply describes a container resource as having certain things as members. This does not necessarily say that the things described as members are the *only* members that exist. For example, the triples given above to describe the Rules Committee say only that Fred, Wilma, and Dino are members of the committee, not that they are the *only*

members of the committee.

Also, Example 14 and Example 15 illustrated a common "pattern" in describing containers, regardless of the type of container involved (e.g., use of a blank node with an appropriate `rdf:type` property to represent the container itself, and use of `rdf:li` to generate sequentially-numbered container membership properties). However, it is important to understand that RDF does not *enforce* this particular way of using the RDF container vocabulary, and so it is possible to use this vocabulary in other ways. For example, in some cases it might be appropriate to use a container resource having a URIref rather than using a blank node. Moreover, it is possible to use the container vocabulary in ways that may not describe graphs with the "well-formed" structures shown in the previous examples. For example, Example 16 shows the RDF/XML for a graph similar to the Alt container shown in Figure 15, but which writes the container membership properties explicitly, rather than using `rdf:li` to generate them:

Example 16: RDF/XML for an "Ill-Formed" Alt Container

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/packages/vocab#">

 <rdf:Description rdf:about="http://example.org/packages/X11">
    <s:DistributionSite>
       <rdf:Alt>
          <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-
syntax-ns#Bag"/>
          <rdf:_2 rdf:resource="ftp://ftp.example.org"/>
          <rdf:_2 rdf:resource="ftp://ftp1.example.org"/>
          <rdf:_5 rdf:resource="ftp://ftp2.example.org"/>
      </rdf:Alt>
    </s:DistributionSite>
 </rdf:Description>
</rdf:RDF>
```

As noted in [RDF-SEMANTICS], RDF imposes no "well-formedness" conditions on the use of the container vocabulary, so Example 16 is perfectly legal, even though the container is described as *both* a Bag and an Alt, it is described as having two distinct values of the `rdf:_2` property, and it does not have `rdf:_1`, `rdf:_3`, or `rdf:_4` properties.

As a result, RDF applications that require containers to be "well-formed" should be written to check that the container vocabulary is being used appropriately, in order to be fully robust.

4.2 RDF Collections

A limitation of the containers described in Section 4.1 is that there is no way to *close* them, i.e., to say "these are all the members of the container". As noted in Section 4.1, a container only says that certain identified resources are members; it does not say that other members do not exist. Also, while one graph may describe some of the members, there is no way to exclude the possibility that there is another graph somewhere that describes additional members. RDF provides support for describing groups containing only the specified members, in the form of RDF *collections*. An RDF collection is a group of things represented as a list structure in the RDF graph. This list structure is constructed using a predefined *collection vocabulary* consisting of the predefined type `rdf:List`, the predefined properties `rdf:first` and `rdf:rest`, and the predefined resource `rdf:nil`.

To illustrate this, the sentence "The students in course 6.001 are Amy, Mohamed, and Johann" could be represented using the graph shown in Figure 16:

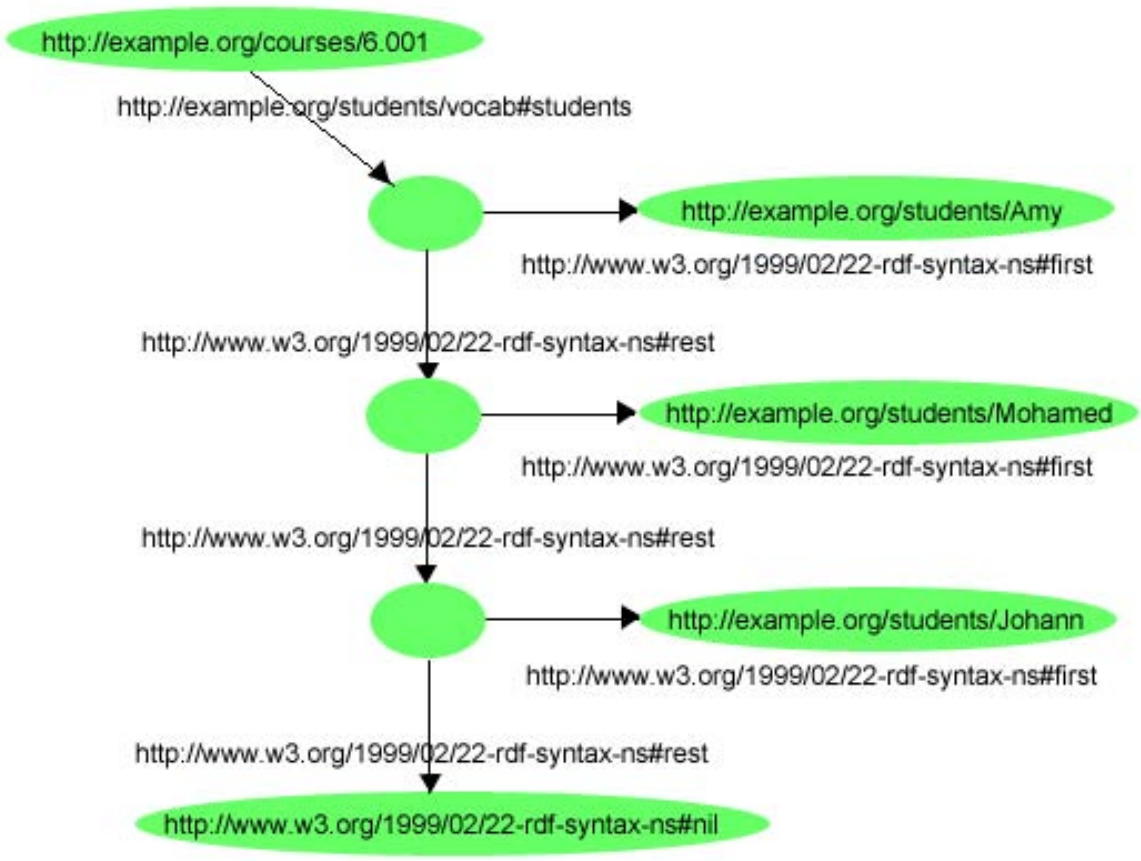


Figure 16: An RDF Collection (list structure)

In this graph, each member of the collection, such as `s:Amy`, is the object of an `rdf:first` property whose subject is a resource (a blank node in this example) that represents a list. This list resource is linked to the rest of the list by an `rdf:rest` property. The end of the list is indicated by the `rdf:rest` property having as its object the resource `rdf:nil` (the resource `rdf:nil` represents the empty list, and is defined as being of type `rdf:List`). This structure will be familiar to those who know the Lisp programming language. As in Lisp, the `rdf:first` and `rdf:rest` properties allow applications to traverse the structure. Each of the blank nodes forming this list structure is implicitly of type `rdf:List` (that is, each of these nodes implicitly has an `rdf:type` property whose value is the predefined type `rdf:List`), although this is not explicitly shown in the graph. The RDF Schema language [RDF-VOCABULARY] defines the properties `rdf:first` and `rdf:rest` as having subjects of type `rdf:List`, so the information about these nodes being lists can generally be inferred, rather than the corresponding `rdf:type` triples being written out all the time.

RDF/XML provides a special notation to make it easy to describe collections using graphs of this form. In RDF/XML, a collection can be described by a property element that has the attribute `rdf:parseType="Collection"`, and that contains a group of nested elements representing the members of the collection. RDF/XML provides the `rdf:parseType` attribute to indicate that the contents of an element are to be interpreted in a special way. In this case, the `rdf:parseType="Collection"` attribute indicates that the enclosed elements are to be used to create the corresponding list structure in the RDF graph (other values of the `rdf:parseType` attribute will be described in later sections of the Primer).

To illustrate how `rdf:parseType="Collection"` works, the RDF/XML from Example 17 would result in the RDF graph shown in Figure 16:

Example 17: RDF/XML for a Collection of Students

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">

   <rdf:Description rdf:about="http://example.org/courses/6.001">
      <s:students rdf:parseType="Collection">
            <rdf:Description rdf:about="http://example.org/students/
Amy"/>
            <rdf:Description rdf:about="http://example.org/students/
Mohamed"/>
            <rdf:Description rdf:about="http://example.org/students/
Johann"/>
      </s:students>
   </rdf:Description>
</rdf:RDF>
```

The use of `rdf:parseType="Collection"` in RDF/XML always defines a list structure like the one shown in Figure 16, i.e., a fixed finite list of items with a given length and terminated by `rdf:nil`, and which uses "new" blank nodes that are unique to the list structure itself. However, RDF does not *enforce* this particular way of using the RDF collection vocabulary, and so it is possible to use this vocabulary in other ways, some of which may not describe lists or closed collections. To see why, note that the graph shown in Figure 16 could also be written in RDF/XML by writing out the same triples "in longhand" (without using `rdf:parseType="Collection"`) using the collection vocabulary, as in Example 18:

Example 18: RDF/XML for a Collection of Students in "Longhand"

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">

<rdf:Description rdf:about="http://example.org/courses/6.001">
   <s:students rdf:nodeID="sch1"/>
</rdf:Description>

<rdf:Description rdf:nodeID="sch1">
   <rdf:first rdf:resource="http://example.org/students/Amy"/>
   <rdf:rest rdf:nodeID="sch2"/>
</rdf:Description>

<rdf:Description rdf:nodeID="sch2">
   <rdf:first rdf:resource="http://example.org/students/Mohamed"/>
   <rdf:rest rdf:nodeID="sch3"/>
</rdf:Description>

<rdf:Description rdf:nodeID="sch3">
   <rdf:first rdf:resource="http://example.org/students/Johann"/>
   <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#nil"/>
</rdf:Description>
</rdf:RDF>
```

As noted in [RDF-SEMANTICS] (and as was the case for the container vocabulary described in Section 4.1), RDF imposes no "well-formedness" conditions on the use of the collection

vocabulary so, when writing triples in longhand, it is possible to define RDF graphs with structures other than the well-structured graphs that would be automatically generated by using `rdf:parseType="Collection"`. For example, it is not illegal to assert that a given node has two distinct values of the `rdf:first` property, to create structures that have forked or non-list tails, or to simply omit part of the description of a collection. Also, graphs defined by using the collection vocabulary in longhand could use URIrefs to identify the components of the list instead of blank nodes unique to the list structure. In this case, it would be possible to create triples in other graphs that effectively added elements to the collection, making it non-closed.

As a result, RDF applications that require collections to be well-formed should be written to check that the collection vocabulary is being used appropriately, in order to be fully robust. In addition, languages such as OWL [OWL], which can define additional constraints on the structure of RDF graphs, can rule out some of these cases.

4.3 RDF Reification

RDF applications sometimes need to describe other RDF statements using RDF, for instance, to record information about when statements were made, who made them, or other similar information (this is sometimes referred to as "provenance" information). For example, Example 9 in Section 3.2 described a particular tent with URIref `exproducts:item10245`, offered for sale by example.com. One of the triples from that description, describing the weight of the tent, was:

```
exproducts:item10245   exterms:weight   "2.4"^^xsd:decimal .
```

and it might be useful for example.com to record who provided that particular piece of information.

RDF provides a built-in vocabulary intended for describing RDF statements. A description of a statement using this vocabulary is called a *reification* of the statement. The RDF reification vocabulary consists of the type `rdf:Statement`, and the properties `rdf:subject`, `rdf:predicate`, and `rdf:object`. However, while RDF provides this reification vocabulary, care is needed in using it, because it is easy to imagine that the vocabulary defines some things that are not actually defined. This point will be discussed further later in this section.

Using the reification vocabulary, a *reification* of the statement about the tent's weight would be given by assigning the statement a URIref such as `exproducts:triple12345` (so statements can be written describing it), and then describing the statement using the statements:

```
exproducts:triple12345   rdf:type        rdf:Statement .
exproducts:triple12345   rdf:subject     exproducts:item10245 .
exproducts:triple12345   rdf:predicate   exterms:weight .
exproducts:triple12345   rdf:object      "2.4"^^xsd:decimal .
```

These statements say that the resource identified by the URIref `exproducts:triple12345` is an RDF statement, that the subject of the statement refers to the resource identified by `exproducts:item10245`, the predicate of the statement refers to the resource identified by `exterms:weight`, and the object of the statement refers to the decimal value identified by the typed literal `"2.4"^^xsd:decimal`. Assuming that the original statement is actually identified by `exproducts:triple12345`, it should be clear by comparing the original statement with the reification that the reification actually does describe it. The conventional use of the RDF reification vocabulary always involves describing a statement using four statements in this pattern; the four statements are sometimes referred to as a "reification quad" for this reason.

Using reification according to this convention, example.com could record the fact that John Smith made the original statement about the tent's weight by first assigning the original statement a URIref (such as `exproducts:triple12345` as before), describing that statement using the reification just described, and then adding an additional statement that `exproducts:triple12345` was written by John Smith (using a URIref to identify which John Smith is being referred to). The resulting statements would be:

```
exproducts:triple12345   rdf:type        rdf:Statement .
exproducts:triple12345   rdf:subject     exproducts:item10245 .
exproducts:triple12345   rdf:predicate   exterms:weight .
exproducts:triple12345   rdf:object      "2.4"^^xsd:decimal .
exproducts:triple12345   dc:creator      exstaff:85740 .
```

The original statement, together with the reification and the attribution of the statement to John Smith, forms the graph shown in Figure 17:

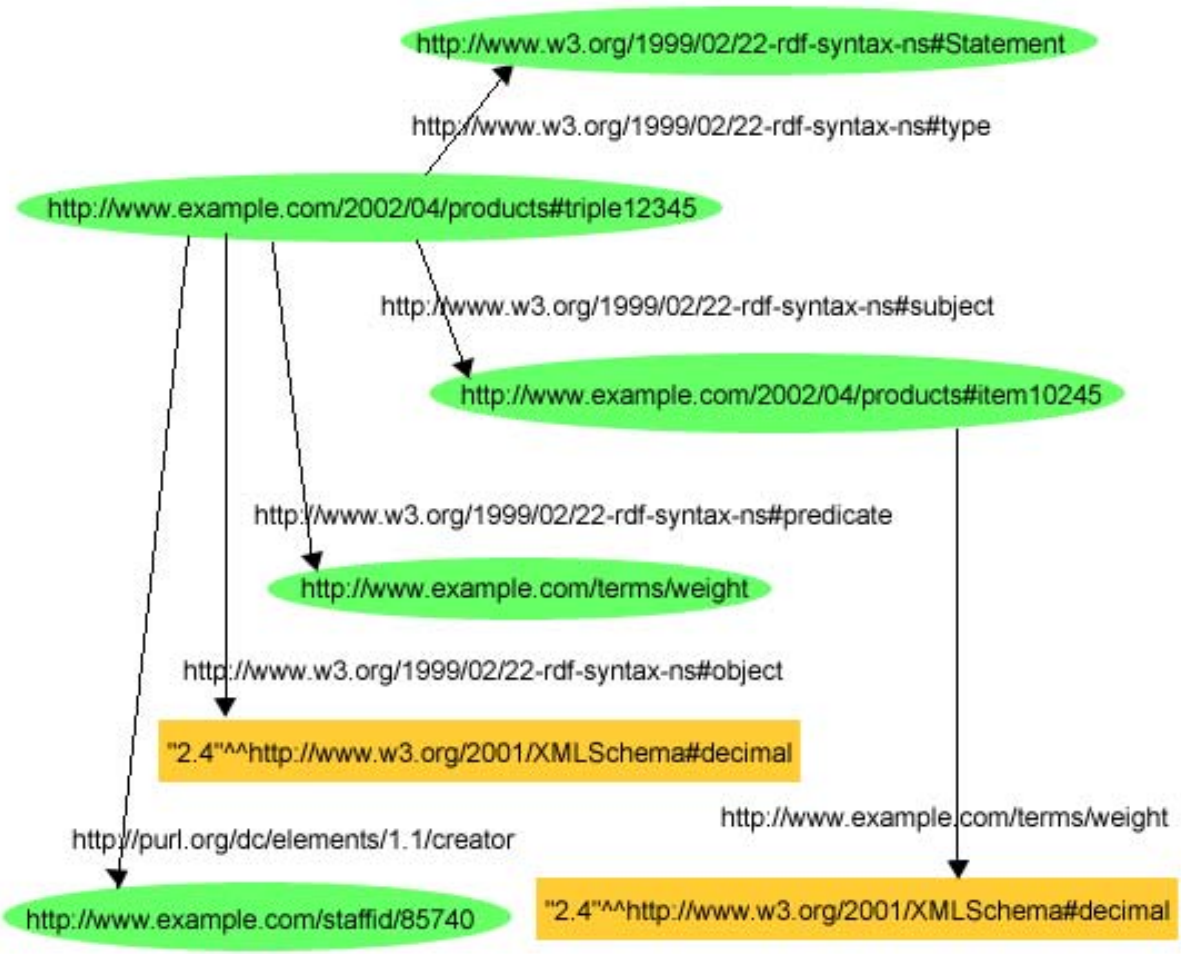


Figure 17: A Statement, Its Reification, and Its Attribution

This graph could be written in RDF/XML as shown in Example 19:

Example 19: RDF/XML for the Reification Example

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:exterms="http://www.example.com/terms/"
            xml:base="http://www.example.com/2002/04/products">
```

```
  <rdf:Description rdf:ID="item10245">
     <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>
  </rdf:Description>

  <rdf:Statement rdf:about="#triple12345">
     <rdf:subject rdf:resource="http://www.example.com/2002/04/
products#item10245"/>
     <rdf:predicate rdf:resource="http://www.example.com/terms/weight"/
>
     <rdf:object rdf:datatype="&xsd;decimal">2.4</rdf:object>

     <dc:creator rdf:resource="http://www.example.com/staffid/85740"/>
  </rdf:Statement>

</rdf:RDF>
```

Section 3.2 introduced the use of the `rdf:ID` attribute in RDF/XML in an `rdf:Description` element to abbreviate the URIref of the subject of a statement. `rdf:ID` can also be used in a property element to automatically produce a reification of the triple that the property element generates. Example 20 shows how this could be used to produce the same graph as Example 19:

Example 20: Generating Reifications using `rdf:ID`

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:exterms="http://www.example.com/terms/"
            xml:base="http://www.example.com/2002/04/products">

  <rdf:Description rdf:ID="item10245">
     <exterms:weight rdf:ID="triple12345" rdf:datatype="&xsd;
decimal">2.4
     </exterms:weight>
  </rdf:Description>

  <rdf:Description rdf:about="#triple12345">
     <dc:creator rdf:resource="http://www.example.com/staffid/85740"/>
  </rdf:Description>

</rdf:RDF>
```

In this case, specifying the attribute `rdf:ID="triple12345"` in the `exterms:weight` element results in the original triple describing the tent's weight:

```
exproducts:item10245   exterms:weight   "2.4"^^xsd:decimal .
```

plus the reification triples:

```
exproducts:triple12345   rdf:type        rdf:Statement .
exproducts:triple12345   rdf:subject     exproducts:item10245 .
exproducts:triple12345   rdf:predicate   exterms:weight .
exproducts:triple12345   rdf:object      "2.4"^^xsd:decimal .
```

The subject of these reification triples is a URIref formed by concatenating the base URI of the document (given in the `xml:base` declaration), the character "#" (to indicate that what follows is a fragment identifier), and the value of the `rdf:ID` attribute; that is, the triples have the same subject `exproducts:triple12345` as in the previous examples.

Note that asserting the reification is not the same as asserting the original statement, and neither implies the other. That is, when someone says that John said something about the weight of a tent, they are not making a statement about the weight of a tent themselves, they are making a statement about something John said. Conversely, when someone describes the weight of a tent, they are not also making a statement about a statement they made (since they may have no intention of talking about things called "statements").

The text above deliberately referred in a number of places to "the conventional use of reification". As noted earlier, care is needed when using the RDF reification vocabulary because it is easy to imagine that the vocabulary defines some things that are not actually defined. While there are applications that successfully use reification, they do so by following some conventions, and making some assumptions, that are in addition to the actual meaning that RDF defines for the reification vocabulary, and the actual facilities that RDF provides to support it.

For one thing, it is important to note that in the conventional use of reification, the subject of the reification triples is assumed to identify a *particular instance* of a triple in a particular RDF document, rather than some arbitrary triple having the same subject, predicate, and object. This particular convention is used because reification is intended for expressing properties such as dates of composition and source information, as in the examples given already, and these properties need to be applied to specific instances of triples. There could be several triples that have the same subject, predicate, and object and, although a graph is defined as a *set* of triples, several instances with the same triple structure might occur in different documents. Thus, to fully support this convention, there needs to be some means of associating the subject of the reification triples with *an individual triple in some document*. However, RDF provides no way to do this.

For instance, in the examples above, there is no explicit information in either the triples or the RDF/XML that actually indicates that the original statement describing the tent's weight is the resource `exproducts:triple12345`, the resource that is the subject of the four reification statements and the statement that John Smith created it. This can be seen by looking at the drawn graph shown in Figure 17. The original statement is certainly part of this graph, but as far as the information in the graph is concerned, `exproducts:triple12345` is a separate resource, rather than identifying that part of the graph. RDF does not provide a built-in way of indicating how a URIref like `exproducts:triple12345` is associated with a particular statement or graph, any more than it provides a built-in way of indicating how a URIref like `exproducts:item10245` is associated with an actual tent. Associating specific URIrefs with specific resources (statements in this case) must be done using mechanisms outside of RDF.

Using `rdf:ID` as shown in Example 20 generates the reification automatically, and provides a convenient way of indicating the URIref to be used as the subject of the statements in the reification. Moreover, it provides a partial "hook" relating the triples in the reification with the piece of RDF/XML syntax that caused them to be created, since the value `triple12345` of the `rdf:ID` attribute is used to generate the URIref of the subject of the reification triples. However, this relationship is once again outside RDF, since there is nothing in the resulting triples that explicitly says that the original triple had the URIref `exproducts:triple12345` (RDF does not assume there is any relationship between a URIref and any RDF/XML that it might have been used or abbreviated in).

The lack of a built-in means for assigning URIrefs to statements does not mean that "provenance" information of this kind cannot be expressed in RDF, just that it cannot be done

using only the meaning RDF associates with the reification vocabulary. For example, if an RDF document (say, a Web page) has a URI, statements could be made about the resource identified by that URI and, based on some application-dependent understanding of how those statements should be interpreted, an application could act as if those statements "distribute" over (apply equally to) all the statements in the document. Also, if some mechanism exists (outside of RDF) to assign URIs to individual RDF statements, then statements could certainly be made about those individual statements, using their URIs to identify them. However, in these cases, it would also not be strictly necessary to use the reification vocabulary in the conventional way.

To see this, assuming the original statement:

```
exproducts:item10245   exterms:weight   "2.4"^^xsd:decimal .
```

had a URIref of `exproducts:triple12345`, the statement could be attributed to John Smith simply by the statement:

```
exproducts:triple12345   dc:creator   exstaff:85740 .
```

with no use of the reification vocabulary (although the description of `exproducts:triple12345` as having `rdf:type rdf:Statement` might also be helpful).

In addition, the reification vocabulary could be used directly according to the convention described above, along with an application-dependent understanding as to how to associate specific triples with their reifications. However, other applications receiving this RDF would not necessarily share this application-dependent understanding, and thus would not necessarily interpret the graphs appropriately.

It is also important to note that the interpretation of reification described here is not the same as "quotation", as found in some languages. Instead, the reification describes the relationship between a particular instance of a triple and the resources the triple refers to. The reification can be read intuitively as saying "this RDF triple talks about these things", rather than (as in quotation) "this RDF triple has this form." For instance, in the reification example used in this section, the triple:

```
exproducts:triple12345   rdf:subject   exproducts:item10245 .
```

describing the `rdf:subject` of the original statement says that the subject of the statement is the resource (the tent) identified by the URIref `exproducts:item10245`. It does *not* say that the subject of the statement is the URIref itself (i.e., a string beginning with certain characters), as quotation would do.

4.4 More on Structured Values: rdf:value

Section 2.3 noted that the RDF model intrinsically supports only *binary* relations; that is, a statement specifies a relation between two resources. For example, the statement:

```
exstaff:85740   exterms:manager   exstaff:62345 .
```

states that the relation `exterms:manager` holds between two employees (presumably one manages the other).

However, in some cases it is necessary to represent information involving higher arity relations (relations between more than two resources) in RDF. Section 2.3 discussed one example of this, where the problem was to represent the relationship between John Smith and his address information, and the value of John's address was a structured value of his street, city, state, and postal code. Writing this as a relation shows that this address is a 5-ary relation of the form:

```
address(exstaff:85740, "1501 Grant Avenue", "Bedford",
"Massachusetts", "01730")
```

Section 2.3 noted that this kind of structured information can be represented in RDF by considering the aggregate thing be described (here, the group of components representing John's address) as a separate resource, and then making separate statements about that new resource, as in the triples:

```
exstaff:85740   exterms:address       _:johnaddress .
_:johnaddress   exterms:street        "1501 Grant Avenue" .
_:johnaddress   exterms:city          "Bedford" .
_:johnaddress   exterms:state         "Massachusetts" .
_:johnaddress   exterms:postalCode    "01730" .
```

(where `_:johnaddress` is the blank node identifier of the blank node representing John's address.)

This is a general way to represent any n-ary relation in RDF: select one of the participants (John in this case) to serve as the subject of the original relation (`address` in this case), then specify an intermediate resource to represent the rest of the relation (either with or without assigning it a URI), then give that new resource properties representing the remaining components of the relation.

In the case of John's address, none of the individual parts of the structured value could be considered the "main" value of the `exterms:address` property; all of the parts contribute equally to the value. However, in some cases one of the parts of the structured value is often thought of as the "main" value, with the other parts of the relation providing additional contextual or other information that qualifies the main value. For instance, in Example 9 in Section 3.2, the weight of a particular tent was given as the decimal value *2.4* using a typed literal, i.e.,

```
exproduct:item10245   exterms:weight   "2.4"^^xsd:decimal .
```

In fact, a more complete description of the weight would have been *2.4 kilograms* rather than just the decimal value *2.4*. To state this, the value of the `exterms:weight` property would need to have two components, the typed literal for the decimal value and an indication of the unit of measure (kilograms). In this situation the decimal value could be considered the "main" value of the `exterms:weight` property, because frequently the value would be recorded simply as the typed literal (as in the triple above), relying on an understanding of the context to fill in the unstated units information.

In the RDF model a qualified property value of this kind can be considered as simply another kind of structured value. To represent this, a separate resource could be used to represent the structured value as a whole (the weight, in this case), and to serve as the object of the original statement. That resource could then be given properties representing the individual parts of the structured value. In this case, there should be a property for the typed literal representing the

decimal value, and a property for the unit. RDF provides a predefined `rdf:value` property to describe the main value (if there is one) of a structured value. So in this case, the typed literal could be given as the value of the `rdf:value` property, and the resource `exunits:kilograms` as the value of an `exterms:units` property (assuming the resource `exunits:kilograms` is defined as part of example.org's vocabulary). The resulting triples would be:

```
exproduct:item10245   exterms:weight   _:weight10245 .
_:weight10245         rdf:value        "2.4"^^xsd:decimal .
_:weight10245         exterms:units    exunits:kilograms .
```

which can be expressed using the RDF/XML shown in Example 21:

Example 21: RDF/XML using `rdf:value`

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:exterms="http://www.example.org/terms/">

  <rdf:Description rdf:about="http://www.example.com/2002/04/
products#item10245">
     <exterms:weight rdf:parseType="Resource">
       <rdf:value rdf:datatype="&xsd;decimal">2.4</rdf:value>
       <exterms:units rdf:resource="http://www.example.org/units/
kilograms"/>
     </exterms:weight>
  </rdf:Description>

</rdf:RDF>
```

Example 21 also illustrates a second use of the `rdf:parseType` attribute introduced in Section 4.2, in this case, `rdf:parseType="Resource"`. An `rdf:parseType="Resource"` attribute is used to indicate that the contents of an element are to be interpreted as the description of a new (blank node) resource, without actually having to write a nested `rdf:Description` element. In this case, the `rdf:parseType="Resource"` attribute used in the `exterms:weight` property element indicates that a blank node is to be created as the value of the `exterms:weight` property, and that the enclosed elements (`rdf:value` and `exterms:units`) describe properties of that blank node. Further details on `rdf:parseType="Resource"` are given in [RDF-SYNTAX].

The same approach can be used to represent quantities using any units of measure, as well as values taken from different classification schemes or rating systems, by using the `rdf:value` property to give the main value, and using additional properties to identify the classification scheme or other information that further describes the value.

There is no need to use `rdf:value` for these purposes (e.g., a user-defined property name, such as `exterms:amount`, could have been used instead of `rdf:value` in Example 21), and RDF does not associate any special meaning with `rdf:value`. `rdf:value` is simply provided as a convenience for use in these commonly-occurring situations.

However, even though much existing data in databases and on the Web (and in later Primer examples) takes the form of simple values for properties such as weights, costs, etc., the principle that such simple values are often insufficient to adequately describe these values is an important one. In a global environment such as the Web, it is generally *not* safe to make the

assumption that anyone accessing a property value will understand the units being used (or other contextually-dependent information that may be involved). For example, a U.S. site might give a weight value in pounds, but someone accessing that data from outside the U.S. might assume that weights are given in kilograms. The correct interpretation of data in the Web environment may require that additional information (such as units information) be explicitly recorded. This can be done in many ways, such as using `rdf:value`, building units into property names (e.g., `exterms:weightInKg`), defining specialized datatypes that include units information (e.g., `extypes:kilograms`), or adding additional user-defined properties to specify this information (e.g., `exterms:unitOfWeight`), either in descriptions of individual items or products, in descriptions of sets of data (e.g., all the data in a catalog or on a site), or in schemas (see Section 5).

4.5 XML Literals

Sometimes the value of a property needs to be a fragment of XML, or text that might contain XML markup. For example, a publisher might maintain RDF metadata that includes the titles of books and articles. While such titles are often just simple strings of characters, this is not always the case. For instance, the titles of books on mathematics may contain mathematical formulas that could be represented using MathML [MATHML]. Titles might also include markup for other reasons, such as for Ruby annotations [RUBY], or for bidirectional rendering or special glyph variants (see, e.g., [CHARMOD]).

RDF/XML provides a special notation to make it easy to write literals of this kind. This is done using a third value of the `rdf:parseType` attribute. Giving an element the attribute `rdf:parseType="Literal"` indicates that the contents of the element are to be interpreted as an XML fragment. Example 22 illustrates the use of `rdf:parseType="Literal"`:

Example 22: RDF/XML for an XML Literal

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc="http://purl.org/dc/elements/1.1/"
           xml:base="http://www.example.com/books">

  <rdf:Description rdf:ID="book12345">
     <dc:title rdf:parseType="Literal">
       <span xml:lang="en">
         The <em>&lt;br /&gt;</em> Element Considered Harmful.
       </span>
     </dc:title>
  </rdf:Description>

</rdf:RDF>
```

The RDF/XML in Example 22 describes a graph containing a single triple with subject `ex:book12345`, and predicate `dc:title`. The `rdf:parseType="Literal"` attribute in the RDF/XML indicates that all the XML within the `<dc:title>` element is an XML fragment that is the value of the `dc:title` property. In the graph, this value is a typed literal, whose datatype, `rdf:XMLLiteral`, is defined in [RDF-CONCEPTS] specifically to represent fragments of XML (including character sequences that may or may not include XML markup). The XML fragment is canonicalized according to the XML Exclusive Canonicalization recommendation [XML-XC14N]. This causes declarations of used namespaces to be added to the fragment, the uniform escaping or unescaping of characters, the expansion of empty-element tags, and other transformations. (For these reasons, and the fact that the triples notation itself requires further escaping, the

actual typed literal is not shown here. RDF/XML provides the `rdf:parseType="Literal"` attribute so that RDF users will *not* have to deal directly with these transformations. Those interested in the details should consult [RDF-CONCEPTS] and [RDF-SYNTAX].) Contextual attributes, such as `xml:lang` and `xml:base` are not inherited from the RDF/XML document, and, if required, must, as shown in the example, be explicitly specified in the XML fragment.

This example illustrates that care must be taken in designing RDF data. It might appear at first glance that titles are simple strings best represented as plain literals, and only later might it be discovered that some titles contain markup. In cases where the value of a property may sometimes contain markup and sometimes not, either `rdf:parseType="Literal"` should be used throughout, or software must handle both plain literals and literals of type `rdf:XMLLiteral` as values of the property.

5. Defining RDF Vocabularies: RDF Schema

RDF provides a way to express simple statements about resources, using named properties and values. However, RDF user communities also need the ability to define the *vocabularies* (terms) they intend to use in those statements, specifically, to indicate that they are describing specific kinds or classes of resources, and will use specific properties in describing those resources. For example, the company example.com from the examples in Section 3.2 would want to describe classes such as `exterms:Tent`, and use properties such as `exterms:model`, `exterms:weightInKg`, and `exterms:packedSize` to describe them (QNames with various "example" namespace prefixes are used as the names of classes and properties here as a reminder that in RDF these names are actually *URI references*, as discussed in Section 2.1). Similarly, people interested in describing bibliographic resources would want to describe classes such as `ex2:Book` or `ex2:MagazineArticle`, and use properties such as `ex2:author`, `ex2:title`, and `ex2:subject` to describe them. Other applications might need to describe classes such as `ex3:Person` and `ex3:Company`, and properties such as `ex3:age`, `ex3:jobTitle`, `ex3:stockSymbol`, and `ex3:numberOfEmployees`. RDF itself provides no means for defining such application-specific classes and properties. Instead, such classes and properties are described as an RDF vocabulary, using extensions to RDF provided by the RDF Vocabulary Description Language 1.0: RDF Schema [RDF-VOCABULARY], referred to here as *RDF Schema*.

RDF Schema does not provide a vocabulary of application-specific classes like `exterms:Tent`, `ex2:Book`, or `ex3:Person`, and properties like `exterms:weightInKg`, `ex2:author` or `ex3:JobTitle`. Instead, it provides the facilities needed to *describe* such classes and properties, and to indicate which classes and properties are expected to be used together (for example, to say that the property `ex3:jobTitle` will be used in describing a `ex3:Person`). In other words, RDF Schema provides a *type system* for RDF. The RDF Schema type system is similar in some respects to the type systems of object-oriented programming languages such as Java. For example, RDF Schema allows resources to be defined as instances of one or more *classes*. In addition, it allows classes to be organized in a hierarchical fashion; for example a class `ex:Dog` might be defined as a subclass of `ex:Mammal` which is a subclass of `ex:Animal`, meaning that any resource which is in class `ex:Dog` is also implicitly in class `ex:Animal` as well. However, RDF classes and properties are in some respects very different from programming language types. RDF class and property descriptions do not create a straightjacket into which information must be forced, but instead provide additional information about the RDF resources they describe. This information can be used in a variety of ways, which will be discussed in Section 5.3.

The RDF Schema facilities are themselves provided in the form of an RDF vocabulary; that is, as a specialized set of predefined RDF resources with their own special meanings. The resources in the RDF Schema vocabulary have URIrefs with the prefix `http://www.w3.org/2000/01/rdf-schema#` (conventionally associated with the QName prefix `rdfs:`). Vocabulary

descriptions (schemas) written in the RDF Schema language are legal RDF graphs. Hence, RDF software that is not written to also process the additional RDF Schema vocabulary can still interpret a schema as a legal RDF graph consisting of various resources and properties, but will not "understand" the additional built-in meanings of the RDF Schema terms. To understand these additional meanings, RDF software must be written to process an extended language that includes not only the `rdf:` vocabulary, but also the `rdfs:` vocabulary, together with their built-in meanings. This point will be illustrated in the next section.

The following sections will illustrate RDF Schema's basic resources and properties.

5.1 Describing Classes

A basic step in any kind of description process is identifying the various kinds of things to be described. RDF Schema refers to these "kinds of things" as *classes*. A *class* in RDF Schema corresponds to the generic concept of a *Type* or *Category*, somewhat like the notion of a class in object-oriented programming languages such as Java. RDF classes can be used to represent almost any category of thing, such as Web pages, people, document types, databases or abstract concepts. Classes are described using the RDF Schema resources `rdfs:Class` and `rdfs:Resource`, and the properties `rdf:type` and `rdfs:subClassOf`.

For example, suppose an organization `example.org` wanted to use RDF to provide information about different kinds of motor vehicles. In RDF Schema, `example.org` would first need a class to represent the category of things that are motor vehicles. The resources that belong to a class are called its *instances*. In this case, `example.org` intends for the instances of this class to be resources that are motor vehicles.

In RDF Schema, a *class* is any resource having an `rdf:type` property whose value is the resource `rdfs:Class`. So the motor vehicle class would be described by assigning the class a URIref, say `ex:MotorVehicle` (using `ex:` to stand for the URIref `http://www.example.org/schemas/vehicles`, which is used as the prefix for URIrefs from example.org's vocabulary) and describing that resource with an `rdf:type` property whose value is the resource `rdfs:Class`. That is, `example.org` would write the RDF statement:

```
ex:MotorVehicle   rdf:type   rdfs:Class .
```

As indicated in Section 3.2, the property `rdf:type` is used to indicate that a resource is an instance of a class. So, having described `ex:MotorVehicle` as a class, resource `exthings:companyCar` would be described as a motor vehicle by the RDF statement:

```
exthings:companyCar   rdf:type   ex:MotorVehicle .
```

(This statement uses a common convention that class names are written with an initial uppercase letter, while property and instance names are written with an initial lowercase letter. However, this convention is not required in RDF Schema. The statement also assumes that `example.org` has decided to define separate vocabularies for classes of things, and instances of things.)

The resource `rdfs:Class` itself has an `rdf:type` of `rdfs:Class`. A resource may be an instance of more than one class.

After describing class `ex:MotorVehicle`, `example.org` might want to describe additional classes representing various specialized kinds of motor vehicle, e.g., passenger vehicles, vans, minivans, and so on. These classes can be described in the same way as class `ex:MotorVehicle`, by assigning a URIref for each new class, and writing RDF statements

describing these resources as classes, e.g., writing:

```
ex:Van     rdf:type   rdfs:Class .
ex:Truck   rdf:type   rdfs:Class .
```

and so on. However, these statements by themselves only describe the individual classes. `example.org` may also want to indicate their special relationship to class `ex:MotorVehicle`, i. e., that they are specialized *kinds* of MotorVehicle.

This kind of specialization relationship between two classes is described using the predefined `rdfs:subClassOf` property to relate the two classes. For example, to state that `ex:Van` is a specialized kind of `ex:MotorVehicle`, `example.org` would write the RDF statement:

```
ex:Van   rdfs:subClassOf   ex:MotorVehicle .
```

The meaning of this `rdfs:subClassOf` relationship is that any instance of class `ex:Van` is also an instance of class `ex:MotorVehicle`. So if resource `exthings:companyVan` is an instance of `ex:Van` then, based on the declared `rdfs:subClassOf` relationship, RDF software written to understand the RDF Schema vocabulary can infer the additional information that `exthings:companyVan` is also an instance of `ex:MotorVehicle`.

This example of `exthings:companyVan` illustrates the point made earlier about RDF Schema defining an extended language. RDF itself does not define the special meaning of terms from the RDF Schema vocabulary such as `rdfs:subClassOf`. So if an RDF schema defines this `rdfs:subClassOf` relationship between `ex:Van` and `ex:MotorVehicle`, RDF software not written to understand the RDF Schema terms would recognize this as a triple, with predicate `rdfs:subClassOf`, but it would not understand the special significance of `rdfs:subClassOf`, and not be able to draw the additional inference that `exthings:companyVan` is also an instance of `ex:MotorVehicle`.

The `rdfs:subClassOf` property is *transitive*. This means, for example, that given the RDF statements:

```
ex:Van       rdfs:subClassOf   ex:MotorVehicle .
ex:MiniVan   rdfs:subClassOf   ex:Van .
```

RDF Schema defines `ex:MiniVan` as also being a subclass of `ex:MotorVehicle`. As a result, RDF Schema defines resources that are instances of class `ex:MiniVan` as also being instances of class `ex:MotorVehicle` (as well as being instances of class `ex:Van`). A class may be a subclass of more than one class (for example, `ex:MiniVan` may be a subclass of both `ex:Van` and `ex:PassengerVehicle`). RDF Schema defines all classes as subclasses of class `rdfs:Resource` (since the instances belonging to all classes are resources).

Figure 18 shows the full class hierarchy being discussed in these examples.

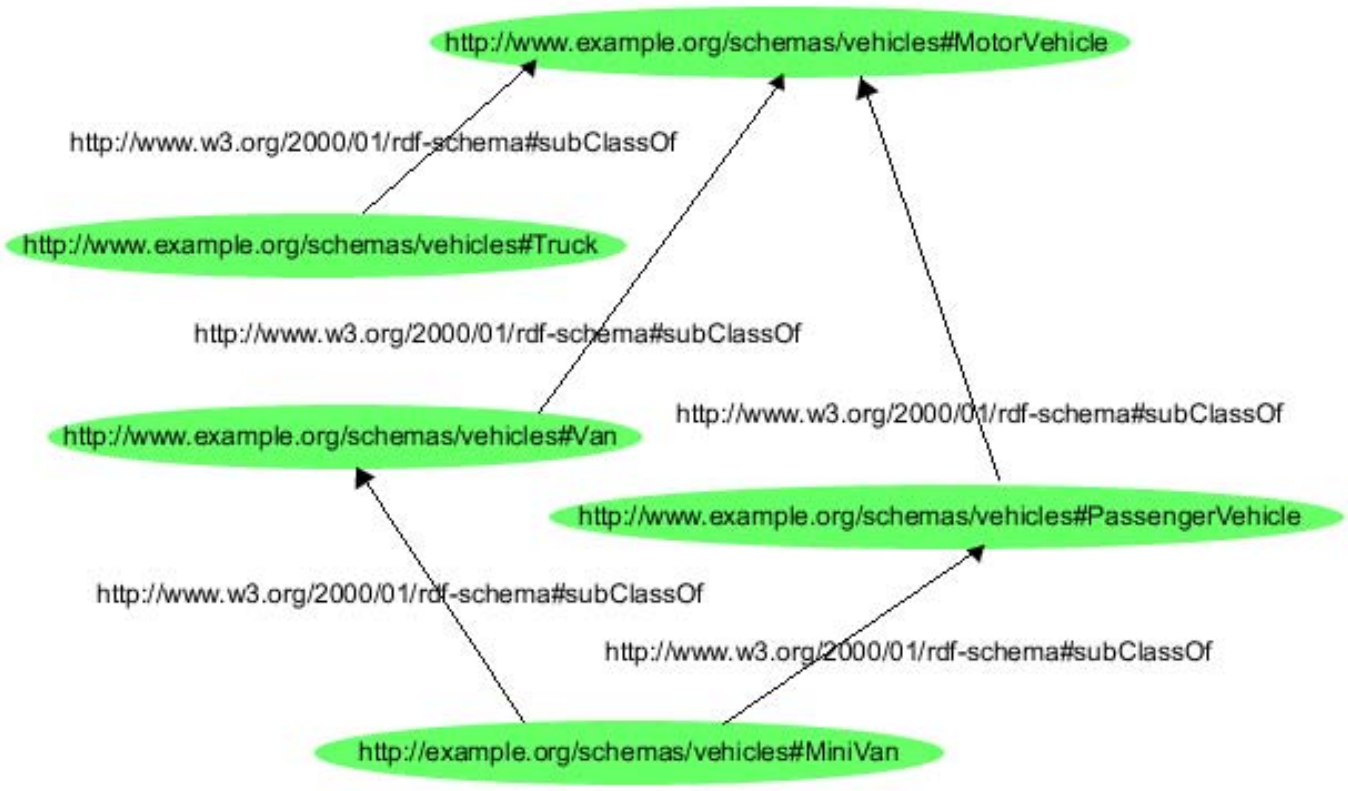


Figure 18: A Vehicle Class Hierarchy

(To simplify the figure, the `rdf:type` properties relating each of the classes to `rdfs:Class` are omitted in Figure 18. In fact, RDF Schema defines both the subjects and objects of statements that use the `rdfs:subClassOf` property to be resources of type `rdfs:Class`, so this information could be inferred. However, in actually writing schemas, it is good practice to explicitly provide this information.)

This schema could also be described by the triples:

```
ex:MotorVehicle       rdf:type          rdfs:Class .
ex:PassengerVehicle   rdf:type          rdfs:Class .
ex:Van                rdf:type          rdfs:Class .
ex:Truck              rdf:type          rdfs:Class .
ex:MiniVan            rdf:type          rdfs:Class .

ex:PassengerVehicle   rdfs:subClassOf   ex:MotorVehicle .
ex:Van                rdfs:subClassOf   ex:MotorVehicle .
ex:Truck              rdfs:subClassOf   ex:MotorVehicle .

ex:MiniVan            rdfs:subClassOf   ex:Van .
ex:MiniVan            rdfs:subClassOf   ex:PassengerVehicle .
```

Example 23 shows how this schema could be written in RDF/XML.

Example 23: The Vehicle Class Hierarchy in RDF/XML

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">
```

```
<rdf:Description rdf:ID="MotorVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="PassengerVehicle">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="Truck">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="Van">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="MiniVan">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>

</rdf:RDF>
```

As discussed in Section 3.2 in connection with Example 13, RDF/XML provides an abbreviation for describing resources having an `rdf:type` property (*typed nodes*). Since RDF Schema classes are RDF resources, this abbreviation can be applied to the description of classes. Using this abbreviation, the schema could also be described as shown in Example 24:

Example 24: The Vehicle Class Hierarchy Using the Typed Node Abbreviation

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">

<rdfs:Class rdf:ID="MotorVehicle"/>

<rdfs:Class rdf:ID="PassengerVehicle">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Truck">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Van">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:ID="MiniVan">
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdfs:Class>

</rdf:RDF>
```

Similar typed node abbreviations will be used throughout the rest of this section.

The RDF/XML in Example 23 and Example 24 introduces names, such as `MotorVehicle`, for the resources (classes) that it describes using `rdf:ID`, to give the effect of "assigning" URIrefs relative to the schema document as described in Section 3.2. `rdf:ID` is useful here because it both abbreviates the URIrefs, and also provides an additional check that the value of the `rdf:ID` attribute is unique against the current base URI (usually the document URI). This helps pick up repeated `rdf:ID` values when defining the names of classes and properties in RDF schemas. Relative URIrefs based on these names can then be used in other class definitions within the same schema (e.g., as `#MotorVehicle` is used in the description of the other classes). The full URIref of this class, assuming that the schema itself was the resource `http://example.org/schemas/vehicles`, would be `http://example.org/schemas/vehicles#MotorVehicle` (shown in Figure 18). As noted in Section 3.2, to ensure that the references to these schema classes would be consistently maintained even if the schema were relocated or copied (or to simply assign a base URIref for the schema classes without assuming they are all published at a single location), the class descriptions could also include an explicit `xml:base="http://example.org/schemas/vehicles"` declaration. Use of an explicit `xml:base` declaration is considered good practice, and one is provided in both examples.

To refer to these classes in RDF instance data (e.g., data describing individual vehicles of these classes) located elsewhere, `example.org` would need to identify the classes either by writing absolute URIrefs, by using relative URIrefs together with an appropriate `xml:base` declaration, or by using QNames together with an appropriate namespace declaration that allows the QNames to be expanded to the proper URIrefs. For example, the resource `exthings:companyCar` could be described as an instance of the class `ex:MotorVehicle` described in the schema of Example 24 by the RDF/XML shown in Example 25 :

Example 25: An Instance of `ex:MotorVehicle`

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex="http://example.org/schemas/vehicles#"
  xml:base="http://example.org/things">

   <ex:MotorVehicle rdf:ID="companyCar"/>

</rdf:RDF>
```

Note that the QName `ex:MotorVehicle`, when expanded using the namespace declaration `xmlns:ex="http://example.org/schemas/vehicles#"`, becomes the full URIref `http://example.org/schemas/vehicles#MotorVehicle`, which is the correct URIref for the `MotorVehicle` class as shown in Figure 18. The `xml:base` declaration `xml:base="http://example.org/things"` is provided to allow the `rdf:ID="companyCar"` to expand to the proper `exthings:companyCar` URIref (since a QName cannot be used as the value of the `rdf:ID` attribute).

5.2 Describing Properties

In addition to describing the specific *classes* of things they want to describe, user communities also need to be able to describe specific *properties* that characterize those classes of things (such as `rearSeatLegRoom` to describe a passenger vehicle). In RDF Schema, properties are described using the RDF class `rdf:Property`, and the RDF Schema properties `rdfs:domain`, `rdfs:range`, and `rdfs:subPropertyOf`.

All properties in RDF are described as instances of class `rdf:Property`. So a new property, such as `exterms:weightInKg`, is described by assigning the property a URIref, and describing that resource with an `rdf:type` property whose value is the resource `rdf:Property`, for example, by writing the RDF statement:

```
exterms:weightInKg   rdf:type   rdf:Property .
```

RDF Schema also provides vocabulary for describing how properties and classes are intended to be used together in RDF data. The most important information of this kind is supplied by using the RDF Schema properties `rdfs:range` and `rdfs:domain` to further describe application-specific properties.

The `rdfs:range` property is used to indicate that the values of a particular property are instances of a designated class. For example, if `example.org` wanted to indicate that the property `ex:author` had values that are instances of class `ex:Person`, it would write the RDF statements:

```
ex:Person   rdf:type     rdfs:Class .
ex:author   rdf:type     rdf:Property .
ex:author   rdfs:range   ex:Person .
```

These statements indicate that `ex:Person` is a class, `ex:author` is a property, and that RDF statements using the `ex:author` property have instances of `ex:Person` as objects.

A property, say `ex:hasMother`, can have zero, one, or more than one range property. If `ex:hasMother` has no range property, then nothing is said about the values of the `ex:hasMother` property. If `ex:hasMother` has one range property, say one specifying `ex:Person` as the range, this says that the values of the `ex:hasMother` property are instances of class `ex:Person`. If `ex:hasMother` has more than one range property, say one specifying `ex:Person` as its range, and another specifying `ex:Female` as its range, this says that the values of the `ex:hasMother` property are resources that are instances of *all* of the classes specified as the ranges, i.e., that any value of `ex:hasMother` is *both* a `ex:Female` *and* a `ex:Person`.

This last point may not be obvious. However, stating that the property `ex:hasMother` has the two ranges `ex:Female` and `ex:Person` involves making two separate statements:

```
ex:hasMother   rdfs:range   ex:Female .
ex:hasMother   rdfs:range   ex:Person .
```

For any given statement using this property, say:

```
exstaff:frank   ex:hasMother   exstaff:frances .
```

in order for *both* the rdfs:range statements to be correct, it must be the case that exstaff:frances is *both* an instance of ex:Female and of ex:Person.

The rdfs:range property can also be used to indicate that the value of a property is given by a typed literal, as discussed in Section 2.4. For example, if example.org wanted to indicate that the property ex:age had values from the XML Schema datatype xsd:integer, it would write the RDF statements:

```
ex:age   rdf:type     rdf:Property .
ex:age   rdfs:range   xsd:integer .
```

The datatype xsd:integer is identified by its URIref (the full URIref being http://www.w3.org/2001/XMLSchema#integer). This URIref can be used without explicitly stating in the schema that it identifies a datatype. However, it is often useful to explicitly state that a given URIref identifies a datatype. This can be done using the RDF Schema class rdfs:Datatype. To state that xsd:integer is a datatype, example.org would write the RDF statement:

```
xsd:integer   rdf:type   rdfs:Datatype .
```

This statement says that xsd:integer is the URIref of a datatype (which is assumed to conform to the requirements for RDF datatypes described in [RDF-CONCEPTS]). Such a statement does *not* constitute a *definition* of a datatype, e.g., in the sense that example.org is defining a new datatype. There is no way to define datatypes in RDF Schema. As noted in Section 2.4, datatypes are defined externally to RDF (and to RDF Schema), and *referred to* in RDF statements by their URIrefs. This statement simply serves to document the existence of the datatype, and indicate explicitly that it is being used in this schema.

The rdfs:domain property is used to indicate that a particular property applies to a designated class. For example, if example.org wanted to indicate that the property ex:author applies to instances of class ex:Book, it would write the RDF statements:

```
ex:Book     rdf:type      rdfs:Class .
ex:author   rdf:type      rdf:Property .
ex:author   rdfs:domain   ex:Book .
```

These statements indicate that ex:Book is a class, ex:author is a property, and that RDF statements using the ex:author property have instances of ex:Book as subjects.

A given property, say exterms:weight, may have zero, one, or more than one domain property. If exterms:weight has no domain property, then nothing is said about the resources that exterms:weight properties may be used with (any resource could have a exterms:weight property). If exterms:weight has one domain property, say one specifying ex:Book as the domain, this says that the exterms:weight property applies to instances of class ex:Book. If exterms:weight has more than one domain property, say one specifying ex:Book as the domain and another one specifying ex:MotorVehicle as the domain, this says that any resource that has a exterms:weight property is an instance of *all* of the classes specified as the domains, i.e., that any resource that has a exterms:weight property is both a ex:Book *and* a ex:MotorVehicle (illustrating the need for care in specifying domains and ranges).

As in the case of rdfs:range, this last point may not be obvious. However, stating that the property exterms:weight has the two domains ex:Book and ex:MotorVehicle involves making two separate statements:

```
exterms:weight   rdfs:domain   ex:Book .
exterms:weight   rdfs:domain   ex:MotorVehicle .
```

For any given statement using this property, say:

```
exthings:companyCar   exterms:weight   "2500"^^xsd:integer .
```

in order for *both* the `rdfs:domain` statements to be correct, it must be the case that `exthings:companyCar` is *both* an instance of `ex:Book` and of `ex:MotorVehicle`.

The use of these range and domain descriptions can be illustrated by extending the vehicle schema, adding two properties `ex:registeredTo` and `ex:rearSeatLegRoom`, a new class `ex:Person`, and explicitly describing the datatype `xsd:integer` as a datatype. The `ex:registeredTo` property applies to any `ex:MotorVehicle` and its value is a `ex:Person`. For the sake of this example, `ex:rearSeatLegRoom` applies only to instances of class `ex:PassengerVehicle`. The value is an `xsd:integer` giving the number of centimeters of rear seat legroom. These descriptions are shown in Example 26:

Example 26: Some Property Descriptions for the Vehicle Schema

```
<rdf:Property rdf:ID="registeredTo">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="rearSeatLegRoom">
  <rdfs:domain rdf:resource="#PassengerVehicle"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>

<rdfs:Class rdf:ID="Person"/>

<rdfs:Datatype rdf:about="&xsd;integer"/>
```

Note that an `<rdf:RDF>` element is not used in Example 26, because it is assumed this RDF/XML is being added to the vehicle schema described in Example 24. This same assumption also allows the use of relative URIrefs like `#MotorVehicle` to refer to other classes from that schema.

RDF Schema provides a way to specialize *properties* as well as classes. This specialization relationship between two properties is described using the predefined `rdfs:subPropertyOf` property. For example, if `ex:primaryDriver` and `ex:driver` are both properties, `example.org` could describe these properties, and the fact that `ex:primaryDriver` is a specialization of `ex:driver`, by writing the RDF statements:

```
ex:driver          rdf:type             rdf:Property .
ex:primaryDriver   rdf:type             rdf:Property .
ex:primaryDriver   rdfs:subPropertyOf   ex:driver .
```

The meaning of this `rdfs:subPropertyOf` relationship is that if an instance `exstaff:fred` is an `ex:primaryDriver` of the instance `ex:companyVan`, then RDF Schema defines `exstaff:fred` as also being an `ex:driver` of `ex:companyVan`. The RDF/XML describing these

properties (assuming again that it is being added to the vehicle schema described in Example 24) is shown in Example 27.

Example 27: More Properties for the Vehicle Schema

```
<rdf:Property rdf:ID="driver">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
</rdf:Property>

<rdf:Property rdf:ID="primaryDriver">
  <rdfs:subPropertyOf rdf:resource="#driver"/>
</rdf:Property>
```

A property may be a subproperty of zero, one or more properties. All RDF Schema `rdfs:range` and `rdfs:domain` properties that apply to an RDF property also apply to each of its subproperties. So, in the above example, RDF Schema defines `ex:primaryDriver` as also having an `rdfs:domain` of `ex:MotorVehicle`, because of its subproperty relationship to `ex:driver`.

Example 28 shows the RDF/XML for the full vehicle schema, containing all the descriptions given so far:

Example 28: The Full Vehicle Schema

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">

<rdfs:Class rdf:ID="MotorVehicle"/>

<rdfs:Class rdf:ID="PassengerVehicle">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Truck">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Van">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="MiniVan">
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Person"/>

<rdfs:Datatype rdf:about="&xsd;integer"/>

<rdf:Property rdf:ID="registeredTo">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
```

```
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="rearSeatLegRoom">
  <rdfs:domain rdf:resource="#PassengerVehicle"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>

<rdf:Property rdf:ID="driver">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
</rdf:Property>

<rdf:Property rdf:ID="primaryDriver">
  <rdfs:subPropertyOf rdf:resource="#driver"/>
</rdf:Property>

</rdf:RDF>
```

Having shown how to describe classes and properties using RDF Schema, instances using those classes and properties can now be illustrated. For example, Example 29 describes an instance of the `ex:PassengerVehicle` class described in Example 28, together with some hypothetical values for its properties.

Example 29: An Instance of `ex:PassengerVehicle`

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:ex="http://example.org/schemas/vehicles#"
            xml:base="http://example.org/things">

  <ex:PassengerVehicle rdf:ID="johnSmithsCar">
       <ex:registeredTo rdf:resource="http://www.example.org/
staffid/85740"/>
       <ex:rearSeatLegRoom
           rdf:datatype="&xsd;integer">127</ex:rearSeatLegRoom>
       <ex:primaryDriver rdf:resource="http://www.example.org/
staffid/85740"/>
  </ex:PassengerVehicle>
</rdf:RDF>
```

This example assumes that the instance is described in a separate document from the schema. Since the schema has an `xml:base` of `http://example.org/schemas/vehicles`, the namespace declaration `xmlns:ex="http://example.org/schemas/vehicles#"` is provided to allow QNames such as `ex:registeredTo` in the instance data to be properly expanded to the URIrefs of the classes and properties described in that schema. An `xml:base` declaration is also provided for this instance, to allow `rdf:ID="johnSmithsCar"` to expand to the proper URIref independently of the location of the actual document.

Note that an `ex:registeredTo` property can be used in describing this instance of `ex:PassengerVehicle`, because `ex:PassengerVehicle` is a subclass of `ex:MotorVehicle`. Note also that a typed literal is used for the value of the `ex:rearSetLegRoom` property in this instance, rather than a plain literal (i.e., rather than stating the value as `<ex:rearSeatLegRoom>127</ex:rearSeatLegRoom>`). Because the schema describes the range of this property as an `xsd:integer`, the value of the property should be a typed literal of

that datatype in order to match the range description (i.e., the range declaration does not automatically "assign" a datatype to a plain literal, and so a typed literal of the appropriate datatype must be explicitly provided). Additional information, either in the schema, or in additional instance data, could also be provided to explicitly specify the *units* of the `ex:rearSetLegRoom` property (centimeters), as discussed in Section 4.4.

5.3 Interpreting RDF Schema Declarations

As noted earlier, the RDF Schema type system is similar in some respects to the type systems of object-oriented programming languages such as Java. However, RDF differs from most programming language type systems in several important respects.

One important difference is that instead of describing a class as having a collection of specific properties, an RDF schema describes properties as applying to specific classes of resources, using *domain* and *range* properties. For example, a typical object-oriented programming language might define a class `Book` with an attribute called `author` having values of type `Person`. A corresponding RDF schema would describe a class `ex:Book`, and, in a separate description, a property `ex:author` having a domain of `ex:Book` and a range of `ex:Person`.

The difference between these approaches may seem to be only syntactic, but in fact there is an important difference. In the programming language class description, the attribute `author` is part of the description of class `Book`, and applies *only* to instances of class `Book`. Another class (say, `softwareModule`) might also have an attribute called `author`, but this would be considered a *different* attribute. In other words, the *scope* of an attribute description in most programming languages is restricted to the class or type in which it is defined. In RDF, on the other hand, property descriptions are, by default, *independent* of class definitions, and have, by default, *global* scope (although they may optionally be declared to apply only to certain classes using domain specifications).

As a result, an RDF schema could describe a property `exterms:weight` without a domain being specified. This property could then be used to describe instances of any class that might be considered to have a weight. One benefit of the RDF property-based approach is that it becomes easier to extend the use of property definitions to situations that might not have been anticipated in the original description. At the same time, this is a "benefit" which must be used with care, to insure that properties are not mis-applied in inappropriate situations.

Another result of the global scope of RDF property descriptions is that it is not possible in an RDF schema to define a specific property as having locally-different ranges depending on the class of the resource it is applied to. For example, in defining the property `ex:hasParent`, it would be desirable to be able to say that if the property is used to describe a resource of class `ex:Human`, then the range of the property is also a resource of class `ex:Human`, while if the property is used to describe a resource of class `ex:Tiger`, then the range of the property is also a resource of class `ex:Tiger`. This kind of definition is not possible in RDF Schema. Instead, any range defined for an RDF property applies to *all* uses of the property, and so ranges should be defined with care. However, while such locally-different ranges cannot be defined in RDF Schema, they can be defined in some of the richer schema languages discussed in Section 5.5.

Another important difference is that RDF Schema descriptions are not necessarily *prescriptive* in the way programming language type declarations typically are. For example, if a programming language declares a class `Book` with an `author` attribute having values of type `Person`, this is usually interpreted as a group of *constraints*. The language will not allow the creation of an instance of `Book` without an `author` attribute, and it will not allow an instance of `Book` with an `author` attribute that does not have a `Person` as its value. Moreover, if `author` is the *only* attribute defined for class `Book`, the language will not allow an instance of `Book` with some other

attribute.

RDF Schema, on the other hand, provides schema information as additional *descriptions* of resources, but does not prescribe how these descriptions should be used by an application. For example, suppose an RDF schema states that an `ex:author` property has an `rdfs:range` of class `ex:Person`. This is simply an RDF statement that RDF statements containing `ex:author` properties have instances of `ex:Person` as objects.

This schema-supplied information might be used in different ways. One application might interpret this statement as specifying part of a template for RDF data it is creating, and use it to ensure that any `ex:author` property has a value of the indicated (`ex:Person`) class. That is, this application interprets the schema description as a *constraint* in the same way that a programming language might. However, another application might interpret this statement as providing additional information about data it is receiving, information which may not be provided explicitly in the original data. For example, this second application might receive some RDF data that includes an `ex:author` property whose value is a resource of unspecified class, and use this schema-provided statement to conclude that the resource must be an instance of class `ex:Person`. A third application might receive some RDF data that includes an `ex:author` property whose value is a resource of class `ex:Corporation`, and use this schema information as the basis of a warning that "there may be an inconsistency here, but on the other hand there may not be". Somewhere else there may be a declaration that resolves the apparent inconsistency (e.g., a declaration to the effect that "a Corporation is a (legal) Person").

Moreover, depending on how the application interprets the property descriptions, a description of an instance might be considered valid either *without* some of the schema-specified properties (e.g., there might be an instance of `ex:Book` without an `ex:author` property, even if `ex:author` is described as having a domain of `ex:Book`), or with *additional* properties (there might be an instance of `ex:Book` with an `ex:technicalEditor` property, even though the schema describing class `ex:Book` does not describe such a property).

In other words, statements in an RDF schema are always *descriptions*. They may also be *prescriptive* (introduce constraints), but only if the application interpreting those statements wants to treat them that way. All RDF Schema does is provide a way of stating this additional information. Whether this information conflicts with explicitly specified instance data is up to the application to determine and act upon.

5.4 Other Schema Information

RDF Schema provides a number of other built-in properties, which can be used to provide documentation and other information about an RDF schema or about instances. For example the `rdfs:comment` property can be used to provide a human-readable description of a resource. The `rdfs:label` property can be used to provide a more human-readable version of a resource's name. The `rdfs:seeAlso` property can be used to indicate a resource that might provide additional information about the subject resource. The `rdfs:isDefinedBy` property is a subproperty of `rdfs:seeAlso`, and can be used to indicate a resource that (in a sense not specified by RDF; e.g., the resource may not be an RDF schema) "defines" the subject resource. RDF Vocabulary Description Language 1.0: RDF Schema [RDF-VOCABULARY] should be consulted for further discussion of these properties.

As with a number of the built-in RDF properties such as `rdf:value`, the uses described for these RDF Schema properties are only their *intended* uses. [RDF-SEMANTICS] defines no special meanings for these properties, and RDF Schema does not define any constraints based on these intended uses. For example, there is no constraint specified that the object of a `rdfs:seeAlso` property *must* provide additional information about the subject of the statement in

which it appears.

5.5 Richer Schema Languages

RDF Schema provides basic capabilities for describing RDF vocabularies, but additional capabilities are also possible, and can be useful. These capabilities may be provided through further development of RDF Schema, or in other languages based on RDF. Other richer schema capabilities that have been identified as useful (but that are not provided by RDF Schema) include:

- *cardinality constraints* on properties, e.g., that a Person has *exactly one* biological father.
- specifying that a given property (such as `ex:hasAncestor`) is *transitive*, e.g., that if A `ex:hasAncestor` B, and B `ex:hasAncestor` C, then A `ex:hasAncestor` C.
- specifying that a given property is a unique identifier (or *key*) for instances of a particular class.
- specifying that two different classes (having different URIrefs) actually represent the same class.
- specifying that two different instances (having different URIrefs) actually represent the same individual.
- specifying constraints on the range or cardinality of a property that depend on the class of resource to which a property is applied, e.g., being able to say that for a soccer team the `ex:hasPlayers` property has 11 values, while for a basketball team the same property should have only 5 values.
- the ability to describe new classes in terms of combinations (e.g., unions and intersections) of other classes, or to say that two classes are disjoint (i.e., that no resource is an instance of both classes).

The additional capabilities mentioned above, in addition to others, are the targets of *ontology* languages such as DAML+OIL [DAML+OIL] and OWL [OWL]. Both these languages are based on RDF and RDF Schema (and both currently provide all the additional capabilities mentioned above). The intent of such languages is to provide additional machine-processable *semantics* for resources, that is, to make the machine representations of resources more closely resemble their intended real world counterparts. While such capabilities are not necessarily needed to build useful applications using RDF (see Section 6 for a description of a number of existing RDF applications), the development of such languages is a very active subject of work as part of the development of the Semantic Web.

6. Some RDF Applications: RDF in the Field

The previous sections have described the general capabilities of RDF and RDF Schema. While examples were used in those sections to illustrate those capabilities, and some of those examples may have suggested potential RDF applications, those sections did not actually discuss any *real* applications. This section will describe some actual deployed RDF applications, showing how RDF supports various real-world requirements to represent and manipulate information about a wide variety of things.

6.1 Dublin Core Metadata Initiative

Metadata is *data about data*. Specifically, the term refers to data used to identify, describe, or locate information resources, whether these resources are physical or electronic. While structured metadata processed by computers is relatively new, the basic concept of metadata has been used for many years in helping manage and use large collections of information. Library card catalogs are a familiar example of such metadata.

The Dublin Core is a set of "elements" (properties) for describing documents (and hence, for recording metadata). The element set was originally developed at the March 1995 Metadata Workshop in Dublin, Ohio. The Dublin Core has subsequently been modified on the basis of later Dublin Core Metadata workshops, and is currently maintained by the Dublin Core Metadata Initiative. The goal of the Dublin Core is to provide a minimal set of descriptive elements that facilitate the description and the automated indexing of document-like networked objects, in a manner similar to a library card catalog. The Dublin Core metadata set is intended to be suitable for use by resource discovery tools on the Internet, such as the "Webcrawlers" employed by popular World Wide Web search engines. In addition, the Dublin Core is meant to be sufficiently simple to be understood and used by the wide range of authors and casual publishers who contribute information to the Internet. Dublin Core elements have become widely used in documenting Internet resources (the Dublin Core `creator` element has already been used in earlier examples). The current elements of the Dublin Core are defined in the Dublin Core Metadata Element Set, Version 1.1: Reference Description [DC], and contain definitions for the following properties:

- **Title**: A name given to the resource.
- **Creator**: An entity primarily responsible for making the content of the resource.
- **Subject**: The topic of the content of the resource.
- **Description**: An account of the content of the resource.
- **Publisher**: An entity responsible for making the resource available
- **Contributor**: An entity responsible for making contributions to the content of the resource.
- **Date**: A date associated with an event in the life cycle of the resource.
- **Type**: The nature or genre of the content of the resource.
- **Format**: The physical or digital manifestation of the resource.
- **Identifier**: An unambiguous reference to the resource within a given context.
- **Source**: A reference to a resource from which the present resource is derived.
- **Language**: A language of the intellectual content of the resource.
- **Relation**: A reference to a related resource.
- **Coverage**: The extent or scope of the content of the resource.
- **Rights**: Information about rights held in and over the resource.

Information using the Dublin Core elements may be represented in any suitable language (e.g., in HTML `meta` elements). However, RDF is an ideal representation for Dublin Core information. The examples below represent the simple description of a set of resources in RDF using the Dublin Core vocabulary. Note that the specific Dublin Core RDF vocabulary shown here is not intended to be authoritative. The Dublin Core Reference Description [DC] is the authoritative reference.

The first example, Example 30, describes a Web site home page using Dublin Core properties:

Example 30: A Web Page Described using Dublin Core Properties

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="http://www.dlib.org">
      <dc:title>D-Lib Program - Research in Digital Libraries</dc:
title>
      <dc:description>The D-Lib program supports the community of
people
       with research interests in digital libraries and electronic
       publishing.</dc:description>
      <dc:publisher>Corporation For National Research Initiatives</dc:
publisher>
```

```
      <dc:date>1995-01-07</dc:date>
      <dc:subject>
        <rdf:Bag>
          <rdf:li>Research; statistical methods</rdf:li>
          <rdf:li>Education, research, related topics</rdf:li>
          <rdf:li>Library use Studies</rdf:li>
        </rdf:Bag>
      </dc:subject>
      <dc:type>World Wide Web Home Page</dc:type>
      <dc:format>text/html</dc:format>
      <dc:language>en</dc:language>
    </rdf:Description>
</rdf:RDF>
```

Note that both RDF and the Dublin Core define an (XML) element called "Description" (although the Dublin Core element name is written in lowercase). Even if the initial letter were identically uppercase, the XML namespace mechanism enables these two elements to be distinguished (one is `rdf:Description`, and the other is `dc:description`). Also, as a matter of interest, accessing http://purl.org/dc/elements/1.1/ (the namespace URI used to identify the Dublin Core vocabulary in this example) in a Web browser (as of the current writing) will retrieve an RDF Schema declaration for [DC].

The second example, Example 31, describes a published magazine:

Example 31: Describing A Magazine Using Dublin Core

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcterms="http://purl.org/dc/terms/">
    <rdf:Description rdf:about="http://www.dlib.org/dlib/
may98/05contents.html">
      <dc:title>DLIB Magazine - The Magazine for Digital Library
Research
        - May 1998</dc:title>
      <dc:description>D-LIB magazine is a monthly compilation of
       contributed stories, commentary, and briefings.</dc:description>
      <dc:contributor>Amy Friedlander</dc:contributor>
      <dc:publisher>Corporation for National Research Initiatives</dc:
publisher>
      <dc:date>1998-01-05</dc:date>
      <dc:type>electronic journal</dc:type>
      <dc:subject>
        <rdf:Bag>
          <rdf:li>library use studies</rdf:li>
          <rdf:li>magazines and newspapers</rdf:li>
        </rdf:Bag>
      </dc:subject>
      <dc:format>text/html</dc:format>
      <dc:identifier rdf:resource="urn:issn:1082-9873"/>
      <dcterms:isPartOf rdf:resource="http://www.dlib.org"/>
    </rdf:Description>
 </rdf:RDF>
```

Example 31 uses (in the third line from the bottom) the Dublin Core *qualifier* `isPartOf` (from a

separate vocabulary) to indicate that this magazine is "part of" the previously-described Web site.

The third example, Example 32, describes a specific article in the magazine described in Example 31.

Example 32: Describing a Magazine Article

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcterms="http://purl.org/dc/terms/">
    <rdf:Description rdf:about="http://www.dlib.org/dlib/may98/
miller/05miller.html">
      <dc:title>An Introduction to the Resource Description Framework</
dc:title>
      <dc:creator>Eric J. Miller</dc:creator>
      <dc:description>The Resource Description Framework (RDF) is an
       infrastructure that enables the encoding, exchange and reuse of
       structured metadata. rdf is an application of xml that imposes
needed
       structural constraints to provide unambiguous methods of
expressing
       semantics. rdf additionally provides a means for publishing both
       human-readable and machine-processable vocabularies designed to
       encourage the reuse and extension of metadata semantics among
       disparate information communities. the structural constraints
rdf
       imposes to support the consistent encoding and exchange of
       standardized metadata provides for the interchangeability of
separate
       packages of metadata defined by different resource description
       communities. </dc:description>
      <dc:publisher>Corporation for National Research Initiatives</dc:
publisher>
      <dc:subject>
        <rdf:Bag>
          <rdf:li>machine-readable catalog record formats</rdf:li>
          <rdf:li>applications of computer file organization and
           access methods</rdf:li>
        </rdf:Bag>
      </dc:subject>
      <dc:rights>Copyright © 1998 Eric Miller</dc:rights>
      <dc:type>Electronic Document</dc:type>
      <dc:format>text/html</dc:format>
      <dc:language>en</dc:language>
      <dcterms:isPartOf rdf:resource="http://www.dlib.org/dlib/
may98/05contents.html"/>
    </rdf:Description>
</rdf:RDF>
```

Example 32 also uses the qualifier `isPartOf`, this time to indicate that this article is "part of" the previously-described magazine.

Computer languages and file formats do not always make explicit provision for embedding metadata with the data it describes. In many cases, the metadata has to be specified as a

separate resource and explicitly linked to the data (this has been done for the RDF metadata that describes the Primer; there is an explicit link to this metadata at the end of the Primer). However, applications and languages are increasingly making explicit provision for embedding metadata directly with the data. For example, the W3C's Scalable Vector Graphics language [SVG] (another XML-based language) provides an explicit `metadata` element for recording metadata along with other SVG data. Any XML-based metadata language can be used inside this element. [SVG] includes the example shown in Example 33 of how to embed metadata describing an SVG document in the SVG document itself. The example uses the Dublin Core vocabulary, and RDF/XML for recording the metadata.

Example 33: Including Metadata in an SVG Document

```
<?xml version="1.0"?>
<svg width="4in" height="3in" version="1.1"
    xmlns = 'http://www.w3.org/2000/svg'>
    <desc xmlns:myfoo="http://example.org/myfoo">
      <myfoo:title>This is a financial report</myfoo:title>
      <myfoo:descr>The global description uses markup from the
        <myfoo:emph>myfoo</myfoo:emph> namespace.</myfoo:descr>
      <myfoo:scene><myfoo:what>widget $growth</myfoo:what>
      <myfoo:contains>$three $graph-bar</myfoo:contains>
        <myfoo:when>1998 $through 2000</myfoo:when> </myfoo:scene>
   </desc>
    <metadata>
      <rdf:RDF
           xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
           xmlns:dc = "http://purl.org/dc/elements/1.1/" >
        <rdf:Description rdf:about="http://example.org/myfoo"
             dc:title="MyFoo Financial Report"
             dc:description="$three $bar $thousands $dollars $from
1998 $through 2000"
             dc:publisher="Example Organization"
             dc:date="2000-04-11"
             dc:format="image/svg+xml"
             dc:language="en" >
          <dc:creator>
            <rdf:Bag>
              <rdf:li>Irving Bird</rdf:li>
              <rdf:li>Mary Lambert</rdf:li>
            </rdf:Bag>
          </dc:creator>
        </rdf:Description>
      </rdf:RDF>
    </metadata>
</svg>
```

Adobe's Extensible Metadata Platform (XMP) is another example of technology that allows metadata about a file to be embedded into the file itself. XMP uses RDF/XML as the basis of its metadata representation. A number of Adobe products already support XMP.

6.2 PRISM

PRISM: Publishing Requirements for Industry Standard Metadata [PRISM] is a metadata specification developed in the publishing industry. Magazine publishers and their vendors formed

the PRISM Working Group to identify the industry's needs for metadata and define a specification to meet them. Publishers want to use existing content in many ways in order to get a greater return on the investment made in creating it. Converting magazine articles to HTML for posting on the Web is one example. Licensing it to aggregators like LexisNexis is another. All of these are "first uses" of the content; typically they all go live at the time the magazine hits the stands. The publishers also want their content to be "evergreen". It might be used in new issues, such as in a retrospective article. It could be used by other divisions in the company, such as in a book compiled from the magazine's photos, recipes, etc. Another use is to license it to outsiders, such as in a reprint of a product review, or in a retrospective produced by a different publisher. This overall goal requires a metadata approach that emphasizes *discovery*, *rights tracking*, and *end-to-end metadata*.

Discovery: Discovery is a general term for finding content which encompasses searching, browsing, content routing, and other techniques. Discussions of discovery frequently center on a consumer searching a public Web site. However, discovering content is much broader than that. The audience may consist of consumers, or it may consist of internal users such as researchers, designers, photo editors, licensing agents, etc. To assist discovery, PRISM provides properties to describe the topics, formats, genre, origin, and contexts of a resource. It also provides means for categorizing resources using multiple subject description taxonomies.

Rights Tracking: Magazines frequently contain material licensed from others. Photos from a stock photo agency are the most common type of licensed material, but articles, sidebars, and all other types of content may be licensed. Simply knowing if content was licensed for one-time use, requires royalty payments, or is wholly-owned by the publisher is a struggle. PRISM provides elements for basic tracking of such rights. A separate vocabulary defined in the PRISM specification supports description of places, times, and industries where content may or may not be used.

End-to-end metadata: Most published content already has metadata created for it. Unfortunately, when content moves between systems, the metadata is frequently discarded, only to be re-created later in the production process at considerable expense. PRISM aims to reduce this problem by providing a specification that can be used in multiple stages in the content production pipeline. An important feature of the PRISM specification is its use of other existing specifications. Rather than create an entirely new thing, the group decided to use existing specifications as much as possible, and only define new things where needed. For this reason, the PRISM specification uses XML, RDF, Dublin Core, and well as various ISO formats and vocabularies.

A PRISM description may be as simple as a few Dublin Core properties with plain literal values. Example 34 describes a photograph, giving basic information on its title, photographer, format, etc.

Example 34: A PRISM Description of a Photograph

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xml:lang="en-US">

 <rdf:Description rdf:about="http://travel.example.com/2000/08/Corfu.
jpg">
  <dc:title>Walking on the Beach in Corfu</dc:title>
  <dc:description>Photograph taken at 6:00 am on Corfu with two models
  </dc:description>
  <dc:creator>John Peterson</dc:creator>
  <dc:contributor>Sally Smith, lighting</dc:contributor>
```

```
  <dc:format>image/jpeg</dc:format>
 </rdf:Description>
</rdf:RDF>
```

PRISM also augments the Dublin Core to allow more detailed descriptions. The augmentations are defined as three new vocabularies, generally cited using the prefixes `prism:`, `pcv:`, and `prl:`.

`prism:` This prefix refers to the main PRISM vocabulary, whose terms use the URI prefix `http://prismstandard.org/namespaces/basic/1.0/`. Most of the properties in this vocabulary are more specific versions of properties from the Dublin Core. For example, more specific versions of `dc:date` are provided by properties like `prism:publicationTime`, `prism:releaseTime`, `prism:expirationTime`, etc.

`pcv:` This prefix refers to the PRISM Controlled Vocabulary (pcv) vocabulary, whose terms use the URI prefix `http://prismstandard.org/namespaces/pcv/1.0/`. Currently, common practice for describing the subject(s) of an article is by supplying descriptive keywords. Unfortunately, simple keywords do not make a great difference in retrieval performance, due to the fact that different people will use different keywords [BATES96]. Best practice is to code the articles with subject terms from a "controlled vocabulary". The vocabulary should provide as many synonyms as possible for its terms in the vocabulary. This way the controlled terms provide a meeting ground for the keywords supplied by the searcher and the indexer. The pcv vocabulary provides properties for specifying terms in a vocabulary, the relations between terms, and alternate names for the terms.

`prl:` This prefix refers to the PRISM Rights Language vocabulary, whose terms use the URI prefix `http://prismstandard.org/namespaces/prl/1.0/`. Digital Rights Management is an area undergoing considerable upheaval. There are a number of proposals for rights management languages, but none are clearly favored throughout the industry. Because there was no clear choice to recommend, the PRISM Rights Language (PRL) was defined as an interim measure. It provides properties which let people say if an item can or cannot be "used", depending on conditions of time, geography, and industry. This is believed to be an 80/20 trade-off which will help publishers begin to save money when tracking rights. It is not intended to be a general rights language, or allow publishers to automatically enforce limits on consumer uses of the content.

PRISM uses RDF because of its abilities for dealing with descriptions of varying complexity. Currently, a great deal of metadata uses simple character string (plain literal) values, such as:

```
<dc:coverage>Greece</dc:coverage>
```

Over time the developers of PRISM expect uses of the PRISM specification to become more sophisticated, moving from simple literal values to more structured values. In fact, that range of values is a situation being faced now. Some publishers already use sophisticated controlled vocabularies, others are barely using manually-supplied keywords. To illustrate this, some examples of the different kinds of values that can be given for the `dc:coverage` property are:

```
<dc:coverage>Greece</dc:coverage>

<dc:coverage rdf:resource="http://prismstandard.org/vocabs/ISO-3166/
GR"/>
```

(i.e., using either a plain literal or a URIref to identify the country) and

```
<dc:coverage>
  <pcv:Descriptor rdf:about="http://prismstandard.org/vocabs/ISO-3166/
GR">
    <pcv:label xml:lang="en">Greece</pcv:label>
    <pcv:label xml:lang="fr">Grèce</pcv:label>
  </pcv:Descriptor>
</dc:coverage>
```

(using a structured value to provide both a URIref and names in various languages).

Note also that there are properties whose meanings are similar, or subsets of other properties. For example, the geographic subject of a resource could be given with

```
<prism:subject>Greece</prism:subject>
<dc:coverage>Greece</dc:coverage>
```

or

```
<prism:location>Greece</prism:location>
```

Any of those properties might use the simple literal value, or a more complex structured value. Such a range of possibilities cannot be adequately described by DTDs, or even by the newer XML Schemas. While there is a wide range of syntactic variations to deal with, RDF's graph model has a simple structure - a set of triples. Dealing with the metadata in the triples domain makes it much easier for older software to accommodate content with new extensions.

This section closes with two final examples. Example 35 says that the image (`.../Corfu.jpg`) cannot be used (`#none`) in the tobacco industry (code 21 in SIC, the Standard Industrial Classifications).

Example 35: A PRISM Description of an Image

```
<rdf:RDF xmlns:prism="http://prismstandard.org/namespaces/basic/1.0/"
         xmlns:prl="http://prismstandard.org/namespaces/prl/1.0/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">

 <rdf:Description rdf:about="http://travel.example.com/2000/08/Corfu.
jpg">
  <dc:rights rdf:parseType="Resource"
         xml:base="http://prismstandard.org/vocabularies/1.0/usage.
xml">
     <prl:usage rdf:resource="#none"/>
     <prl:industry rdf:resource="http://prismstandard.org/vocabs/
SIC/21"/>
  </dc:rights>
 </rdf:Description>
</rdf:RDF>
```

Example 36 says that the photographer for the Corfu image was employee 3845, better known as John Peterson. It also says that the geographic coverage of the photo is Greece. It does so by providing, not just a code from a controlled vocabulary, but a cached version of the information

for that term in the vocabulary.

Example 36: Additional Information about the Image from Example 35

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:pcv="http://prismstandard.org/namespaces/pcv/1.0/"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xml:base="http://travel.example.com/">

  <rdf:Description rdf:about="/2000/08/Corfu.jpg">
    <dc:identifier rdf:resource="/content/2357845" />
    <dc:creator>
      <pcv:Descriptor rdf:about="/emp3845">
        <pcv:label>John Peterson</pcv:label>
      </pcv:Descriptor>
    </dc:creator>
    <dc:coverage>
      <pcv:Descriptor
          rdf:about="http://prismstandard.org/vocabs/ISO-3166/GR">
        <pcv:label xml:lang="en">Greece</pcv:label>
        <pcv:label xml:lang="fr">Grece</pcv:label>
      </pcv:Descriptor>
    </dc:coverage>
  </rdf:Description>
</rdf:RDF>
```

6.3 XPackage

Many situations involve the need to maintain information about structured groupings of resources and their associations that are, or may be, used as a unit. The XML Package (XPackage) specification [XPACKAGE] provides a framework for defining such groupings, called *packages*. XPackage specifies a framework for describing the resources included in such packages, the properties of those resources, their method of inclusion, and their relationships with each other. XPackage applications include specifying the style sheets used by a document, declaring the images shared by multiple documents, indicating the author and other metadata of a document, describing how namespaces are used by XML resources, and providing a manifest for bundling resources into a single archive file.

The XPackage framework is based upon XML, RDF, and the XML Linking Language [XLINK], and provides multiple RDF vocabularies: one for general packaging descriptions, and several other vocabularies for providing supplemental resource information useful to package processors.

One application of XPackage is the description of XHTML documents and their supporting resources. An XHTML document retrieved from a Web site may rely on other resources such as style sheets and image files that also need to be retrieved. However, the identities of these supporting resources may not be obvious without processing the entire document. Other information about the document, such as the name of its author, may also not be available without processing the document. XPackage allows such descriptive information to be stored in a standard way in a package description document containing RDF. The outer elements of a package description document describing such an XHTML document might look like Example 37 (with namespace declarations removed for simplicity):

Example 37: Outer Elements of an XPackage Package Description Document

```
<?xml version="1.0"?>
```

```
<xpackage:description>
  <rdf:RDF>

    (description of individual resources go here)

  </rdf:RDF>
</xpackage:description>
```

Resources (such as the XHTML document, style sheets, and images) are described within this package description document using standard RDF/XML syntax. Each resource description element may include RDF properties from various vocabularies (XPackage uses the term "ontology" for what RDF calls a "vocabulary"). Besides the main packaging vocabulary, XPackage itself specifies several supplemental vocabularies, including:

- a vocabulary (using prefix `file:`) for describing files (with properties such as `file:size`)
- a vocabulary (using prefix `mime:`) for providing MIME information (with properties such as `mime:contentType`)
- a vocabulary (using prefix `unicode:`) for providing character usage information (with properties such as `unicode:script`)
- a vocabulary (using prefix `x:`) for describing XML-based resources (with properties such as `x:namespace` and `x:style`)

In Example 38, the document's MIME content type ("application/xhtml+xml") is defined using a standard XPackage property from the XPackage MIME vocabulary, `mime:contentType`. Another property, the document's author (in this case, "Garret Wilson"), is described using a property from the Dublin Core vocabulary, defined outside of XPackage, resulting in a `dc:creator` property.

Example 38: A Description of an XHTML Document

```
<?xml version="1.0"?>
<xpackage:description
           xmlns:xpackage="http://xpackage.org/namespaces/2003/
xpackage#"
           xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
           xmlns:dc="http://purl.org/dc/elements/1.1/"
           xmlns:mime="http://xpackage.org/namespaces/2003/mime#"
           xmlns:x="http://xpackage.org/namespaces/2003/xml#"
           xmlns:xlink="http://www.w3.org/1999/xlink">
    <rdf:RDF>

    <!--doc.html-->
    <rdf:Description rdf:about="urn:example:xhtmldocument-doc">
      <rdfs:comment>The XHTML document.</rdfs:comment>
      <xpackage:location xlink:href="doc.html"/>
      <mime:contentType>application/xhtml+xml</mime:contentType>
      <x:namespace rdf:resource="http://www.w3.org/1999/xhtml"/>
      <x:style rdf:resource="urn:example:xhtmldocument-stylesheet"/>
      <dc:creator>Garret Wilson</dc:creator>
      <xpackage:manifest rdf:parseType="Collection">
         <rdf:Description rdf:about="urn:example:xhtmldocument-
stylesheet"/>
         <rdf:Description rdf:about="urn:example:xhtmldocument-image"/>
      </xpackage:manifest>
```

```
    </rdf:Description>

    </rdf:RDF>
</xpackage:description>
```

The `xpackage:manifest` property indicates that both the style sheet and image resources are necessary for processing; those resources are described separately within the package description document. The example style sheet resource description in Example 39 lists its location within the package ("stylesheet.css") using the general XPackage vocabulary `xpackage:location` property (which is compatible with XLink), and shows through use of the XPackage MIME vocabulary `mime:contentType` property that it is a CSS style sheet ("text/css").

Example 39: A Style Sheet Resource Description

```
<?xml version="1.0"?>
<xpackage:description
           xmlns:xpackage="http://xpackage.org/namespaces/2003/
xpackage#"
           xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
           xmlns:dc="http://purl.org/dc/elements/1.1/"
           xmlns:mime="http://xpackage.org/namespaces/2003/mime#"
           xmlns:x="http://xpackage.org/namespaces/2003/xml#"
           xmlns:xlink="http://www.w3.org/1999/xlink">
    <rdf:RDF>

    <!--stylesheet.css-->
    <rdf:Description rdf:about="urn:example:xhtmldocument-css">
      <rdfs:comment>The document style sheet.</rdfs:comment>
      <xpackage:location xlink:href="stylesheet.css"/>
      <mime:contentType>text/css</mime:contentType>
    </rdf:Description>

    </rdf:RDF>
</xpackage:description>
```

The full version of this example may be found in [XPACKAGE].

6.4 RSS 1.0: RDF Site Summary

People sometimes need to access a wide variety of information on the Web on a day-to-day basis, such as schedules, to-do lists, news headlines, search results, "What's New", etc. As the sources and diversity of the information on the Web increases, it becomes increasingly difficult to manage this information and integrate it into a coherent whole. RSS 1.0 ("RDF Site Summary") is an RDF vocabulary that provides a lightweight, yet powerful way of describing information for timely, large-scale distribution and reuse. RSS 1.0 is also perhaps the most widely deployed RDF application on the Web.

To give a simple example, the W3C home page is a primary point of contact with the public and serves in part to disseminate information about the deliverables of the Consortium. An example of the W3C home page as of a certain date is shown in Figure 19. The center column of news items changes frequently. To support the timely dissemination of this information, the W3C Team has implemented an RDF Site Summary (RSS 1.0) news feed that makes the content in the

center column available to others to reuse as they will. News syndication sites may merge the headlines into a summary of the day's latest news, others may display the headlines as links as a service to their readers, and, increasingly, individuals may subscribe to this feed with a desktop application. These desktop *RSS readers* allow their users to keep track of potentially hundreds of sites, without having to visit each one in their browser.

Figure 19: The W3C Home Page

Numerous sites all over the Web provide RSS 1.0 feeds. Example 40 is an example of the W3C feed (from a different date):

Example 40: An Example of the W3C RSS 1.0 Feed

```
<?xml version="1.0" encoding="utf-8"?>

<rdf:RDF xmlns="http://purl.org/rss/1.0/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <channel rdf:about="http://www.w3.org/2000/08/w3c-synd/home.rss">
    <title>The World Wide Web Consortium</title>
    <description>Leading the Web to its Full Potential...</description>
    <link>http://www.w3.org/</link>

    <dc:date>2002-10-28T08:07:21Z</dc:date>

    <items>
        <rdf:Seq>
            <rdf:li rdf:resource="http://www.w3.org/News/2002#item164"/
>
            <rdf:li rdf:resource="http://www.w3.org/News/2002#item168"/
>
            <rdf:li rdf:resource="http://www.w3.org/News/2002#item167"/
>
        </rdf:Seq>
    </items>
```

```
  </channel>

  <item rdf:about="http://www.w3.org/News/2002#item164">
    <title>User Agent Accessibility Guidelines Become a W3C
       Proposed Recommendation</title>
    <description>17 October 2002: W3C is pleased to announce the
       advancement of User Agent Accessibility Guidelines 1.0 to
       Proposed Recommendation. Comments are welcome through 14
November.
       Written for developers of user agents, the guidelines lower
       barriers to Web accessibility for people with disabilities
       (visual, hearing, physical, cognitive, and neurological).
       The companion Techniques Working Draft is updated. Read about
       the Web Accessibility Initiative. (News archive)</description>
    <link>http://www.w3.org/News/2002#item164</link>
    <dc:date>2002-10-17</dc:date>
  </item>

  <item rdf:about="http://www.w3.org/News/2002#item168">
    <title>Working Draft of Authoring Challenges for Device
       Independence Published</title>
    <description>25 October 2002: The Device Independence
       Working Group has released the first public Working Draft of
       Authoring Challenges for Device Independence. The draft
describes
       the considerations that Web authors face in supporting access
to
       their sites from a variety of different devices. It is written
       for authors, language developers, device experts and developers
       of Web applications and authoring systems. Read about the
Device
       Independence Activity (News archive)</description>
    <link>http://www.w3.org/News/2002#item168</link>
    <dc:date>2002-10-25</dc:date>
  </item>

  <item rdf:about="http://www.w3.org/News/2002#item167">
    <title>CSS3 Last Call Working Drafts Published</title>
    <description>24 October 2002: The CSS Working Group has
       released two Last Call Working Drafts and welcomes comments
       on them through 27 November. CSS3 module: text is a set of
       text formatting properties and addresses international
contexts.
       CSS3 module: Ruby is properties for ruby, a short run of text
       alongside base text typically used in East Asia. CSS3 module:
       The box model for the layout of textual documents in visual
       media is also updated. Cascading Style Sheets (CSS) is a
       language used to render structured documents like HTML and
       XML on screen, on paper, and in speech. Visit the CSS home
       page. (News archive)</description>
    <link>http://www.w3.org/News/2002#item167</link>
    <dc:date>2002-10-24</dc:date>
  </item>

</rdf:RDF>
```

As Example 40 shows, the format is designed for content that can be packaged into easily distinguishable sections. News sites, Web logs, sports scores, stock quotes, and the like are all use-cases for RSS 1.0.

The RSS feed can be requested by any application able to "speak" HTTP. More recently, however, RSS 1.0 applications are splitting into three different categories:

- On-line aggregators - Sites such as Meerkat and NewsIsFree, shown side-by-side in Figure 20 (each mirroring W3C's column of news). These gather feeds from thousands of sources, separate each of the `<item>`s out, and add them together again into one large group. The whole group is then made searchable. In this way, one can search for the latest news on, for example, "Java" from perhaps thousands of sites, without having to search them all.
- Desktop Readers - Utilities such as Amphetadesk and NetNewsWire Lite allow their users to subscribe to hundreds of feeds from their desktop. Readers customarily refresh each feed once an hour, allowing users to stay up to date.
- Scripts - RSS's original purpose was to allow Webmasters to include the content of another's site within their own. RSS 1.0 is still used in this way, with many sites (Slashdot for example) incorporating RSS feeds on their front page.

Figure 20: MeerKat and NewsIsFree

RSS 1.0 is extensible by design. By importing additional RDF vocabularies (or *modules* as they are known within the RSS development community), the RSS 1.0 author can provide large amounts of metadata and handling instructions to the recipient of the file. Modules can, as with more general RDF vocabularies, be written by anyone. Currently there are 3 official modules and 19 proposed modules readily recognized by the community at large. These modules range from the complete Dublin Core module to more specialized RSS-centric modules such as the Aggregation module.

Care should be taken when discussing "RSS" in the scope of RDF. There are currently two RSS specification strands. One strand (RSS 0.91,0.92,0.93,0.94 and 2.0) does not use RDF. The other strand (RSS 0.9 and 1.0) does.

6.5 CIM/XML

Electric utilities use power system models for a number of different purposes. For example, simulations of power systems are necessary for planning and security analysis. Power system models are also used in actual operations, e.g., by the Energy Management Systems (EMS) used in energy control centers. An operational power system model can consist of thousands of classes of information. In addition to using these models in-house, utilities need to exchange

system modeling information, both in planning, and for operational purposes, e.g., for coordinating transmission and ensuring reliable operations. However, individual utilities use different software for these purposes, and as a result the system models are stored in different formats, making the exchange of these models difficult.

In order to support the exchange of power system models, utilities needed to agree on common definitions of power system entities and relationships. To support this, the Electric Power Research Institute (EPRI) a non-profit energy research consortium, developed a Common Information Model (CIM) [CIM]. The CIM specifies common semantics for power system resources, their attributes, and relationships. In addition, to further support the ability to electronically exchange CIM models, the power industry has developed CIM/XML, a language for expressing CIM models in XML. CIM/XML is an RDF application, using RDF and RDF Schema to organize its XML structures. The North American Electric Reliability Council (NERC) (an industry-supported organization formed to promote the reliability of electricity delivery in North America) has adopted CIM/XML as the standard for exchanging models between power transmission system operators. The CIM/XML format is also going through an IEC international standardization process. An excellent discussion of CIM/XML can be found in [DWZ01]. [NB: This power industry CIM should not be confused with the CIM developed by the Distributed Management Task Force for representing management information for distributed software, network, and enterprise environments. The DMTF CIM also has an XML representation, but does not currently use RDF, although independent research is underway in that direction.]

The CIM can represent all of the major objects of an electric utility as object classes and attributes, as well as their relationships. CIM uses these object classes and attributes to support the integration of independently developed applications between vendor specific EMS systems, or between an EMS system and other systems that are concerned with different aspects of power system operations, such as generation or distribution management.

The CIM is specified as a set of class diagrams using the Unified Modeling Language (UML). The base class of the CIM is the `PowerSystemResource` class, with other more specialized classes such as `Substation`, `Switch`, and `Breaker` being defined as subclasses. CIM/XML represents the CIM as an RDF Schema vocabulary, and uses RDF/XML as the language for exchanging specific system models. Example 41 shows examples of CIM/XML class and property definitions:

Example 41: Examples of CIM/XML Class and Property Definitions

```
<rdfs:Class rdf:ID="PowerSystemResource">
  <rdfs:label xml:lang="en">PowerSystemResource</rdfs:label>
  <rdfs:comment>"A power system component that can be either an
    individual element such as a switch or a set of elements
    such as a substation. PowerSystemResources that are sets
    could be members of other sets. For example a Switch is a
    member of a Substation and a Substation could be a member
    of a division of a Company"</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Breaker">
  <rdfs:label xml:lang="en">Breaker</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Switch" />
  <rdfs:comment>"A mechanical switching device capable of making,
     carrying, and breaking currents under normal circuit conditions
     and also making, carrying for a specified time, and breaking
     currents under specified abnormal circuit conditions e.g. those
     of short circuit. The typeName is the type of breaker, e.g.,
```

```
     oil, air blast, vacuum, SF6."</rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:ID="Breaker.ampRating">
   <rdfs:label xml:lang="en">ampRating</rdfs:label>
   <rdfs:domain rdf:resource="#Breaker" />
   <rdfs:range rdf:resource="#CurrentFlow" />
   <rdfs:comment>"Fault interrupting rating in amperes"</rdfs:comment>
</rdf:Property>
```

CIM/XML uses only a subset of the complete RDF/XML syntax, in order to simplify expressing the models. In addition, CIM/XML implements some extensions to the RDF Schema vocabulary. These extensions support the description of inverse roles and multiplicity (cardinality) constraints describing how many instances of a given property are allowed for a given resource (allowable values for a multiplicity declaration are zero-or-one, exactly-one, zero-or-more, one-or-more). The properties in Example 42 illustrate these extensions (which are identified by a `cims:` QName prefix):

Example 42: Some CIM/XML Extensions of RDF Schema

```
<rdf:Property rdf:ID="Breaker.OperatedBy">
   <rdfs:label xml:lang="en">OperatedBy</rdfs:label>
   <rdfs:domain rdf:resource="#Breaker" />
   <rdfs:range rdf:resource="#ProtectionEquipment" />
   <cims:inverseRoleName rdf:resource="#ProtectionEquipment.Operates" /
>
   <cims:multiplicity rdf:resource="http://www.cim-logic.com/
schema/990530#M:0..n" />
   <rdfs:comment>"Circuit breakers may be operated by
       protection relays."</rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="ProtectionEquipment.Operates">
   <rdfs:label xml:lang="en">Operates</rdfs:label>
   <rdfs:domain rdf:resource="#ProtectionEquipment" />
   <rdfs:range rdf:resource="#Breaker" />
   <cims:inverseRoleName rdf:resource="#Breaker.OperatedBy" />
   <cims:multiplicity rdf:resource="http://www.cim-logic.com/
schema/990530#M:0..n" />
   <rdfs:comment>"Circuit breakers may be operated by
       protection relays."</rdfs:comment>
</rdf:Property>
```

EPRI has conducted successful interoperability tests using CIM/XML to exchange real-life, large-scale models (involving, in the case of one test, data describing over 2000 substations) between a variety of vendor products, and validating that these models would be correctly interpreted by typical utility applications. Although the CIM was originally intended for EMS systems, it is also being extended to support power distribution and other applications as well.

The Object Management Group has adopted an object interface standard to access CIM power system models called the Data Access Facility [DAF]. Like the CIM/XML language, the DAF is based on the RDF model and shares the same CIM schema. However, while CIM/XML enables a model to be exchanged as a document, DAF enables an application to access the model as a set of objects.

CIM/XML illustrates the useful role RDF can play in supporting XML-based exchange of information that is naturally expressed as entity-relationship or object-oriented classes, attributes, and relationships (even when that information will not necessarily be Web-accessible). In these cases, RDF provides a basic structure for the XML in support of identifying objects, and using them in structured relationships. This connection is illustrated by a number of applications using RDF/XML for information interchange, as well as a number of projects investigating linkages between RDF (or ontology languages such as OWL) and UML (and its XML representations). CIM/XML's need to extend RDF Schema to support cardinality constraints and inverse relationships also illustrates the kinds of requirements that have led to the development of more powerful RDF-based schema/ontology languages such as DAML+OIL and OWL described in Section 5.5. Such languages may be appropriate in supporting many similar modeling applications in the future.

Finally, CIM/XML also illustrates an important fact for those looking for additional examples of "RDF in the Field": sometimes languages are described as "XML" languages, or systems are described as using "XML", and the "XML" they are actually using is RDF/XML, i.e., they are RDF applications. Sometimes it is necessary to go fairly far into the description of the language or system in order to find this out (in some examples that have been found, RDF is never explicitly mentioned at all, but sample data clearly shows it is RDF/XML). Moreover, in applications such as CIM/XML, the RDF that is created will not be readily found on the Web, since it is intended for information exchange between software components rather than for general access (although future scenarios could be imagined in which more of this type of RDF would become Web-accessible).

6.6 Gene Ontology Consortium

Structured metadata using controlled vocabularies such as SNOMED RT (Systematized Nomenclature of Medicine Reference Terminology) and MeSH (Medical Subject Headings) plays an important role in medicine, enabling more efficient literature searches and aiding in the distribution and exchange of medical knowledge [COWAN]. At the same time, the field of medicine is rapidly changing, and with that comes the need to develop additional vocabularies.

The objective of the Gene Ontology (GO) Consortium [GO] is to provide controlled vocabularies to describe specific aspects of gene products. Collaborating databases annotate their gene products (or genes) with GO terms, providing references and indicating what kind of evidence is available to support the annotations. The use of common GO terms by these databases facilitates uniform queries across them. The GO ontologies are structured to allow both attribution and querying to be performed at different levels of granularity. The GO vocabularies are dynamic, since knowledge of gene and protein roles in cells is accumulating and changing.

The three organizing principles of the GO are *molecular function*, *biological process*, and *cellular component*. A gene product has one or more molecular functions and is used in one or more biological processes; it may be, or may be associated with, one or more cellular components. Definitions of the terms within all three of these ontologies are contained in a single (text) definition file. XML formatted versions, containing all three ontology files and all available definitions, are generated monthly.

Function, process and component are represented as directed acyclic graphs (DAGs) or networks. A child term may be an "instance" of its parent term (isa relationship) or a component of its parent term (part-of relationship). A child term may have more than one parent term and may have a different class of relationship with its different parents. Synonyms and cross-references to external databases are also represented in the ontologies. GO uses RDF/XML facilities to represent the relationships between terms in the XML versions of the ontologies, because of its flexibility in representing these graph structures, as well as its widespread tool

support. At the same time, GO currently uses *non*-RDF nested XML structures within the term descriptions, so the language used is not pure RDF/XML.

Example 43 shows some sample GO information from the GO documentation:

Example 43: Sample GO Information

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE go:go>
<go:go xmlns:go="http://www.geneontology.org/xml-dtd/go.dtd#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <go:version timestamp="Wed May 9 23:55:02 2001" />

  <rdf:RDF>
     <go:term rdf:about="http://www.geneontology.org/go#GO:0003673">
        <go:accession>GO:0003673</go:accession>
        <go:name>Gene_Ontology</go:name>
        <go:definition></go:definition>
     </go:term>

     <go:term rdf:about="http://www.geneontology.org/go#GO:0003674">
        <go:accession>GO:0003674</go:accession>
        <go:name>molecular_function</go:name>
        <go:definition>The action characteristic of a gene product.</
go:definition>
        <go:part-of rdf:resource="http://www.geneontology.org/
go#GO:0003673" />
        <go:dbxref>
           <go:database_symbol>go</go:database_symbol>
           <go:reference>curators</go:reference>
        </go:dbxref>
     </go:term>

     <go:term rdf:about="http://www.geneontology.org/go#GO:0016209">
        <go:accession>GO:0016209</go:accession>
        <go:name>antioxidant</go:name>
        <go:definition></go:definition>
        <go:isa rdf:resource="http://www.geneontology.org/
go#GO:0003674" />
        <go:association>
           <go:evidence evidence_code="ISS">
              <go:dbxref>
                 <go:database_symbol>fb</go:database_symbol>
                 <go:reference>fbrf0105495</go:reference>
              </go:dbxref>
           </go:evidence>
           <go:gene_product>
              <go:name>CG7217</go:name>
              <go:dbxref>
                 <go:database_symbol>fb</go:database_symbol>
                 <go:reference>FBgn0038570</go:reference>
              </go:dbxref>
           </go:gene_product>
        </go:association>
        <go:association>
           <go:evidence evidence_code="ISS">
```

```
                <go:dbxref>
                   <go:database_symbol>fb</go:database_symbol>
                   <go:reference>fbrf0105495</go:reference>
                </go:dbxref>
             </go:evidence>
             <go:gene_product>
                <go:name>Jafrac1</go:name>
                <go:dbxref>
                   <go:database_symbol>fb</go:database_symbol>
                   <go:reference>FBgn0040309</go:reference>
                </go:dbxref>
             </go:gene_product>
          </go:association>
       </go:term>
   </rdf:RDF>
</go:go>
```

Example 43 illustrates that `go:term` is the basic element. In some cases, the GO has defined its own terms rather than using RDF Schema. For example, term `GO:0016209` has the element `<go:isa rdf:resource="http://www.geneontology.org/go#GO:0003674" />`. This tag represents the relationship "`GO:0016209` isa `GO:0003674`", or, in English, "Antioxidant is a molecular function." Another specialized relationship is `go:part-of`. For example, `GO:0003674` has the element `<go:part-of rdf:resource="http://www.geneontology.org/go#GO:0003673" />`. This says that "Molecular function is part of the Gene Ontology".

Every annotation must be attributed to a source, which may be a literature reference, another database or a computational analysis. The annotation must indicate what kind of evidence is found in the cited source to support the association between the gene product and the GO term. A simple controlled vocabulary is used to record evidence. Examples include:

- ISS means "inferred from sequence similarity [with <database:sequence_id>]"
- IDA means "inferred from direct assay"
- TAS means "traceable author statement"

The `go:dbxref` element represents the term in an external database, and `go:association` represents the gene associations of each term. `go:association` can have both `go:evidence`, which holds a `go:dbxref` to the evidence supporting the association, and a `go:gene_product`, which contains the gene symbol and `go:dbxref`. These elements illustrate that the GO XML syntax is not "pure" RDF/XML, since the nesting of other elements within these elements does not conform to the alternate node/predicate arc "stripes" described in Sections 2.1 and 2.2 of [RDF-SYNTAX].

The GO illustrates a number of interesting points. First, it shows that the value of using XML for information exchange can be enhanced by structuring that XML using RDF. This is particularly true for data that has an overall graph or network structure, rather than being a strict hierarchy. The GO is also another example in which data using RDF will not necessarily appear for direct use on the Web (although the files are Web-accessible). It is also another example of data which is, on the surface, described as "XML", but on closer examination uses RDF/XML facilities (albeit not "pure" RDF/XML). Finally, the GO illustrates the role RDF can play as a basis for representing ontologies. This role will be further enhanced once richer RDF-based languages for specifying ontologies, such as the DAML+OIL or OWL languages discussed in Section 5.5, become more widely used. In fact, a Gene Ontology Next Generation project is currently developing a representation of the GO ontologies in these richer languages.

6.7 Describing Device Capabilities and User Preferences

In recent years a large number of new mobile devices for browsing the Web have appeared. Many of these devices have highly divergent capabilities including a wide range of input and output capabilities as well as different levels of language support. Mobile devices may also have widely differing network connectivity capabilities. Users of these new devices expect a usable presentation regardless of the device's capabilities or the current network characteristics. Likewise, users want their dynamically changing preferences (e.g. turn audio on/off) to be considered when content or an application is presented. The reality, however, is that device heterogeneity, and the lack of a standard way for users to convey their preferences to the server, may result in: content that cannot be stored on the device, content that cannot be displayed, or content that violates the desires of the user. Additionally, the resulting content may take too long to convey over the network to the client device.

A solution for addressing these problems is for a client to encode its *delivery context* - the device's capabilities, the user's preferences, the network characteristics, etc. - in such a way that a server can use the context to customize content for the device and user (see [DIPRINC] for a definition of delivery context). The W3C's Composite Capabilities/Preferences Profile (CC/PP) specification [CC/PP] helps to address this problem by defining a generic framework for describing a delivery context.

The CC/PP framework defines a relatively simple structure - a two-level hierarchy of components and attribute/value pairs. A *component* may be used to capture a part of a delivery context (e.g. network characteristics, software supported by a device, or the hardware characteristics of a device). A component may contain one or more *attributes*. For example a component that encodes user preferences may contain an attribute to specify whether or not *AudioOutput* is desired.

CC/PP defines its structure (the hierarchy described above) using RDF Schema (see [CC/PP] for details of the structure schema). A CC/PP *vocabulary* defines specific components and their attributes. [CC/PP], however, does not define such vocabularies. Instead, vocabularies are defined by other organizations or applications (as described below). [CC/PP] also does not define a protocol for transporting an instance of a CC/PP vocabulary.

An instance of a CC/PP vocabulary is called a *profile*. CC/PP attributes are encoded as RDF properties in a profile. Example 44 shows a profile fragment of user preferences for a user that prefers an audio presentation:

Example 44: A CC/PP Profile Fragment

```
 <ccpp:component>
  <rdf:Description rdf:ID="UserPreferences">
   <rdf:type rdf:resource="http://www.example.org/profiles/prefs/
v1_0#UserPreferences"/>
   <ex:AudioOutput>Yes</ex:AudioOutput>
   <ex:Graphics>No</ex:Graphics>
   <ex:Languages>
    <rdf:Seq>
     <rdf:li>en-cockney</rdf:li>
     <rdf:li>en</rdf:li>
    </rdf:Seq>
   </ex:Languages>
  </rdf:Description>
 </ccpp:component>
```

There are several advantages to using RDF in this application. First, a profile encoded via CC/PP may include attributes that were defined in schemas created by different organizations. RDF is a natural fit for these profiles because no single organization is likely to create a *super* schema for the aggregated profile data. A second advantage of RDF is that it facilitates (by virtue of its graph-based data model) the insertion of arbitrary attributes (RDF properties) into a profile. This is particularly useful for profiles that include frequently changing data such as location information.

The Open Mobile Alliance has defined the User Agent Profile (UAProf) [UAPROF] - a CC/PP-based framework that includes a vocabulary for describing device capabilities, user agent capabilities, network characteristics, etc., as well as a protocol for transporting a profile. UAProf defines six components including: *HardwarePlatform*, *SoftwarePlatform*, *NetworkCharacteristics* and *BrowserUA*. It also defines several attributes for each of its components although a component's attributes are not fixed - they may be supplemented or overridden. Example 45 shows a fragment of UAProf's *HardwarePlatform* component:

Example 45: A Fragment of UAProf's HardwarePlatform Component

```
 <prf:component>
  <rdf:Description rdf:ID="HardwarePlatform">
   <rdf:type rdf:resource="http://www.openmobilealliance.org/profiles/
UAPROF/ccppschema-20021113#HardwarePlatform"/>
   <prf:ScreenSizeChar>15x6</prf:ScreenSizeChar>
   <prf:BitsPerPixel>2</prf:BitsPerPixel>
   <prf:ColorCapable>No</prf:ColorCapable>
   <prf:BluetoothProfile>
    <rdf:Bag>
     <rdf:li>headset</rdf:li>
     <rdf:li>dialup</rdf:li>
     <rdf:li>lanaccess</rdf:li>
    </rdf:Bag>
   </prf:BluetoothProfile>
  </rdf:Description>
 </prf:component>
```

The UAProf protocol supports both *static* profiles and *dynamic* profiles. A *static* profile is accessed via a URI. This has several advantages: a client's request to a server only contains a URI rather a potentially verbose XML document (thus minimizing over the air traffic); the client does not have to store and/or create the profile; the implementation burden on a client is relatively light-weight. *Dynamic* profiles are created on-the-fly and consequently do not have an associated URI. They may consist of a profile fragment containing a *difference* from a static profile, but they may also contain unique data that is not included in the client's static profile. A request may contain any number of static profiles and dynamic profiles. However, the ordering of the profiles is important as later profiles override earlier profiles in the request. See [UAPROF] for more information about UAProf's protocol and its rules for resolving multiple profiles.

Several other communities (i.e. 3GPP's TS 26.234 [3GPP] and the WAP Forum's Multimedia Messaging Service Client Transactions Specification [MMS-CTR]) have defined vocabularies based on CC/PP. As a result, a profile may take advantage of the distributed nature of RDF and include components defined from various vocabularies. Example 46 shows such a profile:

Example 46: A Profile Using Several Vocabularies

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:prf="http://www.wapforum.org/profiles/UAPROF/ccppschema-
20010330#"
```

```
         xmlns:mms="http://www.wapforum.org/profiles/MMS/ccppschema-
20010111#"
     xmlns:pss="http://www.3gpp.org/profiles/PSS/ccppschema-YYYYMMDD#">

 <rdf:Description rdf:ID="SomeDevice">
  <prf:component>
   <rdf:Description rdf:ID="Streaming">
    <rdf:type rdf:resource="http://www.3gpp.org/profiles/PSS/
ccppschema-PSS5#Streaming"/>
    <pss:AudioChannels>Stereo</pss:AudioChannels>
    <pss:VideoPreDecoderBufferSize>30720</pss:
VideoPreDecoderBufferSize>
    <pss:VideoInitialPostDecoderBufferingPeriod>0</pss:
VideoInitialPostDecoderBufferingPeriod>
    <pss:VideoDecodingByteRate>16000</pss:VideoDecodingByteRate>
   </rdf:Description>
  </prf:component>

  <prf:component>
   <rdf:Description rdf:ID="MmsCharacteristics">
    <rdf:type rdf:resource="http://www.wapforum.org/profiles/MMS/
ccppschema-20010111#Streaming"/>
    <mms:MmsMaxMessageSize>2048</mms:MmsMaxMessageSize>
    <mms:MmsMaxImageResolution>80x60</mms:MmsMaxImageResolution>
    <mms:MmsVersion>2.0</mms:MmsVersion>
   </rdf:Description>
  </prf:component>

  <prf:component>
   <rdf:Description rdf:ID="PushCharacteristics">
    <rdf:type rdf:resource="http://www.openmobilealliance.org/profiles/
UAPROF/ccppschema-20010330#PushCharacteristics"/>
    <prf:Push-MsgSize>1024</prf:Push-MsgSize>
    <prf:Push-MaxPushReq>5</prf:Push-MaxPushReq>
    <prf:Push-Accept>
     <rdf:Bag>
      <rdf:li>text/html</rdf:li>
      <rdf:li>text/plain</rdf:li>
      <rdf:li>image/gif</rdf:li>
     </rdf:Bag>
    </prf:Push-Accept>
   </rdf:Description>
  </prf:component>

 </rdf:Description>
</rdf:RDF>
```

The definition of a delivery context and the data within a context will continually evolve. Consequently, RDF's inherent extensibility, and thus support for dynamically changing vocabularies, make RDF a good framework for encoding a delivery context.

7. Other Parts of the RDF Specification

Section 1 indicated that the RDF Specification consists of a number of documents (in addition to this Primer):

- RDF Concepts and Abstract Syntax [RDF-CONCEPTS]
- RDF/XML Syntax Specification [RDF-SYNTAX]
- RDF Vocabulary Description Language 1.0: RDF Schema [RDF-VOCABULARY]
- RDF Semantics [RDF-SEMANTICS]
- RDF Test Cases [RDF-TESTS]

The Primer has already discussed the subjects of several of these documents, basic RDF concepts (in Section 2), the RDF/XML syntax (in Section 3) and RDF Schema (in Section 5). This section briefly describes the remaining documents (even though there have already been numerous references to [RDF-SEMANTICS] as well), in order to explain their role in the complete specification of RDF.

7.1 RDF Semantics

As discussed in the preceding sections, RDF is intended to be used to express statements about resources in the form of a graph, using specific vocabularies (names of resources, properties, classes, etc.). RDF is also intended to be the foundation for more advanced languages, such as those discussed in Section 5.5. In order to serve these purposes, the "meaning" of an RDF graph must be defined in a very precise manner.

Exactly what constitutes the "meaning" of an RDF graph in a very general sense may depend on many factors, including conventions within a user community to interpret user-defined RDF classes and properties in specific ways, comments in natural language, or links to other content-bearing documents. As noted briefly in Section 2.2, much of the meaning conveyed in these forms will not be directly accessible to machine processing, although this meaning may be used by human interpreters of the RDF information, or by programmers writing software to perform various kinds of processing on that RDF information. However, RDF statements also have a *formal* meaning which determines, with mathematical precision, the conclusions (or *entailments*) that machines can draw from a given RDF graph. The RDF Semantics [RDF-SEMANTICS] document defines this formal meaning, using a technique called *model theory* for specifying the semantics of a formal language. [RDF-SEMANTICS] also defines the semantic extensions to the RDF language represented by RDF Schema, and by individual datatypes. In other words, the RDF model theory provides the formal underpinnings for all RDF concepts. Based on the semantics defined in the model theory, it is simple to translate an RDF graph into a logical expression with essentially the same meaning.

7.2 Test Cases

The RDF Test Cases [RDF-TESTS] supplement the textual RDF specifications with test cases (examples) corresponding to particular technical issues addressed by the RDF Core Working Group. To help describe these examples, the Test Cases document introduces a notation called N-Triples, which provides the basis for the triples notation used throughout this Primer. The test cases are published in machine-readable form at Web locations referenced by the Test Cases document, so developers can use these as the basis for automated testing of RDF software.

The test cases are divided into a number of categories:

- Positive and Negative Parser Tests: These test whether RDF/XML parsers produce a correct N-Triples output graph from legal RDF/XML input documents, or correctly report errors if the input documents are not legal RDF/XML.
- Positive and Negative Entailment Tests: These test whether proper entailments (conclusions) are or are not drawn from sets of specified RDF statements.

- Datatype-aware Entailment Tests: These are positive or negative entailment tests that involve the use of datatypes, and hence require additional support for the specific datatypes involved in the tests.
- Miscellaneous Tests: These are tests that do not fall into one of the other categories.

The test cases are not a complete specification of RDF, and are not intended to take precedence over the other specification documents. However, they are intended to illustrate the intent of the RDF Core Working Group with respect to the design of RDF, and developers may find these test cases helpful should the wording of the specifications be unclear on any point of detail.

8. References

8.1 Normative References

[RDF-CONCEPTS]
Resource Description Framework (RDF): Concepts and Abstract Syntax, Klyne G., Carroll J. (Editors), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. The latest version is http://www.w3.org/TR/rdf-concepts/.

[RDF-MIME-TYPE]
MIME Media Types, The Internet Assigned Numbers Authority (IANA). This document is http://www.iana.org/assignments/media-types/ . The registration for `application/rdf+xml` is archived at http://www.w3.org/2001/sw/RDFCore/mediatype-registration .

[RDF-MS]
Resource Description Framework (RDF) Model and Syntax Specification, Lassila O., Swick R. (Editors), World Wide Web Consortium, 22 February 1999. This version is http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. The latest version is http://www.w3.org/TR/REC-rdf-syntax/.

[RDF-SEMANTICS]
RDF Semantics, Hayes P. (Editor), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. The latest version is http://www.w3.org/TR/rdf-mt/.

[RDF-SYNTAX]
RDF/XML Syntax Specification (Revised), Beckett D. (Editor), W3C Recommendation, 10 February 2004. This version http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. The latest version is http://www.w3.org/TR/rdf-syntax-grammar/.

[RDF-TESTS]
RDF Test Cases, Grant J., Beckett D. (Editors), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/. The latest version is http://www.w3.org/TR/rdf-testcases/.

[RDF-VOCABULARY]
RDF Vocabulary Description Language 1.0: RDF Schema, Brickley D., Guha R.V. (Editors), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. The latest version is http://www.w3.org/TR/rdf-schema/.

[UNICODE]
The Unicode Standard, Version 3, The Unicode Consortium, Addison-Wesley, 2000. ISBN 0-201-61633-5, as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions/ for the latest version and additional information on versions of the standard and of the Unicode Character Database).

[URIS]
RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax, Berners-Lee T., Fielding

R., Masinter L., IETF, August 1998, http://www.isi.edu/in-notes/rfc2396.txt.

[XML]
Extensible Markup Language (XML) 1.0, Second Edition, Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. (Editors), World Wide Web Consortium, 6 October 2000. This version is http://www.w3.org/TR/2000/REC-xml-20001006. The latest version is http://www.w3.org/TR/REC-xml.

[XML-BASE]
XML Base, Marsh J. (Editor), World Wide Web Consortium, 27 June 2001. This version is http://www.w3.org/TR/2001/REC-xmlbase-20010627/. The latest version is http://www.w3.org/TR/xmlbase/.

[XML-NS]
Namespaces in XML, Bray T., Hollander D., Layman A. (Editors), World Wide Web Consortium, 14 January 1999. This version is http://www.w3.org/TR/1999/REC-xml-names-19990114/. The latest version is http://www.w3.org/TR/REC-xml-names/.

[XML-XC14N]
Exclusive XML Canonicalization Version 1.0, Boyer J., Eastlake D.E. 3rd, Reagle J. (Authors/Editors), World Wide Web Consortium, 18 July 2002. This version is http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/. The latest version is http://www.w3.org/TR/xml-exc-c14n/.

8.2 Informational References

[3GPP]
3GPP TS 26.234. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Transparent end-to-end packet switched streaming service; Protocols and codecs V5.2.0 (2002-09). This document is available at http://www.3gpp.org/specs/specs.htm via directory ftp://ftp.3gpp.org/specs/2002-09/Rel-5/26_series/.

[ADDRESS-SCHEMES]
Addressing Schemes, Connolly D., 2001. This document is http://www.w3.org/Addressing/schemes.html.

[BATES96]
Indexing and Access for Digital Libraries and the Internet: Human, Database, and Domain Factors, Bates M.J., 1996. This document is http://is.gseis.ucla.edu/research/mjbates.html.

[BERNERS-LEE98]
What the Semantic Web can represent, Berners-Lee T., 1998. This document is http://www.w3.org/DesignIssues/RDFnot.html.

[CC/PP]
Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies, Klyne G., Reynolds F., Woodrow C., Ohto H., Hjelm J., Butler M., Tran, L., W3C Recommendation, 15 January 2004. This version is http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/. The latest version is http://www.w3.org/TR/CCPP-struct-vocab/.

[CG]
Conceptual Graphs, Sowa J., ISO working document ISO/JTC1/SC32/WG2 N 000, 2 April 2001 (work in progress). Available at http://users.bestweb.net/~sowa/cg/cgstand.htm.

[CHARMOD]
Character Model for the World Wide Web 1.0, Dürst M., Yergeau F., Ishida R., Wolf M., Freytag A., Texin T. (Editors), World Wide Web Consortium, 20 February 2002 (work in progress). This version is http://www.w3.org/TR/2002/WD-charmod-20020220/. The latest version is http://www.w3.org/TR/charmod/.

[CIM]
Common Information Model (CIM): CIM 10 Version, EPRI, Palo Alto, CA: 2001, 1001976. This document is available at http://www.epri.com/attachments/286161_1001976(1).pdf

(267pp.).

[COWAN]

Metadata, Reuters Health Information, and Cross-Media Publishing , Cowan J., 2002. Presentation at Seybold New York 2002 Enterprise Publishing Conference. This document is http://seminars.seyboldreports.com/seminars/2002_new_york/presentations/014/cowan_john.ppt. An accompanying transcript is http://seminars.seyboldreports.com/2002_new_york/files/transcripts/doc/transcript_EP7.doc

[DAF]

Utility Management System (UMS) Data Access Facility, version 2.0, Object Management Group, November 2002. This document is available at http://www.omg.org/technology/documents/formal/UMS_Data_Access_Facility.htm.

[DAML+OIL]

DAML+OIL (March 2001) Reference Description, Connolly D., van Harmelen F., Horrocks I., McGuinness D.L., Patel-Schneider P.F., Stein L.A., World Wide Web Consortium, 18 December 2001. This document is http://www.w3.org/TR/daml+oil-reference.

[DC]

Dublin Core Metadata Element Set, Version 1.1: Reference Description, 02 June 2003. This version is http://dublincore.org/documents/2003/06/02/dces/. The latest version is http://dublincore.org/documents/dces/.

[DIPRINC]

Device Independence Principles. Gimson, R., Finkelstein, S., Maes, S., Suryanarayana, L., World Wide Web Consortium, 18 September 2001 (work in progress). This version is http://www.w3.org/TR/2001/WD-di-princ-20010918. The latest version is http://www.w3.org/TR/di-princ/.

[DWZ01]

XML for CIM Model Exchange , deVos A., Widergreen S.E., Zhu J., Proc. IEEE Conference on Power Industry Computer Systems, Sydney, Australia, 2001. This document is http://www.langdale.com.au/PICA/.

[GO]

Gene Ontology: tool for the unification of biology, The Gene Ontology Consortium, *Nature Genetics*, Vol. 25: 25-29, May 2000. Available at http://www.geneontology.org/GO_nature_genetics_2000.pdf

[GRAY]

Logic, Algebra and Databases, Gray P., Ellis Horwood Ltd., 1984. ISBN 0-85312-709-3, 0-85312-803-0, 0-470-20103-7, 0-470-20259-9.

[HAYES]

In Defense of Logic, Hayes P., Proceedings from the International Joint Conference on Artificial Intelligence, 1975, San Francisco. Morgan Kaufmann Inc., 1977. Also in *Computation and Intelligence: Collected Readings*, Luger G. (ed), AAAI press/MIT press, 1995. ISBN 0-262-62101-0.

[KIF]

Knowledge Interchange Format, Genesereth M., draft proposed American National Standard NCITS.T2/98-004. Available at http://logic.stanford.edu/kif/dpans.html.

[LBASE]

LBase: Semantics for Languages of the Semantic Web, Guha R. V., Hayes P., W3C Note, 10 October 2003. This version is http://www.w3.org/TR/2003/NOTE-lbase-20031010/. The latest version is http://www.w3.org/TR/lbase/.

[LUGER]

Artificial Intelligence: Structures and Strategies for Complex Problem Solving (3rd ed.), Luger G., Stubblefield W., Addison Wesley Longman, 1998. ISBN 0-805-31196-3.

[MATHML]

Mathematical Markup Language (MathML) Version 2.0, Carlisle D., Ion P., Miner R., Poppelier N. (Editors); Ausbrooks R., Buswell S., Dalmas S., Devitt S., Diaz A., Hunter R.,

Smith B., Soiffer N., Sutor R., Watt S. (Principal Authors), World Wide Web Consortium, 21 February 2001. This version is http://www.w3.org/TR/2001/REC-MathML2-20010221/. The latest version is http://www.w3.org/TR/MathML2/.

[MMS-CTR]
Multimedia Messaging Service Client Transactions Specification. WAP-206-MMSCTR-20020115-a. This document is available at http://www.openmobilealliance.org/.

[NAMEADDRESS]
Naming and Addressing: URIs, URLs, ..., Connolly D., 2002. This document is http://www.w3.org/Addressing/.

[OWL]
OWL Web Ontology Language Reference, Dean M., Schreiber G (Editors); van Harmelen F., Hendler J., Horrocks I., McGuinness D.L., Patel-Schneider P.F., Stein L.A. (Authors), W3C Recommendation, 10 February 2004. The latest version is http://www.w3.org/TR/owl-ref/.

[PRISM]
PRISM: Publishing Requirements for Industry Standard Metadata, Version 1.1, 19 February 2002. The latest version of the PRISM specification is available at http://www.prismstandard.org/.

[RDFISSUE]
RDF Issue Tracking, McBride B., 2002. This document is http://www.w3.org/2000/03/rdf-tracking/.

[RDF-S]
Resource Description Framework (RDF) Schema Specification 1.0 , Brickley D., Guha, R. V. (Editors), World Wide Web Consortium. 27 March 2000. This version is http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[RSS]
RDF Site Summary (RSS) 1.0, Beged-Dov G., Brickley D., Dornfest R., Davis I., Dodds L., Eisenzopf J., Galbraith D., Guha R.V., MacLeod K., Miller E., Swartz A., van der Vlist E., 2000. This document is http://purl.org/rss/1.0/spec.

[RUBY]
Ruby Annotation, Sawicki M., Suignard M., Ishikawa M., Dürst M., Texin T. (Editors), World Wide Web Consortium, 31 May 2001. This version is http://www.w3.org/TR/2001/REC-ruby-20010531/. The latest version is http://www.w3.org/TR/ruby/.

[SOWA]
Knowledge Representation: Logical, Philosophical and Computational Foundations, Sowa J., Brookes/Cole, 2000. ISBN 0-534-94965-7.

[SVG]
Scalable Vector Graphics (SVG) 1.1 Specification, Ferraiolo J., Fujisawa J., Jackson D. (Editors), World Wide Web Consortium, 14 January 2003. This version is http://www.w3.org/TR/2003/REC-SVG11-20030114/. The latest version is http://www.w3.org/TR/SVG11/.

[UAPROF]
User Agent Profile. OMA-WAP-UAProf-v1_1. This document is available at http://www.openmobilealliance.org/.

[WEBDATA]
Web Architecture: Describing and Exchanging Data, Berners-Lee T., Connolly D., Swick R., World Wide Web Consortium, 7 June 1999. This document is http://www.w3.org/1999/04/WebData.

[XLINK]
XML Linking Language (XLink) Version 1.0, DeRose S., Maler E., Orchard D. (Editors), World Wide Web Consortium, 27 June 2001. This version is http://www.w3.org/TR/2001/REC-xlink-20010627/. The latest version is http://www.w3.org/TR/xlink/.

[XML-SCHEMA2]
XML Schema Part 2: Datatypes, Biron P., Malhotra A. (Editors), World Wide Web

Consortium. 2 May 2001. This version is http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/. The latest version is http://www.w3.org/TR/xmlschema-2/.

[XPACKAGE]
XML Package (XPackage) 1.0 , Wilson G., Editor's Working Draft, 6 March 2003. This version is http://www.xpackage.org/specification/xpackage-draft-20030306.html. The latest version is http://www.xpackage.org/specification/.

9. Acknowledgments

This document has benefited from inputs from many members of the RDF Core Working Group. Specific thanks are due to Art Barstow, Dave Beckett, Dan Brickley, Ron Daniel, Ben Hammersley, Martyn Horner, Graham Klyne, Sean Palmer, Patrick Stickler, Aaron Swartz, Ralph Swick, and Garret Wilson who, together with the many people who commented on earlier versions of the Primer, provided valuable contributions to this document.

In addition, this document contains a significant contribution from Pat Hayes, Sergey Melnik, and Patrick Stickler, who led the development of the RDF datatype facilities described in the RDF family of specifications.

Frank Manola also thanks The MITRE Corporation, Frank's employer during most of the preparation of this document, for its support of his RDF Core Working Group activities under a MITRE Sponsored Research grant.

Appendix A: More on Uniform Resource Identifiers (URIs)

Note: This section is intended to provide a brief introduction to URIs. The definitive specification of URIs is RFC 2396 [URIS], which should be consulted for further details. Additional discussion of URIs can also be found in Naming and Addressing: URIs, URLs, ... [NAMEADDRESS].

As discussed in Section 2.1, the Web provides a general form of identifier, called the Uniform Resource Identifier (URI), for identifying (naming) resources on the Web. Unlike URLs, URIs are not limited to identifying things that have network locations, or use other computer access mechanisms. A number of different *URI schemes* (URI forms) have been already been developed, and are being used, for various purposes. Examples include:

- `http:` (Hypertext Transfer Protocol, for Web pages)
- `mailto:` (email addresses), e.g., `mailto:em@w3.org`
- `ftp:` (File Transfer Protocol)
- `urn:` (Uniform Resource Names, intended to be persistent location-independent resource identifiers), e.g., `urn:isbn:0-520-02356-0` (for a book)

A list of existing URI schemes can be found in Addressing Schemes [ADDRESS-SCHEMES], and it is a good idea to consider adapting one of the existing schemes for any specialized identification purposes, rather than trying to invent a new one.

No one person or organization controls who makes URIs or how they can be used. While some URI schemes, such as URL's `http:`, depend on centralized systems such as DNS, other schemes, such as `freenet:`, are completely decentralized. This means that, as with any other kind of name, no one needs special authority or permission to create a URI for something. Also, anyone can create URIs to refer to things they do not own, just as in ordinary language anyone

can use whatever name they like for things they do not own.

As also noted in Section 2.1, RDF uses *URI references* [URIS] to name subjects, predicates, and objects in RDF statements. A URI reference (or *URIref*) is a URI, together with an optional *fragment identifier* at the end. For example, the URI reference `http://www.example.org/index.html#section2` consists of the URI `http://www.example.org/index.html` and (separated by the "#" character) the fragment identifier `Section2`. RDF URIrefs can contain Unicode [UNICODE] characters (see [RDF-CONCEPTS]), allowing many languages to be reflected in URIrefs.

URIrefs may be either *absolute* or *relative*. An *absolute* URIref refers to a resource independently of the context in which the URIref appears, e.g., the URIref `http://www.example.org/index.html`. A *relative* URIref is a shorthand form of an absolute URIref, where some prefix of the URIref is missing, and information from the context in which the URIref appears is required to fill in the missing information. For example, the relative URIref `otherpage.html`, when appearing in a resource `http://www.example.org/index.html`, would be filled out to the absolute URIref `http://www.example.org/otherpage.html`. A URIref without a URI part is considered a reference to the current document (the document in which it appears). So, an empty URIref within a document is considered equivalent to the URIref of the document itself. A URIref consisting of just a fragment identifier is considered equivalent to the URIref of the document in which it appears, with the fragment identifier appended to it. For example, within `http://www.example.org/index.html`, if `#section2` appeared as a URIref, it would be considered equivalent to the absolute URIref `http://www.example.org/index.html#section2`.

[RDF-CONCEPTS] notes that RDF graphs (the abstract models) do not use relative URIrefs, i.e., the subjects, predicates, and objects (and datatypes in typed literals) in RDF statements must always be identified independently of any context. However, a specific concrete RDF syntax, such as RDF/XML, may allow relative URIrefs to be used as a shorthand for absolute URIrefs in certain situations. RDF/XML does permit such use of relative URIrefs, and some of the RDF/XML examples in this Primer illustrate such uses. [RDF-SYNTAX] should be consulted for further details.

Both RDF and Web browsers use URIrefs to identify things. However, RDF and browsers interpret URIrefs in slightly different ways. This is because RDF uses URIrefs *only* to identify things, while browsers also use URIrefs to *retrieve* things. Often there is no effective difference, but in some cases the difference can be significant. One obvious difference is that when a URIref is used in a browser, there is the expectation that it identifies a resource that can actually be retrieved: that something is actually "at" the location identified by the URI. However, in RDF a URIref may be used to identify something, such as a person, that *cannot* be retrieved on the Web. People sometimes use RDF together with a convention that, when a URIref is used to identify an RDF resource, a page containing descriptive information about that resource will be placed on the Web "at" that URI, so that the URIref can be used in a browser to retrieve that information. This can be a useful convention in some circumstances, although it creates a difficulty in distinguishing the identity of the original resource from the identity of the Web page describing it (a subject discussed further in Section 2.3). However, this convention is not an explicit part of the definition of RDF, and RDF itself does not assume that a URIref identifies something that can be retrieved.

Another difference is in the way URIrefs with fragment identifiers are handled. Fragment identifiers are often seen in the URLs that identify HTML documents, where they serve to identify a specific place within the document identified by the URL. In normal HTML usage, where URI references are used to retrieve the indicated resources, the two URIrefs:

`http://www.example.org/index.html`

```
http://www.example.org/index.html#Section2
```

are related (they both refer to the same document, the second one identifying a location within the first one). However, as noted already, RDF uses URI references purely to *identify* resources, not to retrieve them, and RDF assumes no particular relationship between these two URIrefs. As far as RDF is concerned, they are syntactically different URI references, and hence may refer to unrelated things. This does not mean that the HTML-defined containment relationship might not exist, just that RDF does not assume that a relationship exists based only on the fact that the URI parts of the URI references are the same.

Carrying this point further, RDF does not assume that there is any relationship between URI references that share a common leading string, whether there is a fragment identifier or not. For example, as far as RDF is concerned, the two URIrefs:

```
http://www.example.org/foo.html
http://www.example.org/bar.html
```

have no particular relationship even though both of them start with the string `http://www.example.org/`. To RDF, they are simply different resources, because their URIrefs are different. (They may in fact be two files located in the same directory, but RDF does not assume this or any other relationship exists.)

Appendix B: More on the Extensible Markup Language (XML)

Note: This section is intended to provide a brief introduction to XML. The definitive specification of XML is [XML], which should be consulted for further details.

The Extensible Markup Language [XML] was designed to allow anyone to design their own document format and then write a document in that format. Like HTML documents (Web pages), XML documents contain text. This text consists primarily of plain text content, and markup in the form of *tags*. This markup allows a processing program to interpret the various pieces of content (called *elements*). Both XML content and (with certain exceptions) tags can contain Unicode [UNICODE] characters, allowing information from many languages to be directly represented. In HTML, the set of permissible tags, and their interpretation, is defined by the HTML specification. However, XML allows users to define their own markup languages (tags and the structures in which they can appear) adapted to their own specific requirements (the RDF/XML language described in Section 3 is one such XML markup language). For example, the following is a simple passage marked up using an XML-based markup language:

```
<sentence><person webid="http://example.com/#johnsmith">I</person>
just got a new pet <animal>dog</animal>.</sentence>
```

Elements delimited by tags (`<sentence>`, `<person>`, etc.) are introduced to reflect a particular structure associated with the passage. The tags allow a program written with an understanding of these particular elements, and the way they are structured, to properly interpret the passage. For example, one of the elements in this example is `<animal>dog</animal>`. This consists of the *start-tag* `<animal>`, the element *content*, and a matching *end-tag* `</animal>`. This `animal` element, together with the `person` element, are nested as part of the content of the `sentence` element. The nesting is possibly clearer (and closer to some of the more "structured" XML contained in the rest of this Primer) if the sentence is written:

```
<sentence>
    <person webid="http://example.com/#johnsmith">I</person>
```

```
    just got a new pet
    <animal>dog</animal>.
</sentence>
```

In some cases, an element may have no content. This can be written either by enclosing no content within the pair of delimiting start- and end-tags, as in `<animal></animal>`, or by using a shorthand form of tag called an *empty-element tag*, as in `<animal/>`.

In some cases, a start-tag (or empty-element tag) may contain qualifying information other than the tag name, in the form of *attributes*. For example, the start-tag of the `<person>` element contains the attribute `webid="http://example.com/#johnsmith"` (presumably identifying the specific person referred to). An attribute consists of a name, an equal sign, and a value (enclosed in quotes).

This particular markup language uses the words "sentence," "person," and "animal" as tag names in an attempt to convey some of the meaning of the elements; and they *would* convey meaning to an English-speaking person reading it, or to a program specifically written to interpret this vocabulary. However, there is no built-in meaning here. For example, to non-English speakers, or to a program not written to understand this markup, the element `<person>` may mean absolutely nothing. Take the following passage, for example:

```
<dfgre><reghh bjhbw="http://example.com/#johnsmith">I</reghh>
just got a new pet <yudis>dog</yudis>.</dfgre>
```

To a machine, this passage has exactly the same structure as the previous example. However, it is no longer clear to an English-speaker what is being said, because the tags are no longer English words. Moreover, others may have used the same words as tags in their own markup languages, but with completely different intended meanings. For example, "sentence" in another markup language might refer to the amount of time that a convicted criminal must serve in a penal institution. So additional mechanisms must be provided to help keep XML vocabulary straight.

To prevent confusion, it is necessary to uniquely identify markup elements. This is done in XML using XML Namespaces [XML-NS]. A *namespace* is just a way of identifying a part of the Web (space) which acts as a qualifier for a specific set of names. A namespace is created for an XML markup language by creating a URI for it. By qualifying tag names with the URIs of their namespaces, anyone can create their own tags and properly distinguish them from tags with identical spellings created by others. A convention that is sometimes followed is to create a Web page to describe the markup language (and the intended meaning of the tags) and use the URL of that Web page as the URI for its namespace. However, this is just a convention, and neither XML nor RDF assumes that a namespace URI identifies a retrievable Web resource. The following example illustrates the use of an XML namespace.

```
<user:sentence xmlns:user="http://example.com/xml/documents/">
   <user:person user:webid="http://example.com/#johnsmith">I</user:
person>
just got a new pet <user:animal>dog</user:animal>.
</user:sentence>
```

In this example, the attribute `xmlns:user="http://example.com/xml/documents/"` declares a namespace for use in this piece of XML. It maps the *prefix* `user` to the namespace URI `http://example.com/xml/documents/`. The XML content can then use *qualified names* (or *QNames*) like `user:person` as tags. A QName contains a prefix that identifies a

namespace, followed by a colon, and then a *local name* for an XML tag or attribute name. By using namespace URIs to distinguish specific groups of names, and qualifying tags with the URIs of the namespaces they come from, as in this example, there is no need to worry about tag names conflicting. Two tags having the same spelling are considered the same only if they also have the same namespace URIs.

Every XML document is required to be *well-formed*. This means the XML document must satisfy a number of syntactic conditions, for example, that every start-tag must have a matching end-tag, and that elements must be properly nested within other elements (elements may not overlap). The complete set of well-formedness conditions is defined in [XML].

In addition, an XML document may optionally include an XML *document type declaration* to define additional constraints on the structure of the document, and to support the use of predefined units of text within the document. The document type declaration (introduced with `DOCTYPE`) contains or points to declarations that define a grammar for the document. This grammar is known as a *document type definition*, or *DTD*. The declarations in a DTD specify such things as which XML elements and attributes may appear in XML documents corresponding to the DTD, the relationships of these elements and attributes (e.g., which elements can be nested within which other elements, or which attributes may appear with which elements), and whether elements or attributes are required or optional. The document type declaration can point to a set of declarations located outside the document (called the *external subset*, which can be used to allow common declarations to be shared among multiple documents), can include the declarations directly in the document (called the *internal subset*), or can have both internal and external DTD subsets. The complete DTD for a document consists of both subsets taken together. A simple example of an XML document with a document type declaration is shown in Example 47:

Example 47: An XML Document With a Document Type Declaration

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "http://www.example.org/dtds/hello.dtd">
<greeting>Hello, world!</greeting>
```

In this case, the document has only an external DTD subset, and the *system identifier* `http://www.example.org/dtds/hello.dtd` provides its location (a URIref).

An XML document is *valid* if it has an associated document type declaration and the document complies with the constraints defined by the document type declaration.

An RDF/XML document is only required to be well-formed XML; it is not intended to be validated against an XML DTD (or an XML Schema), and [RDF-SYNTAX] does not specify a normative DTD that could be used for validating arbitrary RDF/XML (an appendix of [RDF-SYNTAX] does provide a non-normative example schema for RDF/XML). As a result, more detailed discussion of XML DTD grammars is beyond the scope of this Primer. Further information on XML DTDs and XML validation can be found in [XML], and the numerous books on XML.

However, there is one use of XML document type declarations that *is* relevant to RDF/XML, and that is their use in defining XML *entities*. An XML entity declaration essentially associates a name with a string of characters. When the entity name is used elsewhere within an XML document, XML processors replace the entity name with the corresponding string. This provides a way to abbreviate long strings such as URIrefs, and can help make XML documents containing such strings more readable. Using a document type declaration just to declare XML entities is allowed, and can be useful, even when (as in RDF/XML) the documents are not intended to be validated.

In RDF/XML documents, entities are generally declared within the document itself, i.e., using only

an internal DTD subset (one reason for this is that RDF/XML is not intended to be validated, and non-validating XML processors are not required to process external DTD subsets). For example, providing the document type declaration shown in Example 48 at the beginning of an RDF/XML document allows the URIrefs in that document for the `rdf`, `rdfs`, and `xsd` namespaces to be abbreviated as `&rdf;`, `&rdfs;`, and `&xsd;` respectively, as shown in the example.

Example 48: Some XML Entity Declarations

```
<?xml version='1.0'?>

<!DOCTYPE rdf:RDF [
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>

<rdf:RDF
    xmlns:rdf = "&rdf;"
    xmlns:rdfs = "&rdfs;"
    xmlns:xsd = "&xsd;">

...RDF statements...

</rdf:RDF>
```

Appendix C: Changes

Only minor editorial and typographic changes have been made since the Proposed Recommendation version. Older changes are detailed in its change log.

Extensible Markup Language (XML) Activity Statement

Work on the Extensible Markup Language (XML) is being managed as part of W3C's Architecture Domain.

> *Activity statements provide an executive overview of W3C's work in this area. The XML home page points to highlights and has links to the individual Working Group pages.*

Introduction

The Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web.

Some XML benefits in brief:

- Enables internationalized media-independent electronic publishing
- Saves businesses money by enabling the use of inexpensive off-the-shelf tools to process data
- Saves training and development costs by having a single format for a wide range of uses
- Increases reliability, because user agents can automate more processing of documents they receive
- Provides the underpinnings of the Semantic Web, enabling a whole new level of interoperability and information interchange
- Encourages industries to define platform-independent protocols for the exchange of data, including electronic commerce

- Allows people to display information the way they want it, under style sheet control
- Enables long-term reuse of data, with no lock-in to proprietary tools or undocumented formats

XML in Perspective

XML in 10 points is an essay by Bert Bos that covers the basics of XML and its relationship to other W3C technologies. It is written for beginners and is the place to start.

Simple Example of XML Usage

The best way to appreciate what XML documents look like is with a simple example. Imagine your company sells products on-line. Marketing descriptions of the products are written in HTML, but names and addresses of customers, and also prices and discounts are formatted with XML. Here is the information describing a customer:

```
<customer-details id="AcPharm39156">
      <name>Acme Pharmaceuticals Co.</name>
      <address country="US">
            <street>7301 Smokey Boulevard</
street>
            <city>Smallville</city>
            <state>Indiana</state>
            <postal>94571</postal>
      </address>
</customer-details>
```

The XML syntax uses matching start and end tags, such as `<name>` and `</name>`, to mark up information. A piece of information marked by the presence of tags is called an *element*; elements may be further enriched by attaching name-value pairs (for example, `country="US"` in the example above) called *attributes*. Its simple syntax is easy to process by machine, and has the attraction of remaining understandable to humans. XML is based on SGML, and is familiar in look and feel to those accustomed to HTML.

Outside and inside W3C, many groups are using XML to define new formats for information interchange. The number of XML applications is growing rapidly, and the growth appears likely to continue. There are many areas, for example, the health-

care industry, the Inland Revenue, government and finance, where XML applications are used to store and process data.

Role of the W3C

XML was originally developed at the W3C. The XML Core Working Group continues to develop and maintain the specifications for XML itself and closely related specifications. The W3C is also the primary center for developing other cross-industry specifications that are based on XML. Some of these are being done within the XML Activity, such as XML Query and XML Schema, and some are being done in other W3C Activities, such as DOM, SVG and XHTML.

Current Situation

This section gives more detail about the individual Working Groups within the XML Activity.

XML Coordination Group

This group comprises the Chairs of the individual Working Groups. It provides a forum for coordination between the Working Groups of the XML Activity, between the XML Activity and other parts of W3C, and between the XML Activity and other organizations. In particular, the Coordination Group:

- Coordinates workflow.
- Watches out for dependencies between Working Groups.
- Creates and dissolves Working Groups within its chartered scope.
- Sets meeting schedules.
- Turns policy and architecture questions over to the W3C internal management or to the Technical Architecture Group as appropriate.
- Maintains liaison inside and outside the W3C.

The chair of the XML Coordination Group is C. M. Sperberg-McQueen of W3C.

XML Core Working Group

The **XML Core Working Group** [Members only] is responsible for supporting the XML Recommendation, both with new versions such as XML 1.1 and with ongoing revisions such as XML 1.0 Third Edition. This support extends to errata and to other closely related specifications, including Namespaces in XML and the XML Information Set.

Paul Grosso of Arbortext and Norm Walsh of Sun Microsystems are co-Chairs of

the XML Core Working Group.

Since the last W3C Advisory Committee meeting (November 2003)

Extensible Markup Language (XML) 1.1 and Namespaces in XML 1.1 are now W3C Recommendations. In addition, a Second Edition of the XML Information Set is also a W3C Recommendation, and updates the Information Set so that specifications referring to it need no longer be tied to a specific version of XML.

The XML Core Working Group has published the first Working Draft for `xml:id`, a way to give XML elements an identity without the need for validation, and also a Note, XML Processing Model Requirements.

There has also been ongoing work on republishing the IETF MIME type specifiation for XML, RFC 3023, and on advancing the XInclude specification again after it was sent back to Working Draft in 2003.

XSL Working Group

The **XSL Working Group** [Members only] joined the XML Activity in 2003. They develop and maintain XSL Transformations (XSLT) for transforming XML documents, XSL Formatting Objects (XSL/FO) for formatting XML documents, and, jointly with the XML Query Working Group, XPath 2.0. XPath is used to address, point into, or match portions of XML documents. In addition, the XSL Working Group maintains a number of other documents which are published jointly with the XQuery Working Group, including the XQuery 1.0 and XPath 2.0 Data Model, XQuery 1.0 and XPath 2.0 Formal Semantics, XQuery 1.0 and XPath 2.0 Functions and Operators and XSLT 2.0 and XQuery 1.0 Serialization.

Since November 2003

The XSL Working Group has been working closely with the XML Query Working Group on XPath 2 and its associated documents. The XSLT 2.0, XPath 2.0, and associated documents went into (and out of) Last Call, and together with the XML Query Working Group, XSL has been working on processing the over 1100 comments sent to the public comments list.

The XSL Working Group is also actively working on XSL 1.1 (formatting objects) and hopes to publish a draft shortly.

XML Query Working Group

Following the W3C Query Languages Workshop (QL'98), the mission of the **XML**

Query Working Group [Members only] is to provide flexible query facilities to extract data from real and virtual documents on the Web.

The XML Query Working Group maintains a number of documents, some of which are published jointly with the XSL Working Group. The full list is on the XML Query Working Group public page.

Paul Cotton of Microsoft is the Chair of the XML Query Working Group.

Since May 2003

This Working Group has been moving closer to getting the XML Query 1.0 specification published, by closing as many open issues as possible. There is a lot of interest in XML Query: there are already more than twenty implementations listed on the group's public page!

The Working Group has continued its work on XML Query and XPath Full-Text Requirements and XML Query and XPath Full-Text Use Cases which were published for the first time in February 2003.

Because of the high volume of comments, a second Last Call for the XML Query 1.0 is expected later in 2004.

XML Binary Characterization Working Group

The XML Binary Characterization Working Group was formed in March 2004 and is chartered to analyze the issues surrounding the exchange of XML information in a binary format.

This Working Group is *not* chartered to do the technical work of defining a binary format. A W3C Workshop on Binary Interchange of XML Information Item Sets in September of 2003 concluded that it was not clear that requirements were sufficiently well-defined. It was also clear that there was not enough information available to determine whether the potential interoperability loss of having multiple incompatible formats called XML, and the possible weakening of the XML brand, would be justified by any possible gains.

The XML Binary Characterization Working Group has been created to answer the question of whether W3C should do standards work in this area. It is chaired by Robin Berjon of Expway.

XML Schema Working Group

XML 1.0 supplies the Document Type Definition (DTD) mechanism for declaring

constraints on the use of markup, but automated processing of XML documents requires more rigorous and comprehensive facilities in this area.

The XML Schema specification (Part 0: Primer, Part 1: Structures, and Part 2: Datatypes) is a Recommendation as of May 2001. W3C XML Schema is widespread use in the XML industry, and is also used as a basis for the type system of the XML Query language, XPath 2.0 and XSLT 2.0 drafts, and elsewhere.

The Chair of the Schema Working Group is C. M. Sperberg-McQueen of the W3C.

Since November 2003

The XML Schema Working Group is currently working to develop a set of requirements for XML Schema 1.1, which is intended to be mostly compatible with XML Schema 1.0 and to have approximately the same scope. The Working Group has also done work on the post-schema-validation Infoset. There has also been work on a formal description of W3C XML schemas.

The XML Schema Working Group has also held joint meetings with the XML Query Working Group and the XSL Working Group in order to ensure that the XPath 2.0 data model and type system is consistent with W3C XML Schema 1.1.

Liaison and review work has taken up most of this Working Group's time.

What the Future Holds

The XML Activity is chartered through 2004-05-30, and is in the process of rechartering.

All of the Working Groups continue to maintain their specifications with errata, test suites, translations and other work, and of course to take the Working Drafts mentioned above forward toward Recommendations.

Careful liaison between Working Groups is a major part of the focus of the XML Activity right now, with several large and important specifications being honed and polished as they progress towards Recommendations.

Contact

Liam Quin, XML Activity Lead

Last modified $Date: 2004/05/05 21:28:58 $

NOTE-ddml-19990119

Document Definition Markup Language (DDML) Specification, Version 1.0

W3C Note, *19-Jan-1999*

This Version:
http://www.w3.org/TR/1999/NOTE-ddml-19990119
Latest Version:
http://www.w3.org/TR/NOTE-ddml
Editors:
Ronald Bourret <rbourret@ito.tu-darmstadt.de>, Darmstadt University of Technology
John Cowan <cowan@locke.ccil.org>
Ingo Macherius <macherius@gmd.de>, GMD
Simon St. Laurent <simonstl@simonstl.com>, simonstl.com

Status of this document

This document is a submission to the World Wide Web Consortium from GMD (see Submission Request, W3C Staff Comment).

This document is a NOTE made available by W3C for discussion only. This indicates no endorsement of its content, nor that W3C has had any editorial control in its preparation, nor that W3C has, is, or will be allocating any resources to the issues addressed by the NOTE.

DDML was formerly known as XSchema. For a list of differences between DDML and XSchema, see Appendix B, "Differences between DDML and XSchema".

Abstract

This document proposes Document Definition Markup Language (DDML), a schema language for XML documents. DDML encodes the logical (as opposed to physical) content of DTDs in an XML document. This allows schema information to be explored and used with widely available XML tools.

DDML is deliberately simple, providing an initial base for implementations. While introducing as few complicating factors as possible, DDML has been designed with future extensions, such as data typing and schema reuse, in mind.

Table of Contents

Appendices

1 Introduction

In order for document processing to be reliable, it is necessary to be able to describe classes of documents and to verify individual documents' membership in these classes -- in other words, to be able to express constraints on documents and thus define 'document types'. XML inherits a mechanism for doing this from SGML: the Document Type Definition. XML DTDs can perform a subset of the functions of SGML DTDs.

DTDs have limited expressiveness and it is necessary to experiment with new ideas in schema design. These ideas include a syntax that is more like that of XML document content, certain kinds of extensibility and a cleaner separation between parsing and verifying. DDML is an experimental schema language designed to provide a starting point for these experiments.

So that DDML documents will be immediately useful with existing software, the DDML specification will describe a conversion from DDML documents to DTDs. This initial version of the DDML specification is deliberately simple, providing an initial base for implementations while introducing as few complicating factors as possible. Authors accustomed to DTD creation will find their tool set constricted; it is hoped that supporting software and tools available from other standards will make up for this reduced tool set.

1.1 Origin and Goals

Proposals for describing SGML document type definitions using document syntax rather than the separate declaration syntax have been under development for a number of years, and used by several tools for documentation. The current proposal arose from a number of concerns surrounding XML's usability and consistency. Originally conceived of as a mapping of DTD syntax to document syntax, the project has developed into an effort focused on creating schemas describing element and attribute structures rather than preserving every function provided by XML 1.0 DTDs.

The list of goals developed by the xml-dev discussion follows:

1. DDML documents shall use XML document syntax, using element nesting and attributes to describe all constraints that may be verified by a processor using DDML .
2. DDML shall define a transformation from DDML documents to DTDs.
3. DDML documents shall be capable of representing the normalized element and attribute structures defined in XML 1.0 DTDs, and provide namespace support.
4. DDML documents shall be parseable, manageable, and manipulable using the same tools used to parse, manage, and manipulate XML documents.
5. DDML documents shall be easy to create, read, and modify, and shall provide authoring support for XML documents.
6. DDML documents shall be easy to use in combination with a parser to provide structural validation of documents.
7. DDML shall include a DDML document and an XML 1.0 DTD defining the structure of DDML documents.
8. DDML shall suggest mechanisms for applying DDML documents to documents.
9. DDML shall include mechanisms for extending the information included in DDML documents to support metadata.

10. The DDML specification shall be readable, clear, and rigorous, using terminology and nomenclature as close to the XML 1.0 specification as possible.
11. The DDML specification will comply with and be consistent with W3C recommendations.
12. DDML documents shall provide constructs for human- and machine-readable documentation.

1.2 Relation to Standards

DDML documents use XML 1.0 document instance syntax and may be applied to XML 1.0 [XML] documents. This specification refers to several IETF standards, notably Multipurpose Internet Mail Extensions (MIME) ([RFC 2046]and [RFC 2048]) and XML Media Types [RFC 2376].

Namespace usage in DDML is based on the 17 November, 1998 "Namespaces in XML" Proposed Recommendation [Namespaces]. Because this recommendation is still subject to change, all namespace attributes (the xmlns and xmlns:DDML attributes of the DocumentDef element and all ns, prefix, and ElementNS attributes) and processing (Section 3, "DDML and Namespaces") are subject to change, even after the rest of the DDML specification is finalized.

It is hoped that future versions of DDML will use [XLink] and [XPointer] to implement schema reuse.

DDML has been influenced by the XML-Data proposal [XML-Data]. It is hoped that DDML may be mapped to an RDF vocabulary.

1.3 Related Work

Several other proposals for XML schema languages also exist. These are Document Content Description for XML ([DCD]), Schema for Object-Oriented XML ([SOX]), and XML-Data [XML-Data]. Resource Description Framework (RDF) Schema Specification (RDF Schemas) proposes a related language, although this is not currently suitable for replacing DTDs.

1.4 Terminology

The requirement levels used throughout this document reflect the approach of [RFC 2119], though keywords (like may and must) are not capitalized. Other terms used are defined in the XML 1.0 Recommendation [XML].

1.5 Comments and History

Please send public comments to the XML-Dev mailing list [XML-DEV] and private comments to Simon St.Laurent or Ronald Bourret.

DDML is a cooperative effort of members of the XML-Dev mailing list and has been submitted as a Note to the W3C through the GMD - Forschungszentrum Informationstechnik GmbH, Bonn, Germany. For a complete list of contributors, see Appendix E, "Contributors". Historical information regarding the development of DDML is available at http://purl.oclc.org/NET/ddml.

2 DDML Syntax

This section describes DDML document syntax. In version 1.0, the DDML document is an XML document containing a single DocumentDef element in which information describing the schema is nested. The DocumentDef element must be preceded by an XML declaration and may be

preceded by other declarations, comments, and processing instructions. In future versions of DDML, DocumentDef elements may be embedded in instance documents.

2.1 The DocumentDef Element

The DocumentDef element is the root element for all DDML documents. The declaration for the DocumentDef element is:

```
<!ELEMENT DocumentDef (Doc?, More?, (ElementDecl | Model | AttDef |
AttGroup | Notation | UnparsedEntity | Enumeration | DocumentDef)*)>
<!ATTLIST DocumentDef
    xmlns         CDATA   #FIXED   "http://www.purl.org/NET/ddml/v1"
    xmlns:DDML    CDATA   #FIXED   "http://www.purl.org/NET/ddml/v1"
    ns            CDATA   #IMPLIED
    ElementNS     CDATA   #IMPLIED
    prefix        NMTOKEN #IMPLIED
    Version       CDATA   #FIXED   "1.0"
    MimeType      CDATA             "application/xml"
    FileExtension CDATA             "xml"
    id            ID      #IMPLIED>
```

The DocumentDef element contains other elements describing the document type and building a schema. These elements are described in later sections of this specification. The DocumentDef element may also contain other DocumentDef elements nested inside of it. This nesting of DocumentDef elements improves reusability of DDML documents by allowing the combination of multiple DDML documents inside of a single DDML document. It also allows finer-grained control over documentation for subsections of a DDML document.

The DocumentDef element's attributes include information about the namespaces used by DDML and elements defined with DDML, the version of the DDML specification used, and information about the type of documents described by the DDML document.

The xmlns and xmlns:DDML attributes identify the URI of the namespace containing the DocumentDef elements. For more information, see Section 3.1, "The DDML Namespace."

The ns attribute provides the URI of the namespace containing elements and attributes being declared, as with an ElementDecl element. The ElementNS attribute identifies the URI of the namespace of elements being referenced, as with a Ref element. The prefix attribute identifies the prefix to be used when converting the DDML document to a DTD. These attributes can be overridden in the elements that declare and reference elements and attributes. For more information, see Section 3.2, "Namespaces of Elements and Attributes Being Defined."

Information about the DDML specification version used to create this DDML document, contained in the Version attribute, is critical to proper handling of documents should the specification be updated in the future. This specification is identified as version 1.0. Future major and minor versions of the DDML specification should identify themselves differently. No provision is made at this time for nesting DocumentDef elements using different versions of the specification under a parent DocumentDef element.

The MimeType and FileExtension attributes are used to provide a suggested MIME (Multipurpose Internet Mail Extensions) Content-type and file extension for documents associated with a particular DDML document. Applications may use this information to identify XML document types. A document library that generates XML documents dynamically could assign file extensions and MIME types based on the DDML document used.

Applications using this information should use the values stored in the first DocumentDef element encountered during processing. For instance, if a DocumentDef element includes another nested DocumentDef element, the values for the MimeType and FileExtension attributes of the root DocumentDef element should be used.

By default, most XML documents are assumed to have a MIME type of application/xml, as described in [RFC 2376]. Developers who need different MIME types for documents associated with particular DDML documents may register other MIME types with the IETF, as described in [RFC 2048], or use the 'x-' prefix syntax for subtypes, as described in [RFC 2046].

For information about the id attribute, see Section 2.8, "id Attributes".

2.2 Element Declarations

Element declarations in DDML documents are made using the ElementDecl element and its contents:

```
<!ELEMENT ElementDecl (Doc?, More?, Model, AttGroup?)>
<!-- Name is the element name -->
<!ATTLIST ElementDecl
    Name   NMTOKEN #REQUIRED
    ns     CDATA   #IMPLIED
    prefix NMTOKEN #IMPLIED
    id     ID      #IMPLIED
    Root   (Recommended | Possible | Unlikely) "Possible">
```

The Name attribute identifies the name of the element, and is required. An element declaration would look like:

```
<ElementDecl Name="Species">
...additionalElementInformation...
</ElementDecl>
```

This declaration would declare an element named "Species", which would appear in an instance as:

```
<Species>...content...</Species>
```

The Name attribute must be unique within the set of elements in the defined namespace. It provides the name of the element as declared here and is also used by other elements to refer to this element in their content model declarations. The Name attribute must match the NCName production in [Namespaces]. (Effectively, this requires element names to begin with a letter or underscore and not include a colon.)

The ns attribute identifies the URI of the namespace containing this element and its attributes. The prefix attribute identifies the prefix that will be applied to this element and its attributes during conversion to DTDs. These attributes override the values of the same attributes on the DocumentDef element and can be overridden in the AttGroup and AttDef elements. For more information, see Section 3.2, "Namespaces of Elements and Attributes Being Defined."

The Root attribute provides authoring tools with a guide for which elements are likely root elements for documents. This is intended to simplify the choices presented to authors during document composition. Composition tools could use this to build a menu of likely starting points for a document. The Root attribute is purely a suggestion and does not require any action on the

part of the processor.

For information about the id attribute, see Section 2.8, "id Attributes".

Note that an element *must* declare a content model of some type, using the Model element, even if that content model is empty. Documentation (in the Doc element), non-DDML extensions (in the More element) and attribute declarations (using the AttGroup element) are optional.

Documentation about the element, additional extensions, content-model information, and attribute information are stored as sub-elements of the ElementDecl element. Documentation is covered in Section 2.7.1, Documentation Extensions. Additional extensions are covered in Section 2.7.2, Other Extensions. Content Models are covered in Section 2.3, Content Model Declarations, and attributes are covered in Section 2.4, Attribute Declarations.

2.3 Content Model Declarations

Content model declarations are made within the Model sub-element of the declaration for the element to which they apply.

Model elements may appear inside DocumentDef elements for reusability, documentation, and reference, but will need to be linked to particular element declarations through mechanisms not yet defined (most likely XLink). All content model declarations have an optional id attribute; for more infomation, see Section 2.8, "id Attributes".

The Model element holds the content model for an element.

```
<!ELEMENT Model (Doc?, More?, (Ref | Choice | Seq | Empty | Any | PCData
| Mixed))>
<!ATTLIST Model
    id ID #IMPLIED>
```

Model elements are pure containers. A Model element nested inside a Choice or Seq element can only contain Doc, More, Ref, Choice, and Seq elements.

2.3.1 Empty Content Model

The simplest content model is empty, which indicates that the parent element has no sub-elements and no character data content. The Empty element indicates that an element is empty.

```
<!ELEMENT Empty EMPTY>
<!ATTLIST Empty
    id ID #IMPLIED>
```

For example, to declare the Species element shown in the previous section empty, use the following DDML declaration:

```
<ElementDecl Name="Species">
  <Model>
    <Empty/>
  </Model>
</ElementDecl>
```

This would not allow the Species element to contain any text or sub-elements.

2.3.2 Any Content Model

The Any content model, which allows the element to contain parsed character data or any other elements as content, is equally simple:

```
<!ELEMENT Any EMPTY>
<!ATTLIST Any
    id ID #IMPLIED>
```

Using the Any content model is much like using the Empty content model. To declare that the Species element had a content model of any, use the following declaration:

```
<ElementDecl Name="Species">
  <Model>
    <Any/>
  </Model>
</ElementDecl>
```

This allows the Species element to contain text and any sub-elements an author desired.

2.3.3 PCData Content Model

The PCData content model, which allows the element to contain only parsed character data, is also represented by a single empty element.

```
<!ELEMENT PCData EMPTY>
<!ATTLIST PCData
    id ID #IMPLIED>
```

Using the PCData content model is much like using the Empty and Any content models. For example, to assign the Species element a content model of PCData, use the following declaration:

```
<ElementDecl Name="Species">
  <Model>
    <PCData/>
  </Model>
</ElementDecl>
```

This allows the Species element to contain text, but no sub-elements.

2.3.4 Reference Content Model

The Reference content model allows an element to specify other elements that it may contain. Ref elements identify the contained elements, as well as the frequency with which they appear:

```
<!ELEMENT Ref EMPTY>
<!-- Element references the name in an ElementDecl element -->
<!ATTLIST Ref
    Element       NMTOKEN #REQUIRED
    ElementNS     CDATA   #IMPLIED
    id            ID      #IMPLIED
    Frequency (Required | Optional | ZeroOrMore | OneOrMore) 'Required'>
```

A Model element may directly contain at most one Ref element. To define content models that

permit or require the use of more elements, use the Any, Mixed, Choice, or Sequence content models.

The Element and ElementNS attributes identify the contained element. These must match the values of the Name and ns attributes, respectively, of an ElementDecl element elsewhere in the DocumentDef document. The ElementNS attribute overrides the value of the same attribute on the DocumentDef, Mixed, Choice, or Seq element. For more information, see Section 3.2, "Namespaces of Elements and Attributes Being Defined."

The Frequency attribute controls the number of referenced elements that may occur.

To declare that the Species element must contain a single CommonName element, and nothing else, use the following declaration:

```
<ElementDecl Name="Species">
  <Model>
    <Ref Element="CommonName" Frequency="Required"/>
  </Model>
</ElementDecl>
```

This requires the Species element to contain a single CommonName element. To make the CommonName element optional - though it may still only appear once, set the Frequency attribute to 'Optional':

```
<ElementDecl Name="Species">
  <Model>
    <Ref Element="CommonName" Frequency="Optional"/>
  </Model>
</ElementDecl>
```

Optional is the equivalent of the ? occurrence indicator in XML 1.0 DTDs.

To require the Species element to contain at least one but possibly multiple CommonName elements, set the Frequency attribute to 'OneOrMore':

```
<ElementDecl Name="Species">
  <Model>
    <Ref Element="CommonName" Frequency="OneOrMore"/>
  </Model>
</ElementDecl>
```

OneOrMore is the equivalent of the + occurrence indicator in XML 1.0 DTDs.

Finally, to allow the Species element to contain any number (including zero) of CommonName elements, set the Frequency attribute to 'ZeroOrMore':

```
<ElementDecl Name="Species">
  <Model>
    <Ref Element="CommonName" Frequency="ZeroOrMore"/>
  </Model>
</ElementDecl>
```

ZeroOrMore is the equivalent of the * occurrence indicator in XML 1.0 DTDs.

2.3.5 Mixed Content Model

The mixed content model allows the unordered use of different element types and parsed character data. Content within an element declared as mixed can be parsed character data, one or more of the elements referenced by Ref elements nested in the Mixed element, or a mixture of both. The Mixed element in a DDML document must contain only Ref elements; there is no need to include a PCData element because this is inherent in the mixed content model.

```
<!ELEMENT Mixed (Ref+)>
<!ATTLIST Mixed
    ElementNS CDATA        #IMPLIED
    id        ID           #IMPLIED
    Frequency (ZeroOrMore) #FIXED   "ZeroOrMore">
```

To declare that the Species element may contain a mix of parsed character data, CommonName elements, LatinName elements, and PreferredFood elements in any order, use the following declaration:

```
<ElementDecl Name="Species">
  <Model>
    <Mixed>
      <Ref Element="CommonName"/>
      <Ref Element="LatinName"/>
      <Ref Element="PreferredFood"/>
    </Mixed>
  </Model>
</ElementDecl>
```

The DDML processor must ignore any frequency attributes in Ref elements that appear as subelements of the Mixed element.

For information about the ElementNS attribute, see Section 2.3.4, "Reference Content Model."

2.3.6 Choice Content Model

The Choice content model allows for either-or inclusions of elements and groups of elements. The Choice content model represents groups of element content possibilities and must contain at least two sub-elements. Situations where only one element is needed should use the Ref content model instead of Choice. The Choice element may indicate a frequency, allowing the content model defined by the Choice model to appear one, one or zero, one or more, or zero or more times.

```
<!-- A Choice must have two or more children -->
<!ELEMENT Choice ((Seq | Ref | Model), (Seq | Ref | Model)+)>
<!ATTLIST Choice>
    ElementNS CDATA #IMPLIED
    id        ID    #IMPLIED
    Frequency (Required | Optional | ZeroOrMore | OneOrMore) 'Required'>
```

The simplest Choice element will contain two Ref elements and a frequency attribute. By default, the Choice element's content model is required to appear once.

To declare that a Species element may contain either a common name or a Latin name, but not both, use the following declaration:

```
<ElementDecl Name="Species">
```

```
  <Model>
    <Choice Frequency="Required">
      <Ref Element="CommonName"/>
      <Ref Element="LatinName"/>
    </Choice>
  </Model>
</ElementDecl>
```

The Ref elements in a Choice element may also specify the frequency with which they appear, as may the Seq elements described in Section 2.3.7, "Sequence Content Model". The Choice element is the equivalent of the choice group (*element* | *element*) in XML 1.0 DTDs. The ordering of the sub-elements within an Choice element has no effect.

For information about the ElementNS attribute, see Section 2.3.4, "Reference Content Model."

2.3.7 Sequence Content Model

The Sequence content model allows for the sequential appearance of sub-elements. Elements, if they are required to appear, must appear in the order of the Choice and Ref sub-elements in the Seq element. The Seq element may also indicate a frequency, allowing the content model defined by the Seq model to appear one, one or zero, one or more, or zero or more times.

```
<!-- A Seq must have two or more children -->
<!ELEMENT Seq ((Choice | Ref | Model),(Choice | Ref | Model)+)>
<!ATTLIST Seq
    ElementNS CDATA #IMPLIED
    id        ID    #IMPLIED
    Frequency (Required | Optional | ZeroOrMore | OneOrMore) 'Required'>
```

The simplest Seq element will contain two Ref elements in the order in which they should appear and a frequency attribute. By default, the Seq element's content model is required to appear once.

To declare that the Species element requires a common name and a Latin name, in that order, use the following declaration:

```
<ElementDecl Name="Species">
  <Model>
    <Seq Frequency="Required">
      <Ref Element="CommonName"/>
      <Ref Element="LatinName"/>
    </Seq>
  </Model>
</ElementDecl>
```

The Ref elements in an Seq element may also specify the frequency with which they appear, as may the Choice elements. The Seq element is the equivalent of the sequence group (*element*, *element*) in XML 1.0 DTDs.

For information about the ElementNS attribute, see Section 2.3.4, "Reference Content Model."

2.4 Attribute Declarations

Attributes are declared with AttDef elements. The name, type, and default value (if any) of an attribute are defined with attributes of the AttDef element, as well as whether the attribute is

required. Values for enumerated types are provided by subelements.

AttGroup elements provide a container for multiple AttDef elements. If an AttGroup element is directly beneath an ElementDecl element, the contained AttDef elements apply to the element being declared. If an AttGroup element or AttDef element is directly beneath a DocumentDef element, the defined attributes are "global", and can only be used by reference.

```
<!ELEMENT AttGroup (Doc?, More?, (AttDef | AttGroup)*)>
<!ATTLIST AttGroup
    ns        CDATA   #IMPLIED
    prefix    NMTOKEN #IMPLIED
    id        ID      #IMPLIED>

<!ELEMENT AttDef (Doc?, More?, Enumeration?)>
<!ATTLIST AttDef
    Name      NMTOKEN      #REQUIRED
    ns        CDATA        #IMPLIED
    prefix    NMTOKEN      #IMPLIED
    Type      (CData    |
               ID       |
               IDRef    |
               IDRefs   |
               Entity   |
               Entities |
               Nmtoken  |
               Nmtokens |
               Notation |
               Enumerated) "CData"
    Required  (Yes | No)   "No"
    AttValue  CDATA        #IMPLIED
    id        ID           #IMPLIED>

<!ELEMENT Enumeration (Doc?, More?, EnumerationValue+)>
<!ATTLIST Enumeration
    id ID #IMPLIED>

<!ELEMENT EnumerationValue (Doc?, More?)>
<!ATTLIST EnumerationValue
    Value CDATA #REQUIRED>
```

The Name and ns attributes provide the name of the attribute and the URI of its namespace, respectively. The value of the Name attribute must match the NCName production in [Namespaces]; that is, it must begin with a letter or underscore and cannot include a colon. If an attribute uses the same namespace as the element to which it applies, its name must be unique within that element. If it uses a different namespace or does not apply to an element (that is, it is a global attribute), its name must be unique within the Global Attribute Partition of its namespace.

If an AttDef element is nested beneath an ElementDecl element, it applies to that element. The following example declares that the Species element has a Latin attribute:

```
<ElementDecl Name="Species">
  ...additionalElementInformation...
  <AttGroup>
    <AttDef Name="Latin" ...additionalInformation.../>
  </AttGroup>
```

```
</ElementDecl>
```

If an AttDef element is not nested beneath an ElementDecl element, it is a global attribute; a mechanism for using global attributes will be defined in a later version. The following example declares date, time, and location as global attributes:

```
<AttDef Name="date" ...additionalInformation.../>
...
<AttGroup>
  <AttDef Name="time" ...additionalInformation.../>
  <AttDef Name="location" ...additionalInformation.../>
</AttGroup>
```

The prefix attribute identifies the prefix that will be applied to the attribute during conversion to DTDs. For more information about prefixes and namespaces, see Section 3.2, "Namespaces of Elements and Attributes Being Defined."

For information about the id attribute, see Section 2.8, "id Attributes".

By default, attributes are assumed to contain character data (CData), not be required, and have no default value. Thus, the simplest attribute declaration requires only an attribute name:

```
<AttDef Name="Latin"/>
```

2.4.1 Attribute Types

DDML 1.0 provides equivalents for all of the XML 1.0 DTD attribute types. All of them are declared using attribute values within the AttDef element.

The CData attribute type is one of the most common, permitting an attribute to contain character data as defined by the XML 1.0 specification. If the Species element were to contain an attribute providing the Latin name of the species, the declaration could look like the following. (The Type attribute could actually be omitted in this case, as CData is the default type.)

```
<ElementDecl Name="Species">
  ...additionalElementInformation...
  <AttGroup>
    <AttDef Name="Latin" Type="CData"/>
  </AttGroup>
</ElementDecl>
```

This attribute would then be available for use in instances of the Species element:

```
<Species Latin="Passerina cyanea">...additionalContent...</Species>
```

The ID attribute type is used to uniquely identify elements in a document for application processing. IDRef and IDRefs attribute types are used to refer to a single ID value in the same document or multiple ID values in the same document, separated by whitespace, respectively. These attribute declarations must be used with the same constraints as apply to ID, IDREF, and IDREFS attribute types in XML 1.0.

The Entity and Entities attribute types identify the names of unparsed entities. The use of these attribute types must be made with the same constraints as apply to the ENTITY and ENTITIES attribute types in XML 1.0. The name of an unparsed entity identified by an Entity or Entities

attribute must match the Name attribute of an UnparsedEntity element elsewhere in the DDML document.

The Nmtoken and Nmtokens attribute types are used to declare attributes that must contain information conforming to the Nmtoken and Nmtokens productions in XML 1.0.

The Notation and Enumerated attribute types are more complex, requiring an Enumeration subelement, which in turn contains EnumerationValue subelements, to identify their possible content. These two declarations use similar syntax, but the allowed values of Notation declarations must match the Notations declared elsewhere in the DDML document.

If the status attribute of the Species element were to allow the values of extinct, endangered, protected, and non-threatened, an appropriate enumerated type declaration would look like:

```
<ElementDecl Name="Species">
  ...additionalElementInformation...
  <AttGroup>
    <AttDef Name="status" Type="Enumerated">
      <Enumeration>
        <EnumerationValue Value="extinct"/>
        <EnumerationValue Value="endangered"/>
        <EnumerationValue Value="protected"/>
        <EnumerationValue Value="non-threatened"/>
      </Enumeration>
    </AttDef>
  </AttGroup>
</ElementDecl>
```

A Species element created conforming to this declaration might look like:

```
<Species status="extinct">...additionalContentAboutDodos...</Species>
```

2.4.2 Attribute Defaults

DDML requires attribute declarations to provide information about the default value of a given attribute. DDML provides for the four cases supported by XML 1.0: #REQUIRED, #IMPLIED, #FIXED *AttValue*, and *AttValue*, though they are expressed as choices between required and not required with an optional default value. There may be only one default value declaration per attribute.

Required attributes (identified in XML 1.0 by #REQUIRED) are identified by assigning the value "Yes" to the Required attribute of an AttDef element and not assigning a value to the AttValue attribute. For instance, if the Latin attribute described above was required by the Species element, the AttDef element would contain a Required attribute with a value of "Yes":

```
<ElementDecl Name="Species">
  ...additionalElementInformation...
  <AttGroup>
    <AttDef Name="Latin" Required="Yes"/>
  </AttGroup>
</ElementDecl>
```

Optional attributes (identified in XML 1.0 by #IMPLIED) are identified assigning the value "No" to the Required attribute of an AttDef element and not assigning a value to the AttValue attribute. Implied indicates that there is no default value provided, and also that no value is required. If the

Latin attribute is optional, the AttDef element would contain a "No" value for the Required attribute. (Note that this is the default status and the Required declaration does not need to be made explicitly.)

```
<ElementDecl Name="Species">
  ...additionalElementInformation...
  <AttGroup>
    <AttDef Name="Latin" Required="No"/>
  </AttGroup>
</ElementDecl>
```

Fixed attributes (identified in XML 1.0 by #FIXED *AttValue*) are identified through the use of the Required attribute in combination with the AttValue attribute, which must contain the fixed value for the attribute. Attributes declared as fixed can only contain the declared value for that attribute. Fixed effectively hard codes attribute values into particular elements. If the Required attribute has a value of "Yes", and the AttValue attribute is present, the attribute value should be treated as a #FIXED value in XML 1.0.

For example, to declare a planet attribute for the Species element, a Required attribute given the value of "Yes" would identify the fixed nature of the attribute and the AttValue attribute would provide the value.

```
<ElementDecl Name="Species">
  ...additionalElementInformation...
  <AttGroup>
    <AttDef Name="planet" Required="Yes" AttValue="Earth"/>
  </AttGroup>
</ElementDecl>
```

Attributes may also be provided with a default value that may be overridden by other declarations. These default values are identified through the use of the AttValue attribute. The status attribute of species elements described above would be an appropriate target for such a default value, especially if most species being described fell into a particular category:

```
<ElementDecl Name="Species">
  ...additionalElementInformation...
  <AttGroup>
    <AttDef Name="status" Type="Enumerated" AttValue="non-threatened"/>
      <Enumeration>
        <EnumerationValue Value="extinct"/>
        <EnumerationValue Value="endangered"/>
        <EnumerationValue Value="protected"/>
        <EnumerationValue Value="non-threatened"/>
      </Enumeration>
    </AttDef>
  </AttGroup>
</ElementDecl>
```

Any default (required, fixed, etc.) may be used with any attribute type, though default values must always correspond to acceptable values for the attribute type.

2.4.3 Combinations of Types, Defaults, and Default Values

This notation also permits the declaration of certain attributes (IDs with defaults, for instance) that are prohibited by the standard XML 1.0 DTD syntax. Developers who use these combinations

should test that their documents will behave as expected in DTD-only environments as well as DDML environments. Additional processing of document instances may be necessary to produce normalized-for-DTD use documents if they included such attributes as default values. The attribute type should always be considered more important than its default values in DDML to DTD conversion.

The table below summarizes the possible combinations of DDML attribute defaults and their XML 1.0 DTD equivalents.

Required	AttValue	XML 1.0 Equivalent
Yes	<value>	#FIXED <value>
Yes	--	#REQUIRED
No	<value>	AttValue
No	--	#IMPLIED

(-- indicates an undeclared value)

2.5 Notation Declarations

Notation declarations are made with Notation elements nested in the DocumentDef element.

```
<!ELEMENT Notation (Doc?, More?)>
<!ATTLIST Notation
    Name          NMTOKEN #REQUIRED
    PubidLiteral  CDATA   #IMPLIED
    SystemLiteral CDATA   #IMPLIED
    id            ID      #IMPLIED>
```

The Name attribute provides the name of the notation. It must match the Name production in the XML 1.0 specification.

Notations may include a public identifier or a system literal, or both. DDML processors should ignore Notation elements that contain neither. Public identifiers and system literals should conform to the rules in Section 4.7 of the XML 1.0 Specification.

For information about the id attribute, see Section 2.8, "id Attributes".

2.6 Unparsed Entity Declarations

Unparsed entities are declared with UnparsedEntity elements nested in the DocumentDef element.

```
<!ELEMENT UnparsedEntity (Doc?, More?)>
<!ATTLIST UnparsedEntity
    Name          NMTOKEN #REQUIRED
    SystemLiteral CDATA   #REQUIRED
    PubidLiteral  CDATA   #IMPLIED
    Notation      NMTOKEN #REQUIRED
    id            ID      #IMPLIED>
```

The Name attribute provides the name of the unparsed entity. It must match the Name production

in the XML 1.0 specification and must be unique within the set of unparsed entities defined in the DDML document. The Notation attribute provides the name of a notation that gives the format of the unparsed entity. It must match the Name production in the XML 1.0 specification and must also match the Name attribute of a Notation element elsewhere in the DDML document.

UnparsedEntity elements must include a system literal and may include a public identifier. Public identifiers and system literals should conform to the rules in Section 4.7 of the XML 1.0 Specification.

For information about the id attribute, see Section 2.8, "id Attributes".

2.7 DDML Extensions

DDML provides areas in which DDML developers can provide supplemental information and metadata regarding DDML components in both human- and machine-readable formats. Human-readable information is provided through the use of a subset of HTML that conforms to XML syntax, while machine-readable information may be provided through the DDML:More element.

2.7.1 Documentation Extensions

Human-readable documentation for DDML documents should be provided using the Itsy Bitsy Teeny Weeny Simple Hypertext [IBTWSH]. This is an XML DTD which describes a subset of HTML 4.0 for embedded use within other XML DTDs. It is equivalent (within its scope) to -//W3C//DTD HTML 4.0 Transitional//EN. Documentation that uses portions of the IBTWSH format may be included in the DDML:Doc element, a subelement available to all declarations. The DDML:Doc element provides basic formatting options for DDML documentation.

```
<!ENTITY % ibtwsh SYSTEM "http://www.ccil.org/~cowan/XML/ibtwsh.dtd">
%ibtwsh;
<!ELEMENT DDML:Doc %struct.model;>
<!ATTLIST DDML:Doc
    xmlns CDATA #FIXED "">
```

Note that because DDML:Doc redefines the default namespace to support IBTWSH, the DDML: prefix must be used for DDML:Doc. Any element allowed in the IBTWSH struct.model set of elements (A, ABBR, ACRONYM, ADDRESS, BIG, BLOCKQUOTE, BR, CITE, CODE, DFN, DIR, DIV, DL, EM, H1, H2, H3, HR, KBD, OL, P, PRE, SAMP, SMALL, SPAN, STRONG, UL, VAR, XML) may be used in the DDML:Doc element. To preserve compatibility with HTML, IBTWSH does not use namespaces.

DDML applications should ignore all DDML declarations (i.e., elements prefixed with DDML: or another appropriate DDML prefix) within a DDML:Doc element. (The XML element of IBTWSH allows an ANY content model.)

2.7.2 Other Extensions

The DDML:More element provides an area which developers can use to create their own supplements to DDML, defining content types more tightly than is possible through DDML 1.0. The DDML:More element has a simple ANY content model, though DDML processors should ignore the appearance of any elements from the DDML namespace in this area.

```
<!ELEMENT DDML:More ANY>
<!ATTLIST DDML:More
    xmlns CDATA "">
```

Because DDML:More redefines the default namespace, the DDML: prefix must be used for DDML:More. Developers may override the blank value of the xmlns attribute to define their own default namespace for elements contained in the DDML:More element.

2.8 id Attributes

All DDML elements except EnumerationValue, More, and Doc have an optional id attribute. These attributes, if they appear, must have a unique value within the document. They have no defined use in DDML 1.0, but are included so that future extensions (possibly involving XLink) can uniquely identify elements in a DDML document.

3 DDML and Namespaces

DDML uses namespaces for its own operations and also supports schemas that take advantage of namespace facilities. DDML processors are responsible only for elements that use the DDML namespace appropriate to the version of DDML they are processing. Elements in other namespaces may be used in the DDML:Doc and DDML:More elements and passed to other applications as the processor deems appropriate.

DDML documents can be used by namespace-unaware applications provided the following conditions are met:

- The names of all DDML and IBTWSH elements except Doc and More must not be prefixed.
- Doc and More must always be prefixed with DDML:.
- The names of user-defined elements inside More must not collide with the names of any DDML or IBTWSH elements.
- The names of all elements declared in the DDML file must be unique, even if they are declared in different namespaces.
- The names of all attributes declared in the DDML file for a particular element must be unique, even if they are declared in different namespaces.

Note: This section is subject to change, even after the DDML specification is otherwise finalized. For more information, see Section 1.2, "Relation to Standards."

3.1 The DDML Namespace

The namespace for DDML 1.0 is built into the DDML DTD as the fixed default value of the xmlns and xmlns:DDML attributes of the DocumentDef element. The URL of the DDML namespace is a PURL (permanent URL) provided by the OCLC. PURLs use redirection to maintain a permanent address for sites that may change address. (For more information, see http://www.purl.org.) While DDML specification information may be stored at the location to which the PURL server redirects visitors, DDML applications should not rely on any of that information being there.

DDML:Doc and DDML:More must use the DDML: prefix because they declare other values for the default namespace. All other DDML elements may use the DDML: prefix if desired, but are not required to do so.

3.2 Namespaces of Elements and Attributes Being Defined

Each element or attribute defined in a DDML document can belong to its own namespace. The URI of this namespace is provided by the ns attribute and any documents that contain the element or attribute must use the same URI. For example, if the Species element is part of the http://www.taxonomy namespace, a DDML document might contain the following declaration:

```
<DocumentDef ns="http://www.taxonomy">
  <ElementDecl Name="Species">
    ...additionalElementInformation...
  </ElementDecl>
  ...
</DocumentDef>
```

The document that uses the Species element might contain:

```
<TAXON:Species xmlns:TAXON="http://www.taxonomy">
  ...additionalElementContent...
</TAXON:Species>
```

The ns attribute occurs on the DocumentDef, AttGroup, ElementDecl, and AttDef elements. Its value is inherited by lower-level elements, which can override it. In the simplest case, only the root DocumentDef element has an ns attribute. **It is strongly recommended that the ns attribute be used only on the DocumentDef element.**

Ref elements refer to an element defined elsewhere in the DDML document and which may belong to a different namespace. The name and namespace of the referenced element are provided with the Element and ElementNS attributes, respectively. The ElementNS attribute occurs on the DocumentDef, Mixed, Choice, and Seq elements, as well as the Ref element. Its value is inherited by lower-level elements, which can override it. In the simplest case, only the root DocumentDef element has an ElementNS attribute.

The following example shows how the Species element, defined in the http://taxonomy namespace, is referenced in the content model of the Animal element, defined in the http://zooinventory namespace.

```
<DocumentDef>
  <DocumentDef ns=""http://www.taxonomy">
    <ElementDecl Name="Species">
      ...additionalElementInformation...
    </ElementDecl>
    ...
  </DocumentDef>
  <DocumentDef ns=""http://www.zooinventory">
    <ElementDecl Name="Animal">
      <Model>
        <Seq>
          <Ref Element="Species" ElementNS="http://www.taxonomy" />
          <Ref Element="Quantity" />
          ...additionalReferences...
        </Seq>
      </Model>
    </ElementDecl>
  </DocumentDef>
</DocumentDef>
```

If no ns or ElementNS attribute applies to an element or attribute being defined or referenced, then that element or attribute is not considered to belong to any particular namespace. In particular, the element or attribute does not belong to the DDML namespace, nor does it belong to the current default namespace of any documents in which it is used, assuming a default namespace is defined.

For conversion to and from DTDs, DDML provides prefix attributes, which declares the namespace prefixes used in element and attribute declarations in DTDs. This allows documents and their associated DDML documents to track the same namespace using different prefixes if necessary. DDML-to-DTD converters should use the prefix attribute of a DocumentDef, ElementDecl, AttGroup, or AttDef element when creating DTD element and attribute declarations. DTD-to-DDML converters should use the prefixes assigned in the DTD and request further information about the 'real' namespace for use in the ns attribute. This may be accomplished by parsing a sample document instance, or by direct input from the person doing the conversion.

4 DDML Documents and DTDs

A DDML document is related to two different DTDs: the DTD of the DDML document itself and the DTD of the document described by the DDML document. This section discusses the relationship of DDML documents to these DTDs and describes what conversions are possible between the DDML document and the latter DTD. There is no requirement that either DTD actually exist.

4.1 DTDs in DDML Documents

A DDML document may include a DTD as an internal subset, external subset, or both. If included, the Name in the DOCTYPE statement must be DocumentDef and the DTD must include all of the markup declarations in Appendix C, "DDML DTD." It may also include additional markup declarations, such as declarations of elements to be used under the More element. However, these declarations must not override any of the declarations from Appendix C.

The main reason to include a DTD in a DDML document is so a DDML-unaware XML parser can supply default attribute values and determine the system and public identifiers of notations and unparsed general entities. Default attribute values are used in the DDML DTD defined in Appendix C. Notations and unparsed general entities can be used by user-defined elements under the More element.

Secondary reasons for including a DTD in a DDML document are to declare parsed entities (see Section 5.3.1, "Parsed Entities in DDML Documents") and to allow the document to be validated by DDML-unaware software.

4.2 DTDs in Documents Described by DDML Documents

A document described by a DDML document may include a DTD as well as processing instructions that refer to DDML documents (see Section 5.1.1, "DDML Processing Instruction"). This DTD can describe the same information as the DDML documents as well as additional information.

Reasons to include a DTD in a document described by a DDML document include:

- To allow a DDML-unaware XML parser to supply default attribute values and determine the system and public identifiers of notations and unparsed general entities.
- To allow the document to be used with both DDML-aware and -unaware software.
- To define the root element in the document.
- To declare parsed general entities.

If an XML document includes both a DTD and processing instructions that refer to DDML documents, it is the responsibility of the document author to ensure that the information common to both is the same. If the common information is different, it might not be possible to use the document with both DDML-aware and -unaware software. For example, it might not be possible to

validate the document against both the DTD and the DDML documents.

If a DDML processor is built on top of an XML parser, the DDML processor is not required to process the DTD of the XML document. If a DDML processor also functions as an XML parser, it is required to process the DTD only to the extent required of a non-validating parser.

4.3 Converting Between DDML Documents and DTDs

Schema information can be converted between DDML documents and DTDs, although some information may be lost. Most logical information (such as element and attribute declarations) can be converted from DTDs to DDML documents, while some logical information (such as attribute declarations not assigned to elements) cannot be converted from DDML documents to DTDs. In general, physical information (such as parsed entity declarations and use, the order of declarations, and the distribution of declarations among different files) either cannot be converted or is converted only at the option of the converter.

Converters may include as many or as few comments in the output document as they choose and may place these at any (legal) locations they choose. In particular, converters are not limited to converting between DDML Doc elements and XML comments. For example, a converter might place the entire input document in one or more comments in the output document for documentation's sake or it might generate comments noting which structures it does not convert.

4.3.1 Converting DTDs to DDML Documents

The following DTD structures must be converted to the corresponding DDML structures:

- Element, attribute, notation, and unparsed entity declarations. The order of the resulting elements, including whether attribute declarations are placed inside element declarations, is the choice of the converter.
- Namespace prefixes in element and attribute declarations. These must be stripped from the element or attribute name and stored in the prefix attribute. The converter may prompt the user for the URI of the namespace to be stored in the corresponding ns or ElementNS attribute.

The following DTD structures may be converted to the corresponding DDML structures or discarded:

- Comments. These may be converted to Doc elements. The position of resulting Doc elements in the DDML document is the choice of the converter. For example, a converter might place comments in a Doc element inside the following element, attribute, notation, or unparsed entity declaration.
- Parameter entity declarations and use. These may be converted to parsed general entity declarations and use.

The following DTD structures cannot be converted to DDML structures because such structures do not exist:

- Duplicate attribute and unparsed entity declarations.
- Parsed general entity declarations and use.
- Conditional sections in external parameter entities.
- Processing instructions.

4.3.2 Converting DDML Documents to DTDs

The following DDML structures must be converted to the corresponding DTD structures:

- All information in element, notation, and unparsed entity declarations and attribute declarations that apply to a particular element, except as noted elsewhere. The order of the resulting declarations is the choice of the converter.
- Namespace prefixes declared in prefix attributes. These must be prepended to the element or attribute name.

The following DDML structures may be converted to the corresponding DTD structures or discarded:

- Doc elements. These may be converted to comments. The position of resulting comments in the DTD is the choice of the converter.
- Parsed entities declared in the DTD of the DDML document. These may be converted to parsed general entities or parameter entities as appropriate.

The following DDML structures cannot be converted to DTD structures because such structures do not exist:

- More elements, AttDef and AttGroup elements that do not apply to a particular element, and Model and Enumeration elements nested directly beneath a DocumentDef element.
- All id attributes, all attributes of the DocumentDef element except for prefix, all ns and ElementNS attributes, and the Root attribute of the ElementDecl element.
- Nesting of schema information provided by nested DDML elements.

5 Using DDML Documents

This section describes how to associate DDML documents with XML documents and suggests ways to use DDML documents.

5.1 Associating DDML Documents with XML Documents

A DDML document can define a class of XML documents in the same way a DTD defines a class of XML documents. A document declares that it conforms to a class by including the DDML processing instruction. A document fragment can declare that it conforms to a class by including a nested DDML element; this latter usage is experimental.

5.1.1 DDML Processing Instruction

The DDML processing instruction is similar to the SYSTEM declaration in a DOCTYPE statement. It states that the document conforms to the class of documents described by the DDML document. The processing instruction has the following form:

```
[1] DDMLPI ::= '<?DDML' S SystemID PubID? Version? S? '?>'

[2] SystemID ::= 'System' Eq SystemLiteral

[3] PubID ::= S 'Public' Eq PubidLiteral

[4] Version ::= S 'Version' Eq (' VersionNum ' | " VersionNum ")
```

where the productions S, Eq, SystemLiteral, PubidLiteral, and VersionNum are the same as in [XML]. VersionNum describes the version number of the DDML document and must match the

value of the Version attribute on the root DocumentDef element of the document. The rules for retrieving the DDML document are the same as those for retrieving external entities, as described in Section 4.2.2, "External Entities," of [XML], except that a DDML processor may choose not to retrieve the document if the version number specified by VersionNum is different from the version of DDML supported by the processor.

A DDML processing instruction must occur before the root element to be used; any DDML processing instructions that occur after the root element will be ignored.

An XML document may include multiple DDML processing instructions. The effect is as if a superior root DocumentDef element contains the root DocumentDef element of each DDML document. This allows a document to conform to elements in many existing DDML documents. For more information, see Section 5.3.5, "Reusing Element Declarations with Entities or Processing Instructions."

5.1.2 Inline DDML Elements (Non-Normative)

NOTE: Inline DDML elements are considered experimental and may change in the future.

In some applications it is useful to repeatedly change the schema of the XML document at run time.
For example, consider a system that continuously logs data in XML format. >From an XML standpoint, it is as if a root element was started when the system was started, all incoming information is nested beneath the root element, and the root element ends only when the system stops. For practical purposes, the root element might not actually exist.

If the system logs information from different sources, the format (schema) of the nested elements might be different for each source. DDML elements can be interspersed in this stream to describe the format of following information:

```
<Root>
  <DocumentDef>...schema #1...</DocumentDef>
  ...log information that conforms to schema #1...
  <DocumentDef>...schema #2...</DocumentDef>
  ...log information that conforms to schema #2...
  ...
</Root>
```

Because such use is not well-defined today, DDML processors that use inline DDML elements should follow these rules for the greatest chance of forward compatibility:

- The schema information in a DocumentDef element applies to all following elements at the same level until the next DocumentDef element at that level is encountered.
- The schema information in a DocumentDef element completely replaces the schema information in the previous DocumentDef element at the same level. That is, no partial replacement of schema information is allowed.

5.2 Validation

A DDML processor can validate an XML document against a DDML document; such a processor is called a validator. Because DDML does not support parsed entity declarations (see Section 5.3.2, "Entity Support in DDML,") this validation is slightly less comprehensive than that defined in [XML]. Validators must enforce all Validity Constraints in [XML] except:

- Root Element Type
- Proper Declaration/PE Nesting
- Standalone Document Declaration
- Proper Group/PE Nesting
- Entity Declared

If the instance document contains a DOCTYPE statement and the validator can discover the root element type declared there (for example, the validator parses the document itself or the parser informs it of the root element type), then the validator must enforce the Root Element Type Validity Constraint as well.

Validators, like all DDML processors, are not required to parse XML documents.

5.3 Suggested Uses of DDML Documents

(Non-Normative)

The following sections suggest possible uses of DDML documents. They are not binding on DDML processors or documents.

5.3.1 Parsed Entities in DDML Documents

Parsed general entities are used in DDML documents for the same reasons they are used in XML documents: to distribute documents across multiple files, to enable multiple character encodings, to act as text substitution macros, and so on. They can also be used in a manner similar to parameter entities in a DTD. For example, suppose the DTD contains the following declaration:

```
<!ENTITY latinattribute "<AttDef Name='Latin' Type='CData'
Required='No'/">
```

This can be used in the content of the DDML document to add the Latin attribute to an element:

```
<ElementDecl Name="Species">
  ...additionalElementInformation...
  <AttGroup>
    &latinattribute;
  </AttGroup>
</ElementDecl>
```

Because parameter entities are used only in the DTD, they offer no special advantages to DDML documents.

5.3.2 Entity Support in DDML

DDML does not directly support parsed entities. That is, there is no DDML element that can be used to define a parsed entity. The reason for this is that parsed entities are physical in nature. They describe how to physically construct a document: how to distribute a document across multiple files and what text substitutions to perform. This is orthogonal to the function of a schema, which is logical in nature: it describes the legal content of a document and how that content can be arranged. That a DTD can describe both the logical and physical structure of a document was viewed as a historical accident that DDML chose not to perpetuate.

One benefit of this is that it reduces the amount of information that must be duplicated in a DTD. Because DDML processors are expected to be built on top of DDML-unaware parsers, any parsed

entity declarations that might otherwise have been in a DDML document would also have been required in the DTD; without them, DDML-unaware parsers could not parse the instance documents. Such duplication of declarations is prone to error.

5.3.3 DTD Replacement

As was noted in Section 5.1, a DDML document can define a class of XML documents. In this respect, it fulfills some of the functions of a DTD. That is, a DDML processor can validate an XML document against a DDML document and a DDML-aware XML parser can retrieve information about the XML document, such as default attribute values and the system and public identifiers of notations and unparsed general entities.

5.3.4 Schema Repository

DDML documents are not required to define a particular class of XML documents. For example, a DDML document might consist of nothing but attribute definitions. In this manner, a DDML document can function as a repository for schema definitions, which can then be reused by other DDML documents. Note that while a DDML document that defines a class of XML documents can always act as a repository, the converse is not always true.

5.3.5 Reusing Element Declarations with Entities or Processing Instructions

Element declarations in one DDML document can be reused by referring to them in a Ref element in second DDML document. For example, suppose a DDML repository defines a FullName element:

```
<ElementDecl Name="FullName">
  <Model>
    <Seq>
      <Ref Element="LastName"/>
      <Ref Element="FirstName"/>
      <Ref Element="MiddleName" Frequency="ZeroOrMore"/>
    </Seq>
  </Model>
</ElementDecl>
```

The DDML document that describes Letter documents might include FullName by reference, where the first instance is the author of the letter and the second instance is the recipient:

```
<ElementDecl Name="Letter">
  <Model>
    <Seq>
      <Ref Element="FullName"/>
      <Ref Element="FullName"/>
      <Ref Element="Paragraph" Frequency="OneOrMore"/>
    </Seq>
  </Model>
</ElementDecl>
```

The referenced declaration can be resolved in one of two ways. First, the second DDML document can include the first, either by cutting and pasting or through an external parsed general entity. For example:

```
<!DOCTYPE DocumentDef [
<!ENTITY nameRepository SYSTEM "names.ddm">
```

```
]>
<DocumentDef>
  &nameRepository;
  ... other declarations ...
</DocumentDef>
```

Second, a Letter (instance) document that can included processing instructions that point to both DDML documents. For example:

```
<!DOCTYPE Letter>
<?DDML SystemID="names.ddm" ?>
<?DDML SystemID="letter.ddm" ?>
<Letter>
  ...
</Letter>
```

Note that including a DDML processing instruction in letter.ddm that points to names.ddm will not have the intended effect. Rather than including the names.ddm, this processing instruction states that letter.ddm (a DDML document) conforms to the elements declared names.ddm. This is unlikely to be true.

5.3.6 Reusing Schema Definitions through XLinks

In the future, it should be possible to reuse schema definitions in a DDML document through XLinks. Although the exact manner in which this works cannot be determined until the XLink and XPointer specifications are complete, the example from section 5.3.5 might be performed as follows:

```
<ElementDecl Name="Letter">
  <Model>
    <Seq>
      <Ref Element="FullName"/>
      <Ref Element="FullName"/>
      <Ref Element="Paragraph" Frequency="OneOrMore"/>
    </Seq>
  </Model>
</ElementDecl>

<ElementDecl xml:link="simple"
             href="names.ddm#id(FullName)"
             inline="true"
             show="replace"/>
```

The second ElementDecl element points to, and is replaced by, the ElementDecl element for the FullName element in names.ddm. This eliminates the need to include the names.ddm through cut-and-paste, an entity, or a processing instruction.

DDML has been designed with such linking in mind. It is partially because of this that the container elements AttGroup, Enumeration, and Model exist and can be directly or indirectly nested inside themselves. For example, a new AttGroup might be constructed by nesting multiple AttGroup elements inside it, each of which contains an XLink to an AttGroup in a different DDML document.

5.3.7 Authoring

DDML documents support authoring tools (editors) by providing human-readable documentation

and a template for legal document structures.

A typical editing session using a DDML-aware editor might proceed as follows:

1. The editor displays a list of available DDML documents. The user chooses a DDML document to use as a template.
2. The editor reads the chosen document and displays a list of elements for which the Root attribute has a value of Recommended. The user chooses a starting element.
3. The editor prompts the user for element content and attributes based on the information in the DDML document. When the user requests help about a particular structure, the editor retrieves it from the corresponding Doc element.
4. When the user is done, the editor saves the new document, using the file extension specified by the FileExtension attribute of the root DocumentDef element. At the start of this document, the editor inserts a DOCTYPE statement declaring the root element type and a DDML processing instruction declaring the template DDML document.

An editor can also support schema building and modification. For example, it might allow the user to construct a new DDML document from elements in existing DDML documents or add new elements to existing DDML documents.

5.3.8 General Schema Information

Because DDML documents can contain information about a class of documents, they can be used by tools that work with (as opposed to on) these documents. For example, a database tool might read a DDML document and construct a database schema or a programming tool might read a DDML document and create Java classes for each element. DDML documents can also be used as starting points for search engines, which can use them to construct query-by-example interfaces.

5.3.9 Custom Uses

The More element in DDML provides a way for users to customize their DDML documents. For example, subelements of the More element might be used to assign the data type (integer, date, string, etc.) of PCData elements or associate Java classes with elements.

NOTE: There are a number of existing proposals for data types in XML and it is hoped that the W3C (and therefore DDML) will adopt one of these in the future. For example, see [DCD] or [SOX].

Appendix A: References

[DCD]
Tim Bray, Charles Frankston, and Ashok Malhotra. Document Content Description for XML. 31 July 1998. See http://www.w3.org/TR/NOTE-dcd.

[IBTWSH]
John Cowan. Itsy Bitsy Teeny Weeny Simple Hypertext. See http://www.ccil.org/~cowan/XML/ ibtwsh.dtd.

[Namespaces]
Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. 17 Nov 1998. See http://www.w3.org/TR/1998/PR-xml-names-19981117.

[RDF Schemas]
Dan Brickley, R. V. Guha, and Andrew Layman. Resource Description Framework (RDF) Schema Specification. 14 August, 1998. See http://www.w3.org/TR/WD-rdf-schema.

[RFC 2046]
IETF (Internet Engineering Task Force). RFC 2046: Multipurpose Internet Mail Extensions

(MIME) Part Two: Media Types, ed. N. Freed and N. Borenstein. November, 1996. See http://www.isi.edu/in-notes/rfc2046.txt.

[RFC 2048]
IETF (Internet Engineering Task Force). RFC 2048: Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures, ed. N. Freed, J. Klensin, and J. Postel. November, 1996. See http://www.isi.edu/in-notes/rfc2048.txt.

[RFC 2119]
IETF (Internet Engineering Task Force). RFC 2119: Key words for use in RFCs to Indicate Requirement Levels, ed. Scott Bradner. 1997. See http://www.isi.edu/in-notes/rfc2119.txt.

[RFC 2376]
IETF (Internet Engineering Task Force). RFC 2376: XML Media Types, ed. E.J.Whitehead and Murata Makoto. July, 1998. See http://www.isi.edu/in-notes/rfc2376.txt.

[SOX]
Matt Fuchs, Murray Maloney, and Alex Milowski. Schema for Object-oriented XML. 1998. See http://www.w3.org/TR/NOTE-SOX.

[XML]
Tim Bray, Jean Paoli, and C.M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. 1998. See http://www.w3.org/TR/REC-xml.

[XML-Data]
Andrew Layman, et al. XML-Data. 5 Jan 1998. See http://www.w3.org/TR/1998/NOTE-XML-data.

[XML-DEV]
XML-DEV Mailing List, archived at http://www.lists.ic.ac.uk/hypermail/xml-dev/.

[XLink]
Eve Maler and Steve DeRose. XML Linking Language (XLink). 1998. See http://www.w3.org/TR/WD-xlink.

[XPointer]
Eve Maler and Steve DeRose. XML Pointer Language (XPointer). 1998. See http://www.w3.org/TR/WD-xptr.

Appendix B: Differences between DDML and XSchema

There are only two technical differences between DDML and XSchema:

- In DDML, the root element is named DocumentDef, not XSchema.
- In DDML, the processing instruction used to associate a DDML document with an XML document uses the keyword "DDML", not "XSchema".

Appendix C: DDML DTD

```
<!ELEMENT DocumentDef (Doc?, More?, (ElementDecl | Model | AttDef |
AttGroup | Notation | UnparsedEntity | Enumeration | DocumentDef)*)>
<!ATTLIST DocumentDef
    xmlns          CDATA   #FIXED   "http://www.purl.org/NET/ddml/v1"
    xmlns:DDML     CDATA   #FIXED   "http://www.purl.org/NET/ddml/v1"
    ns             CDATA   #IMPLIED
    ElementNS      CDATA   #IMPLIED
    prefix         NMTOKEN #IMPLIED
    Version        CDATA   #FIXED   "1.0"
    MimeType       CDATA            "application/xml"
    FileExtension  CDATA            "xml"
    id             ID      #IMPLIED>
```

```
<!ELEMENT ElementDecl (Doc?, More?, Model, AttGroup?)>
<!-- Name is the element name -->
<!ATTLIST ElementDecl
    Name   NMTOKEN #REQUIRED
    ns     CDATA   #IMPLIED
    prefix NMTOKEN #IMPLIED
    id     ID      #IMPLIED
    Root   (Recommended | Possible | Unlikely) "Possible">

<!ELEMENT Model (Doc?, More?, (Ref | Choice | Seq | Empty | Any | PCData
| Mixed))>
<!ATTLIST Model
    id ID #IMPLIED>

<!ELEMENT Empty EMPTY>
<!ATTLIST Empty
    id ID #IMPLIED>

<!ELEMENT Any EMPTY>
<!ATTLIST Any
    id ID #IMPLIED>

<!ELEMENT PCData EMPTY>
<!ATTLIST PCData
    id ID #IMPLIED>

<!ELEMENT Ref EMPTY>
<!-- Element references the name in an ElementDecl element -->
<!ATTLIST Ref
    Element       NMTOKEN #REQUIRED
    ElementNS     CDATA   #IMPLIED
    id            ID      #IMPLIED
    Frequency (Required | Optional | ZeroOrMore | OneOrMore) 'Required'>

<!ELEMENT Mixed (Ref+)>
<!ATTLIST Mixed
    ElementNS CDATA        #IMPLIED
    id        ID           #IMPLIED
    Frequency (ZeroOrMore) #FIXED   "ZeroOrMore">

<!-- A Choice must have two or more children -->
<!ELEMENT Choice ((Seq | Ref | Model), (Seq | Ref | Model)+)>
<!ATTLIST Choice>
    ElementNS CDATA #IMPLIED
    id        ID    #IMPLIED
    Frequency (Required | Optional | ZeroOrMore | OneOrMore) 'Required'>

<!-- A Seq must have two or more children -->
<!ELEMENT Seq ((Choice | Ref | Model),(Choice | Ref | Model)+)>
<!ATTLIST Seq
    ElementNS CDATA #IMPLIED
    id        ID    #IMPLIED
    Frequency (Required | Optional | ZeroOrMore | OneOrMore) 'Required'>

<!ELEMENT AttGroup (Doc?, More?, (AttDef | AttGroup)*)>
```

```
<!ATTLIST AttGroup
    ns        CDATA   #IMPLIED
    prefix    NMTOKEN #IMPLIED
    id        ID      #IMPLIED>

<!ELEMENT AttDef (Doc?, More?, Enumeration?)>
<!ATTLIST AttDef
    Name      NMTOKEN      #REQUIRED
    ns        CDATA        #IMPLIED
    prefix    NMTOKEN      #IMPLIED
    Type      (CData    |
               ID       |
               IDRef    |
               IDRefs   |
               Entity   |
               Entities |
               Nmtoken  |
               Nmtokens |
               Notation |
               Enumerated) "CData"
    Required  (Yes | No)   "No"
    AttValue  CDATA        #IMPLIED
    id        ID           #IMPLIED>

<!ELEMENT Enumeration (Doc?, More?, EnumerationValue+)>
<!ATTLIST Enumeration
    id ID #IMPLIED>

<!ELEMENT EnumerationValue (Doc?, More?)>
<!ATTLIST EnumerationValue
    Value CDATA #REQUIRED>

<!ELEMENT Notation (Doc?, More?)>
<!ATTLIST Notation
    Name          NMTOKEN #REQUIRED
    PubidLiteral  CDATA   #IMPLIED
    SystemLiteral CDATA   #IMPLIED
    id         ;    ID      #IMPLIED>

<!ELEMENT UnparsedEntity (Doc?, More?)>
<!ATTLIST UnparsedEntity
    Name          NMTOKEN #REQUIRED
    SystemLiteral CDATA   #REQUIRED
    PubidLiteral  CDATA   #IMPLIED
    Notation      NMTOKEN #REQUIRED
    id            ID      #IMPLIED>

<!ENTITY % ibtwsh SYSTEM "http://www.ccil.org/~cowan/XML/ibtwsh.dtd">
%ibtwsh;
<!ELEMENT DDML:Doc %struct.model;>
<!ATTLIST DDML:Doc
    xmlns CDATA #FIXED "">

<!ELEMENT DDML:More ANY>
<!ATTLIST DDML:More
```

```
    xmlns CDATA "">
```

Appendix D: DDML in DDML

```
<?xml version="1.0"?>

<!DOCTYPE "DocumentDef">

<DocumentDef FileExtension="ddm" prefix="">

  <ElementDecl Name="DocumentDef" Root="Recommended">
    <Model>
      <Seq>
        <Ref Element="Doc" Frequency="Optional"/>
        <Ref Element="More" Frequency="Optional"/>
        <Choice Frequency="ZeroOrMore">
          <Ref Element="ElementDecl"/>
          <Ref Element="Model"/>
          <Ref Element="AttDef"/>
          <Ref Element="AttGroup"/>
          <Ref Element="Notation"/>
          <Ref Element="UnparsedEntity"/>
          <Ref Element="Enumeration"/>
          <Ref Element="DocumentDef"/>
        </Choice>
      </Seq>
    </Model>
    <AttGroup>
      <AttDef Name="xmlns" Required="Yes" AttValue="http://www.purl.org/
NET/ddml/v1"/>
      <AttDef Name="DDML" prefix="xmlns" Required="Yes" AttValue="http://
www.purl.org/NET/ddml/v1"/>
      <AttDef Name="ns"/>
      <AttDef Name="ElementNS"/>
      <AttDef Name="prefix" Type="Nmtoken"/>
      <AttDef Name="Version" Required="Yes" AttValue="1.0"/>
      <AttDef Name="MimeType" AttValue="application/xml"/>
      <AttDef Name="FileExtension" AttValue="xml"/>
      <AttDef Name="id" Type="ID"/>
    </AttGroup>
  </ElementDecl>

  <ElementDecl Name="ElementDecl">
    <Model>
      <Seq>
        <Ref Element="Doc" Frequency="Optional"/>
        <Ref Element="More" Frequency="Optional"/>
        <Ref Element="Model"/>
        <Ref Element="AttGroup" Frequency="Optional"/>
      </Seq>
    </Model>
    <AttGroup>
      <AttDef Name="Name" Type="Nmtoken" Required="Yes"/>
      <AttDef Name="ns"/>
      <AttDef Name="prefix" Type="Nmtoken"/>
```

```
    <AttDef Name="id" Type="ID"/>
    <AttDef Name="Root" Type="Enumerated" AttValue="Possible">
      <Enumeration>
        <EnumerationValue Value="Recommended"/>
        <EnumerationValue Value="Possible"/>
        <EnumerationValue Value="Unlikely"/>
      </Enumeration>
    </AttDef>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="Model">
  <Model>
    <Seq>
      <Ref Element="Doc" Frequency="Optional"/>
      <Ref Element="More" Frequency="Optional"/>
      <Choice>
        <Ref Element="Ref"/>
        <Ref Element="Choice"/>
        <Ref Element="Seq"/>
        <Ref Element="Empty"/>
        <Ref Element="Any"/>
        <Ref Element="PCData"/>
        <Ref Element="Mixed"/>
      </Choice>
    </Seq>
  </Model>
  <AttGroup>
    <AttDef Name="id" Type="ID"/>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="Empty">
  <Model>
    <Empty/>
  </Model>
  <AttGroup>
    <AttDef Name="id" Type="ID"/>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="Any">
  <Model>
    <Empty/>
  </Model>
  <AttGroup>
    <AttDef Name="id" Type="ID"/>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="PCData">
  <Model>
    <Empty/>
  </Model>
  <AttGroup>
    <AttDef Name="id" Type="ID"/>
```

```
      </AttGroup>
    </ElementDecl>

    <ElementDecl Name="Ref">
      <Model>
        <Empty/>
      </Model>
      <AttGroup>
        <AttDef Name="Element" Type="Nmtoken" Required="Yes"/>
        <AttDef Name="ElementNS"/>
        <AttDef Name="id" Type="ID"/>
        <AttDef Name="Frequency" Type="Enumerated" AttValue="Required">
          <Enumeration>
            <EnumerationValue Value="Required"/>
            <EnumerationValue Value="Optional"/>
            <EnumerationValue Value="ZeroOrMore"/>
            <EnumerationValue Value="OneOrMore"/>
          </Enumeration>
        </AttDef>
      </AttGroup>
    </ElementDecl>

    <ElementDecl Name="Mixed">
      <Model>
        <Ref Element="Ref" Frequency="OneOrMore"/>
      </Model>
      <AttGroup>
        <AttDef Name="ElementNS"/>
        <AttDef Name="id" Type="ID"/>
        <AttDef Name="Frequency" Type="Enumerated" Required="Yes"
  AttValue="ZeroOrMore">
          <Enumeration>
            <EnumerationValue Value="ZeroOrMore"/>
          </Enumeration>
        </AttDef>
      </AttGroup>
    </ElementDecl>

    <ElementDecl Name="Choice">
      <Model>
        <Seq>
          <Choice>
            <Ref Element="Seq"/>
            <Ref Element="Ref"/>
            <Ref Element="Model"/>
          </Choice>
          <Choice Frequency="OneOrMore">
            <Ref Element="Seq"/>
            <Ref Element="Ref"/>
            <Ref Element="Model"/>
          </Choice>
        </Seq>
      </Model>
      <AttGroup>
        <AttDef Name="ElementNS"/>
        <AttDef Name="id" Type="ID"/>
```

```
      <AttDef Name="Frequency" Type="Enumerated" AttValue="Required">
        <Enumeration>
          <EnumerationValue Value="Required"/>
          <EnumerationValue Value="Optional"/>
          <EnumerationValue Value="ZeroOrMore"/>
          <EnumerationValue Value="OneOrMore"/>
        </Enumeration>
      </AttDef>
    </AttGroup>
  </ElementDecl>

  <ElementDecl Name="Seq">
    <Model>
      <Seq>
        <Choice>
          <Ref Element="Choice"/>
          <Ref Element="Ref"/>
          <Ref Element="Model"/>
        </Choice>
        <Choice Frequency="OneOrMore">
          <Ref Element="Choice"/>
          <Ref Element="Ref"/>
          <Ref Element="Model"/>
        </Choice>
      </Seq>
    </Model>
    <AttGroup>
      <AttDef Name="ElementNS"/>
      <AttDef Name="id" Type="ID"/>
      <AttDef Name="Frequency" Type="Enumerated" AttValue="Required">
        <Enumeration>
          <EnumerationValue Value="Required"/>
          <EnumerationValue Value="Optional"/>
          <EnumerationValue Value="ZeroOrMore"/>
          <EnumerationValue Value="OneOrMore"/>
        </Enumeration>
      </AttDef>
    </AttGroup>
  </ElementDecl>

  <ElementDecl Name="AttGroup">
    <Model>
      <Seq>
        <Ref Element="Doc" Frequency="Optional"/>
        <Ref Element="More" Frequency="Optional"/>
        <Choice Frequency="ZeroOrMore">
           <Ref Element="AttDef"/>
           <Ref Element="AttGroup"/>
        </Choice>
      </Seq>
    </Model>
    <AttGroup>
      <AttDef Name="ns"/>
      <AttDef Name="prefix" Type="Nmtoken"/>
      <AttDef Name="id" Type="ID"/>
    </AttGroup>
```

```
</ElementDecl>

<ElementDecl Name="AttDef">
  <Model>
    <Seq>
      <Ref Element="Doc" Frequency="Optional"/>
      <Ref Element="More" Frequency="Optional"/>
      <Ref Element="Enumeration" Frequency="Optional"/>
    </Seq>
  </Model>
  <AttGroup>
    <AttDef Name="Name" Type="Nmtoken" Required="Yes"/>
    <AttDef Name="ns"/>
    <AttDef Name="prefix" Type="Nmtoken"/>
    <AttDef Name="Type" Type="Enumerated" AttValue="CData">
      <Enumeration>
        <EnumerationValue Value="CData"/>
        <EnumerationValue Value="ID"/>
        <EnumerationValue Value="IDRef"/>
        <EnumerationValue Value="IDRefs"/>
        <EnumerationValue Value="Entity"/>
        <EnumerationValue Value="Entities"/>
        <EnumerationValue Value="Nmtoken"/>
        <EnumerationValue Value="Nmtokens"/>
        <EnumerationValue Value="Notation"/>
        <EnumerationValue Value="Enumerated"/>
      </Enumeration>
    </AttDef>
    <AttDef Name="Required" Type="Enumeration" AttValue="No">
      <Enumeration>
        <EnumerationValue Value="Yes"/>
        <EnumerationValue Value="No"/>
      </Enumeration>
    </AttDef>
    <AttDef Name="AttValue"/>
    <AttDef Name="id" Type="ID"/>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="Enumeration">
  <Model>
    <Seq>
      <Ref Element="Doc" Frequen cy="Optional"/>
      <Ref Element="More" Frequency="Optional"/>
      <Ref Element="EnumerationValue" Frequency="OneOrMore"/>
    </Seq>
  </Model>
  <AttGroup>
    <AttDef Name="id" Type="ID"/>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="EnumerationValue">
  <Model>
    <Seq>
      <Ref Element="Doc" Frequency="Optional"/>
```

```
      <Ref Element="More" Frequency="Optional"/>
    </Seq>
  </Model>
  <AttGroup>
    <AttDef Name="Value" Required="Yes"/>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="Notation">
  <Model>
    <Seq>
      <Ref Element="Doc" Frequency="Optional"/>
      <Ref Element="More" Frequency="Optional"/>
    </Seq>
  </Model>
  <AttGroup>
    <AttDef Name="Name" Type="Nmtoken" Required="Yes"/>
    <AttDef Name="PubidLiteral"/>
    <AttDef Name="SystemLiteral"/>
    <AttDef Name="id" Type="ID"/>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="UnparsedEntity">
  <Model>
    <Seq>
      <Ref Element="Doc" Frequency="Optional"/>
      <Ref Element="More" Frequency="Optional"/>
    </Seq>
  </Model>
  <AttGroup>
    <AttDef Name="Name" Type="Nmtoken" Required="Yes"/>
    <AttDef Name="SystemLiteral" Required="Yes"/>
    <AttDef Name="PubidLiteral"/>
    <AttDef Name="Notation" Type="Nmtoken" Required="Yes"/>
    <AttDef Name="id" Type="ID"/>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="Doc" prefix="DDML">
  <Model>
     <!-- The struct model from IBTWSH goes here.
          Defining IBTWSH in DDML is left as an
          exercise to the reader.           ;       -->
  </Model>
  <AttGroup>
    <AttDef Name="xmlns" Required="Yes" AttValue=""/>
  </AttGroup>
</ElementDecl>

<ElementDecl Name="More" prefix="DDML">
  <Model>
    <Any/>
  </Model>
  <AttGroup>
    <AttDef Name="xmlns" AttValue=""/>
```

```
    </AttGroup>
  </ElementDecl>

</DocumentDef>
```

Appendix E: Contributors

The DDML specification is the result of contributions from a large number of people on the XML-Dev mailing list [XML-DEV], coordinated by a smaller group of editors. We apologize if we have left any contributors off the list below.

Eric Albright
Jacek Ambroziak
James Anderson
Mark D. Anderson
Curt Arnold
Jack Bolles
Jon Bosak
Frank Boumphrey
Tim Bray
Dan Brickley
David Brownell
Marcus Carr
Steven Champeon
Robin Cover
Alain Deseine
David G. Durand
Lars Marius Garshol
Bryan Gilbert
Matthew Gertner
Dirk Gouders
Jeremy H. Griffith
Paul Haahr
Carl Hage
Guy Huard
Will Hunt
Rick Jelliffe
Parameshwor Karki
Michael Kay
Bill la Forge
Andrew Layman
Chris Maden
Murata Makoto
Murray Maloney
Sean McGrath
David Megginson
Kenneth J. Meltsner
Matt Mower
Peter Murray-Rust
Steven R. Newcomb
Thuy-Lin Nguyen
Simon North
Francis Norton
Gisli Olafsson
David Ornstein
Don Park
W.E. Perry
Paul Prescod
Liam Quin
Paul Rabin
Lisa Rein
David Rosenborg
James K. Tauber
Arjun Ray
Todd Ross
John Simpson
Toby Speight
Jarle Stabell
Jeni Tennison
Mark Tucker
Scott Vanderbilt
Stefan Wagner
Steve Withall
Akitoshi Yoshida

The Schematron

An XML Structure Validation Language using Patterns in Trees

The Schematron is a simple and powerful Structural Schema Language.
It is currently undergoing ISO standardization to become
`ISO/IEC 19757 - DSDL Document Schema Definition Language - Part 3: Rule-based validation - Schematron`

- News
- Overview
- Schematron 1.5 Specification (RDDL page)
- Tutorial
- Schematron Love-In mailing list
- Download Schematron 1.5 Tools
- Download Schematron 1.3 Tools and Schemas
- Other Information (This is the old main page)

(If you have have news about Schematron or similar schema languages, please email Rick Jelliffe)

News 2003

June

- Jing is a high-quality validation library from the famous James Clark, who was the technical lead on XML, editor of the Xpath and ISO DSSL specifications, co-inventor of the RELAX NG schema language, and developer of some of the most popular Open Source softtware libraries: sgmls, nsgmls (SP), xt, xp, jade. This library provides support for RELAX

NG (compact and full syntaxes), Schematron, WXS datatypes, NRL adn even WXS (through Xerces).

May

- Topologi announced Validating Proxy, which allows Schematron and WXS valdiation of incoming data to Web Services or web services.
- Rick Jelliffe announces beta versions of Schematron 1.6 implementation. The software is a new skeleton, a new preprocessor (before the skeleton), and a new http://www.topologi.com/resources/tmp/schematron-report-portable1-6.xsl"> report tool.

News 2002

September

- Schematron.Net is an open source, high-performance implementation of Schematron for the .NET platform by Daniel Cazzulino. This implementation is interesting because it does not use XSLT but C#. Daniel reports that it seems to be about 50% faster than the fastest XSLT engines.
- Topologi Collaborative Markup Editor version 1.0 released, with free evaluation downloads. The editor supports Schematron validation and phases, as well as embedded Schematron in RELAX NG and W3C XML Schema. From a validation POV it it also interesting because it supports "feasible" validation (of DTDs and RELAX NG schemas).

August

- Schematron scores better than any other schema language: results of study by respected authorities! *XML 2001 Schema Language Comparison Town Hall*
- Revised Schematron 1.5 specification corrects errors, provides roadmap for ISO Schematron.

June

- Topologi Schematron Validator updated, adding RELAX NG schemas and Schematron-embedded-in-RELAX NG.
- Eddie Robertsson announces XSLT tools for Embedded Schematron in Relax along similar lines to Embedded Schematron in W3C XML Schemas and explained in an article.

May

- Implementation of Schematron available for PHP at http://phpxmlclasses.sourceforge.net/
- Schematron moves closer to ISO standardization. See article on DSDL at www.dsdl.org/ and also another article
- Article Filling in the Gaps with Schematron on XML.COM by Bob DuCharme on Schematron.
- Schematron featured in three papers at XML Europe 2002 Conference in Barcelona: a comparison of Schematron and XCSL, a disussion on progressive validation, and a discussion on Schema Languages.

Before May

- Schematron used in Apache Cocoon project for validating XML forms. This is a Java implementation rather than using XSLT, and is retported to have excellent performance.
- Schematron-in-Relax supported added to Sun's Multi Schema Validator.

(Due shifts in countries and jobs, I was not able to maintain this list for about a year. I apologise for this, especially for people who had news - Rick Jelliffe)

News 2001

July

- XML Content Management System Using XSLT, Schematron, and Ant by Eric van der Vlist, Conference paper at O'Reilly Open Source Convention, XTech 2001: Cutting Edge XML.

June

- Release of Topologi Schematron Validator, a free Windows GUI-based tool for running Schematron validations over multiple files. Also supports DTDs and W3C XML Schema. Highly configurable, to allow experimentation and development of schemas.
- Part fo the same release is updated version of most of the publically-available Schematron schemas, and improved versions of the 1.5 implementation. These will be placed on the Schematron 1.5 website shortly.

May

- XML Schemas Best Practices (compiler: R. Costello) includes section on embedding Schematron into XML Schemas `appinfo` elements.
- Commerical product supporting Schematron: FourThought Inc's 4Suite Server allows Schematron validation (sorry, I missed announcing this at the right time.)
- This is again undated, byt nice to see Schematron correctly listed under "programming by contract" at Java Repository
- Rick Jelliffe speaks at XSE 01, the *XML Software Engineering* Workshop connected with the big ACM Software Engineering conference in Toronto. Subject: *Schematron versus XML Schemas*. Trip sponsored by Zuelke Engineering of Swizterland. Workshop program committee also includes Grady Booch, showing the growing interest in XML. (Paper to be put on line soon.)

April

- Schematron Java API from Anna Hovhannisian, Oliver Becker
- Schematron: validating XML using XSLT by Leigh Dodds, Conference paper from XSLT-UK Conference. Also at Dodd's site.
- Types and Data Formats: DTDs, Schema, Schematron by Alexander Nakhimovsky. Flier for conference paper at XML Devcon Spring 2001.

March

- Examplotron
- WSDL page by Simon Fell includes WSDL Schema in Schematron
- Schemas by Elliotte Rusty Harold, Conference paper from XMLOne London (but note I think the second assert in the song example should be something like `*[position()=1][self::TITLE]`.)
- XML Schemas Training Course by Allette Systems includes Schematron material
- Eddie Robertsson's XSD_Schtrn is a stylesheet for extracting schematron schema fragments embedded as appinfo in XML Schemas `<;appinfo>` elements *beta*.
- Miloslav Nic's zvonSchematron is an implementation of Schematron 1.5 and is accompanied by a nice test reporting system, see Text Output. *beta*

February

- Schematron under consideration for project to create A Simple Collaboration System which is directly inspired by Doug Englebart's ideas. Very interesting. K. Holman put them onto Schematron, it seems. Very

interesting to see where this goes.
- Schemarama
- XML Schemas, by Brad Perry, Course notes including good Schematron overview. Note also an assignment comparing DTD, XML Schemas and Schematron. Nice.
- Small note on Logical Inferences from Schematron Schemas by Rick
- skeleton1-5.xsl reference implementation page, featuring links to conformance report, basic API documentation, sample code and sample . BAT file. `schematron-report`, `schematron-message` and `schematron-basic` moved over from 1.3. and documentation updated.
- Good summary of exchanges on XML-DEV comparing rule-based and grammar-based schema approaches at XML.COM
- The Screamatron Torture Test are the results so far of Rick's attempts to get Schematron going on 12 different command-line XSLT implementations for Windows, and to make the code inoffensive. Result: MSXSL (MSXML3), SAXON and Oracle work fine. XT, Xalan, and Sablotron can work with certain caveats. (Note: this is not an exhaustive test of all XSLT or Schematron features, nor of all XSLT implementations. For example, FourThought's Python-based implementation, which is reported to work, has not been tested yet.)
- Start at API documentation for extending the skeleton1-5.xsl
- Testing continues with Unicorn (no), Xalan for C 1 (yes with workaround), Xalan for Java 2 (yes, with workaround).
- Alternative validation language idea Hook may interest some people

January

- Complete *beta* implementation of Schematron 1.5 by Rick Jelliffe and collaborators (merging code from existing implementations) available now.
    - Same version works with both namespace and non-namespace
    - Implements phases, diagnostics, inherited abstract rules, value-of.
    - Command-line options to select the active phase or turn off diagnostics.
    - Better compile-time error messages.
    - Uses Oliver Becker's architecture, so should be compatible with existing 1.3 *meta-stylesheets*.
    - Extends the architecture to make available more attributes.
    - Uses conservative subset of XSLT so is still compatible with XP (providing key() is not used in schema). Also tested on SAXON, Instant SAXON and Sablotron (OK but does not support import yet, so metastylesheets have to be merged by hand.) See Implementation Notes for diary of some issues.

  The Schematron 1.5 Schema for Schematron 1.5 has been upgraded to use diagnostics, phases and an abstract element to demonstrate the

usage of these. Existing 1.3 Schematron schemas should be compatible with the new 1.5. This is a beta, please report any problems and suggested fixes/enhancements.
- Accompanying this is an implementation of the conformance language. This is a *meta-stylesheet* for the skeleton, and is also a good example of how to use the skeleton. The conformance language is used to check whether an implementation works correctly on different versions of XSLT.
- Schematron 1.5 manual updated.
- *Conformance Language* for Schematron 1.5 implementations *draft* available now
- Schematron 1.5 *beta* now being tested with XT, SAXON, Instant SAXON and Sablotron. Due for this week.
- New page design for home page at Academia Sinica (i.e. what you are reading now) with rewrite.
- Namespace URI for Schematron now locates a RDDL Resource Directory.
- New DTD, W3C XML Schema schema and Schematron schema for Schematron 1.5.

For old news, see the old home page.

Overview

The Schematron differs in basic concept from other schema languages in that it **not based on grammars** but on finding **tree patterns** in the parsed document. This approach allows many kinds of structures to be represented which are inconvenient and difficult in grammar-based schema languages. If you know XPath or the XSLT expression language, you can start to use The Schematron immediately.

And it has free and open source implementations available. The Schematron is trivially simple to implement on top of XSLT and to customize. (There are also implementations in Python and Perl)

The Schematron allows you to develop and mix two kinds of schemas:

- *Report* elements allow you to diagnose which variant of a language you are dealing with.
- *Assert* elements allow you to confirm that the document conforms to a particular schema.

The Schematron is based on a simple action:

- First, **find** a context nodes in the document (typically an element) based on XPath path criteria;
- Then, **check** to see if some other XPath expressions are true, for each of those nodes.

The Schematron can be useful **in conjunction** with many grammar-based structure-validation languages: DTDs, XML Schemas, RELAX, TREX, etc. You can even embed a Schematron schema inside an XML Schema `<appinfo>` element!

Schematron's Six Basic Elements

There are only 6 basic elements in **Schematron 1.5** which makes it very easy to learn, especially if you already know XPaths. (There are others, but these mainly just help construct nice user interfaces for validators.) Here is the basic structure

- `<schema xmlns="http://www.ascc.net/xml/schematron" >` contains
  - optional `<title>` then
  - zero or more `<ns prefix="???" uri="???" />` giving the namespaces and prefixes used for the XPaths, then
  - several `<pattern>`, which contain
    - several `<rule context="???" >` where the `context` attribute is an XSLT `expression`, which contain mixed
      - `<assert test="???">` where the `test`attribute is an XPath `location`, and which contains rich text expressing the statement being asserted in plain language, and
      - `<report test="???">` where the `test`attribute is an XPath `location`, and which contains rich text expressing the fact to be reported in plain language.

So here is a very small example. It is a mini-schema for Schematron.

```
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>A Schematron Mini-Schema for Schematron</title>
  <ns prefix="sch" uri="http://www.ascc.net/xml/schematron">
  <pattern>
    <rule context="sch:schema">
      <assert test="sch:pattern"
      >A schema contains patterns.</assert>
      <assert test="sch:pattern/sch:rule[@context]"
```

```
      >A pattern is composed of rules.
      These rules should have context attributes.</assert>
      <assert test="sch:pattern/sch:rule/sch:assert[@test]
or sch:pattern/sch:rule/sch:report[@test]"
      >A rule is composed of assert and report statements.
      These rules should have a test attribute.</assert>
    </rule>
  </pattern>
</schema>
```

In that mini-schema, the `rule` element sets the context: the rule applies to any `sch:schema` element in a document. The rules say that there must be at least one child element `sch:pattern`, at least one child element `sch:pattern` with a child `sch:rule` with a `context` attribute, and at least one child element `sch:pattern` with a child `sch:rule` with a `sch:assert` or `sch:report`with a `test` attribute.

This is probably not the most useful schema: it only tells you what is wrong with an empty document rather than checking a full Schematron schema. However, it does show that there are many different kinds of schemas possible: some of them similar to DTDs and some of them very different. Schematron lets you perform many kinds of new validation!

Academia Sinica

XML Schema Part 1: Structures

W3C Recommendation 2 May 2001

This version:
http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
(in XML (with its own DTD, XSL stylesheet) and HTML), with separate provision of the schema and DTD for schemas described herein.
Latest version:
http://www.w3.org/TR/xmlschema-1/
Previous version:
http://www.w3.org/TR/2001/PR-xmlschema-1-20010330/
Editors:
Henry S. Thompson (University of Edinburgh) <ht@cogsci.ed.ac.uk>
David Beech (Oracle Corporation) <David.Beech@oracle.com>
Murray Maloney (for Commerce One) <murray@muzmo.com>
Noah Mendelsohn (Lotus Development Corporation) <Noah_Mendelsohn@lotus.com>

Abstract

XML Schema: Structures specifies the XML Schema definition language, which offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those which exploit the XML Namespace facility. The schema language, which is itself represented in XML 1.0 and uses namespaces, substantially reconstructs and considerably extends the capabilities found in XML 1.0 document type definitions (DTDs). This specification depends on *XML Schema Part 2: Datatypes*.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document has been produced by the W3C XML Schema Working Group as part of the W3C XML Activity. The goals of the XML Schema language are discussed in the XML Schema Requirements document. The authors of this document are the XML Schema WG members. Different parts of this specification have different editors.

This version of this document incorporates some editorial changes from earlier versions.

Please report errors in this document to www-xml-schema-comments@w3.org (archive). The list of known errors in this specification is available at http://www.w3.org/2001/05/xmlschema-errata.

The English version of this specification is the only normative version. Information about translations of this document is available at http://www.w3.org/2001/05/xmlschema-translations.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/.

Table of contents

Appendices

---

1 Introduction

This document sets out the structural part (*XML Schema: Structures*) of the XML Schema definition language.

Chapter 2 presents a Conceptual Framework (§2) for XML Schemas, including an introduction to the nature of XML Schemas and an introduction to the XML Schema abstract data model, along with other terminology used throughout this document.

Chapter 3, Schema Component Details (§3), specifies the precise semantics of each component of the abstract model, the representation of each component in XML, with reference to a DTD and XML Schema for an XML Schema document type, along with a detailed mapping between the elements and attribute vocabulary of this representation and the components and properties of the abstract model.

Chapter 4 presents Schemas and Namespaces: Access and Composition (§4), including the connection between documents and schemas, the import, inclusion and redefinition of declarations and definitions and the foundations of schema-validity assessment.

Chapter 5 discusses Schemas and Schema-validity Assessment (§5), including the overall approach to schema-validity assessment of documents, and responsibilities of schema-aware processors.

The normative appendices include a Schema for Schemas (normative) (§A) for the XML representation of schemas and References (normative) (§B).

The non-normative appendices include the DTD for Schemas (non-normative) (§G) and a Glossary (non-normative) (§F).

This document is primarily intended as a language definition reference. As such, although it contains a few examples, it is *not* primarily designed to serve as a motivating introduction to the design and its features, or as a tutorial for new users. Rather it presents a careful and fully explicit definition of that design, suitable for guiding implementations. For those in search of a step-by-step introduction to the design, the non-normative [XML Schema: Primer] is a much better starting point than this document.

1.1 Purpose

The purpose of *XML Schema: Structures* is to define the nature of XML schemas and their component parts, provide an inventory of XML markup constructs with which to represent schemas, and define the application of schemas to XML documents.

The purpose of an *XML Schema: Structures* schema is to define and describe a class of XML documents by using schema components to constrain and document the meaning, usage and relationships of their constituent parts: datatypes, elements and their content and attributes and their values. Schemas may also provide for the specification of additional document information, such as normalization and defaulting of attribute and element values. Schemas have facilities for self-documentation. Thus, *XML Schema: Structures* can be used to define, describe and catalogue XML vocabularies for classes of XML documents.

Any application that consumes well-formed XML can use the *XML Schema: Structures* formalism to express syntactic, structural and value constraints applicable to its document instances. The *XML Schema: Structures* formalism allows a useful level of constraint checking to be described and implemented for a wide spectrum of XML applications. However, the language defined by this specification does not attempt to provide *all* the facilities that might be needed by *any* application. Some applications may require constraint capabilities not expressible in this language, and so may need to perform their own additional validations.

1.2 Dependencies on Other Specifications

The definition of *XML Schema: Structures* depends on the following specifications: [XML-Infoset], [XML-Namespaces], [XPath], and [XML Schemas: Datatypes].

See Required Information Set Items and Properties (normative) (§D) for a tabulation of the information items and properties specified in [XML-Infoset] which this specification requires as a precondition to schema-aware processing.

1.3 Documentation Conventions and Terminology

The section introduces the highlighting and typography as used in this document to present technical material.

Special terms are defined at their point of introduction in the text. For example [Definition:] a **term** is something used with a special meaning. The definition is labeled as such and the term it defines is displayed in boldface. The end of the definition is not specially marked in the displayed or printed text. Uses of defined terms are links to their definitions, set off with middle dots, for instance ·term·.

Non-normative examples are set off in boxes and accompanied by a brief explanation:

Example

```
<schema targetNamespace="http://www.example.com/XMLSchema/1.0/mySchema">
```

And an explanation of the example.

The definition of each kind of schema component consists of a list of its properties and their contents, followed by descriptions of the semantics of the properties:

Schema Component: Example

{example property}
Definition of the property.

References to properties of schema components are links to the relevant definition as exemplified above, set off with curly braces, for instance {example property}.

The correspondence between an element information item which is part of the XML representation of a schema and one or more schema components is presented in a tableau which illustrates the element information item(s) involved. This is followed by a tabulation of the correspondence between properties of the component and properties of the information item. Where context may determine which of several different components may arise, several tabulations, one per context, are given. The property correspondences are normative, as are the illustrations of the XML representation element information items.

In the XML representation, bold-face attribute names (e.g. **count** below) indicate a required attribute information item, and the rest are optional. Where an attribute information item has an enumerated type

definition, the values are shown separated by vertical bars, as for `size` below; if there is a default value, it is shown following a colon. Where an attribute information item has a built-in simple type definition defined in [XML Schemas: Datatypes], a hyperlink to its definition therein is given.

The allowed content of the information item is shown as a grammar fragment, using the Kleene operators ?, * and +. Each element name therein is a hyperlink to its own illustration.

> **NOTE:** The illustrations are derived automatically from the Schema for Schemas (normative) (§A). In the case of apparent conflict, the Schema for Schemas (normative) (§A) takes precedence, as it, together with the ·Schema Representation Constraints·, provide the normative statement of the form of XML representations.

XML Representation Summary: `example` Element Information Item

```
<example
  count = integer
  size = (large | medium | small) : medium>
  Content: (all | any*)
</example>
```

Example Schema Component

Property	Representation
{example property}	Description of what the property corresponds to, e.g. the value of the `size` [attribute]

References to elements in the text are links to the relevant illustration as exemplified above, set off with angle brackets, for instance <example>.

References to properties of information items as defined in [XML-Infoset] are notated as links to the relevant section thereof, set off with square brackets, for example [children].

Properties which this specification defines for information items are introduced as follows:

PSVI Contributions for example information items

[new property]
The value the property gets.

References to properties of information items defined in this specification are notated as links to their introduction as exemplified above, set off with square brackets, for example [new property].

The following highlighting is used for non-normative commentary in this document:

> **NOTE:** General comments directed to all readers.

Following [XML 1.0 (Second Edition)], within normative prose in this specification, the words *may* and *must* are defined as follows:

may
Conforming documents and XML Schema-aware processors are permitted to but need not behave as described.

must
Conforming documents and XML Schema-aware processors are required to behave as described; otherwise they are in error.

Note however that this specification provides a definition of error and of conformant processors' responsibilities with respect to errors (see Schemas and Schema-validity Assessment (§5)) which is considerably more complex than that of [XML 1.0 (Second Edition)].

2 Conceptual Framework

This chapter gives an overview of *XML Schema: Structures* at the level of its abstract data model. Schema Component Details (§3) provides details on this model, including a normative representation in XML for the components of the model. Readers interested primarily in learning to write schema documents may wish to first read [XML Schema: Primer] for a tutorial introduction, and only then consult the sub-sections of Schema Component Details (§3) named *XML Representation of ...* for the details.

2.1 Overview of XML Schema

An XML Schema consists of components such as type definitions and element declarations. These can be used to assess the validity of well-formed element and attribute information items (as defined in [XML-Infoset]), and furthermore may specify augmentations to those items and their descendants. This augmentation makes explicit information which may have been implicit in the original document, such as normalized and/or default values for attributes and elements and the types of element and attribute information items.

Schema-validity assessment has two aspects:

1 determining local schema-validity, that is whether an element or attribute information item satisfies the constraints embodied in the relevant components of an XML Schema;
2 Synthesizing an overall validation outcome for the item, combining local schema-validity with the results of schema-validity assessments of its descendants, if any, and adding appropriate augmentations to the infoset to record this outcome.

Throughout this specification, [Definition:] the word **valid** and its derivatives are used to refer to clause 1 above, the determination of local schema-validity.

Throughout this specification, [Definition:] the word **assessment** is used to refer to the overall process of local validation, schema-validity assessment and infoset augmentation.

2.2 XML Schema Abstract Data Model

This specification builds on [XML 1.0 (Second Edition)] and [XML-Namespaces]. The concepts and definitions used herein regarding XML are framed at the abstract level of information items as defined in [XML-Infoset]. By definition, this use of the infoset provides *a priori* guarantees of well-formedness (as defined in [XML 1.0 (Second Edition)]) and namespace conformance (as defined in [XML-Namespaces]) for all candidates for ·assessment· and for all ·schema documents·.

Just as [XML 1.0 (Second Edition)] and [XML-Namespaces] can be described in terms of information items, XML Schemas can be described in terms of an abstract data model. In defining XML Schemas in terms of an abstract data model, this specification rigorously specifies the information which must be available to a conforming XML Schema processor. The abstract model for schemas is conceptual only, and does not mandate any particular implementation or representation of this information. To facilitate interoperation and sharing of schema information, a normative XML interchange format for schemas is

provided.

[Definition:] **Schema component** is the generic term for the building blocks that comprise the abstract data model of the schema. [Definition:] An **XML Schema** is a set of ·schema components·. There are 13 kinds of component in all, falling into three groups. The primary components, which may (type definitions) or must (element and attribute declarations) have names are as follows:

- Simple type definitions
- Complex type definitions
- Attribute declarations
- Element declarations

The secondary components, which must have names, are as follows:

- Attribute group definitions
- Identity-constraint definitions
- Model group definitions
- Notation declarations

Finally, the "helper" components provide small parts of other components; they are not independent of their context:

- Annotations
- Model groups
- Particles
- Wildcards
- Attribute Uses

During ·validation·, [Definition:] **declaration** components are associated by (qualified) name to information items being ·validated·.

On the other hand, [Definition:] **definition** components define internal schema components that can be used in other schema components.

[Definition:] Declarations and definitions may have and be identified by **name**s, which are NCNames as defined by [XML-Namespaces].

[Definition:] Several kinds of component have a **target namespace**, which is either ·absent· or a namespace name, also as defined by [XML-Namespaces]. The ·target namespace· serves to identify the namespace within which the association between the component and its name exists. In the case of declarations, this in turn determines the namespace name of, for example, the element information items it may ·validate·.

> **NOTE:** At the abstract level, there is no requirement that the components of a schema share a ·target namespace·. Any schema for use in ·assessment· of documents containing names from more than one namespace will of necessity include components with different ·target namespaces·. This contrasts with the situation at the level of the XML representation of components, in which each schema document contributes definitions and declarations to a single target namespace.

·Validation·, defined in detail in Schema Component Details (§3), is a relation between information items and schema components. For example, an attribute information item may ·validate· with respect to an attribute declaration, a list of element information items may ·validate· with respect to a content model, and so on. The following sections briefly introduce the kinds of components in the schema abstract data model, other major features of the abstract model, and how they contribute to ·validation·.

2.2.1 Type Definition Components

The abstract model provides two kinds of type definition component: simple and complex.

[Definition:] This specification uses the phrase **type definition** in cases where no distinction need be made between simple and complex types.

Type definitions form a hierarchy with a single root. The subsections below first describe characteristics of that hierarchy, then provide an introduction to simple and complex type definitions themselves.

2.2.1.1 Type Definition Hierarchy

[Definition:] Except for a distinguished ·ur-type definition·, every ·type definition· is, by construction, either a ·restriction· or an ·extension· of some other type definition. The graph of these relationships forms a tree known as the **Type Definition Hierarchy**.

[Definition:] A type definition whose declarations or facets are in a one-to-one relation with those of another specified type definition, with each in turn restricting the possibilities of the one it corresponds to, is said to be a **restriction**. The specific restrictions might include narrowed ranges or reduced alternatives. Members of a type, A, whose definition is a ·restriction· of the definition of another type, B, are always members of type B as well.

[Definition:] A complex type definition which allows element or attribute content in addition to that allowed by another specified type definition is said to be an **extension**.

[Definition:] A distinguished **ur-type definition** is present in each ·XML Schema·, serving as the root of the type definition hierarchy for that schema. The ur-type definition, whose name is **anyType**, has the unique characteristic that it can function as a complex or a simple type definition, according to context. Specifically, ·restrictions· of the ur-type definition can themselves be either simple or complex type definitions.

[Definition:] A type definition used as the basis for an ·extension· or ·restriction· is known as the **base type definition** of that definition.

2.2.1.2 Simple Type Definition

A simple type definition is a set of constraints on strings and information about the values they encode, applicable to the ·normalized value· of an attribute information item or of an element information item with no element children. Informally, it applies to the values of attributes and the text-only content of elements.

Each simple type definition, whether built-in (that is, defined in [XML Schemas: Datatypes]) or user-defined, is a ·restriction· of some particular simple ·base type definition·. For the built-in primitive types, this is the simple version of the ·ur-type definition·, whose name is **anySimpleType**. This is in turn understood to be a restriction of the ·ur-type definition·. Simple types may also be defined whose members are lists of items themselves constrained by some other simple type definition, or whose membership is the union of the memberships of some other simple type definitions. List and union simple type definitions are also understood as restrictions of the simple ·ur-type definition·.

For detailed information on simple type definitions, see Simple Type Definitions (§3.14) and [XML Schemas: Datatypes]. The latter also defines an extensive inventory of pre-defined simple types.

2.2.1.3 Complex Type Definition

A complex type definition is a set of attribute declarations and a content type, applicable to the [attributes] and [children] of an element information item respectively. The content type may require the [children] to contain neither element nor character information items (that is, to be empty), to be a string which belongs to a particular simple type or to contain a sequence of element information items which conforms to a

particular model group, with or without character information items as well.

Each complex type definition is either

- a restriction of a complex ·base type definition·

or

- an ·extension· of a simple or complex ·base type definition·

or

- a ·restriction· of the ·ur-type definition·.

A complex type which extends another does so by having additional content model particles at the end of the other definition's content model, or by having additional attribute declarations, or both.

> **NOTE:** This specification allows only appending, and not other kinds of extensions. This decision simplifies application processing required to cast instances from derived to base type. Future versions may allow more kinds of extension, requiring more complex transformations to effect casting.

For detailed information on complex type definitions, see Complex Type Definitions (§3.4).

2.2.2 Declaration Components

There are three kinds of declaration component: element, attribute, and notation. Each is described in a section below. Also included is a discussion of element substitution groups, which is a feature provided in conjunction with element declarations.

2.2.2.1 Element Declaration

An element declaration is an association of a name with a type definition, either simple or complex, an (optional) default value and a (possibly empty) set of identity-constraint definitions. The association is either global or scoped to a containing complex type definition. A top-level element declaration with name 'A' is broadly comparable to a pair of DTD declarations as follows, where the associated type definition fills in the ellipses:

```
<!ELEMENT A . . .>
<!ATTLIST A . . .>
```

Element declarations contribute to ·validation· as part of model group ·validation·, when their defaults and type components are checked against an element information item with a matching name and namespace, and by triggering identity-constraint definition ·validation·.

For detailed information on element declarations, see Element Declarations (§3.3).

2.2.2.2 Element Substitution Group

In XML 1.0, the name and content of an element must correspond exactly to the element type referenced in the corresponding content model.

[Definition:] Through the new mechanism of **element substitution groups**, XML Schemas provides a more powerful model supporting substitution of one named element for another. Any top-level element declaration can serve as the defining element, or head, for an element substitution group. Other top-level element declarations, regardless of target namespace, can be designated as members of the substitution group headed by this element. In a suitably enabled content model, a reference to the head ·validates· not

just the head itself, but elements corresponding to any member of the substitution group as well.

All such members must have type definitions which are either the same as the head's type definition or restrictions or extensions of it. Therefore, although the names of elements can vary widely as new namespaces and members of the substitution group are defined, the content of member elements is strictly limited according to the type definition of the substitution group head.

Note that element substitution groups are not represented as separate components. They are specified in the property values for element declarations (see Element Declarations (§3.3)).

2.2.2.3 Attribute Declaration

An attribute declaration is an association between a name and a simple type definition, together with occurrence information and (optionally) a default value. The association is either global, or local to its containing complex type definition. Attribute declarations contribute to ·validation· as part of complex type definition ·validation·, when their occurrence, defaults and type components are checked against an attribute information item with a matching name and namespace.

For detailed information on attribute declarations, see Attribute Declarations (§3.2).

2.2.2.4 Notation Declaration

A notation declaration is an association between a name and an identifier for a notation. For an attribute information item to be ·valid· with respect to a `NOTATION` simple type definition, its value must have been declared with a notation declaration.

For detailed information on notation declarations, see Notation Declarations (§3.12).

2.2.3 Model Group Components

The model group, particle, and wildcard components contribute to the portion of a complex type definition that controls an element information item's content.

2.2.3.1 Model Group

A model group is a constraint in the form of a grammar fragment that applies to lists of element information items. It consists of a list of particles, i.e. element declarations, wildcards and model groups. There are three varieties of model group:

- Sequence (the element information items match the particles in sequential order);
- Conjunction (the element information items match the particles, in any order);
- Disjunction (the element information items match one of the particles).

For detailed information on model groups, see Model Groups (§3.8).

2.2.3.2 Particle

A particle is a term in the grammar for element content, consisting of either an element declaration, a wildcard or a model group, together with occurrence constraints. Particles contribute to ·validation· as part of complex type definition ·validation·, when they allow anywhere from zero to many element information items or sequences thereof, depending on their contents and occurrence constraints.

[Definition:] A particle can be used in a complex type definition to constrain the ·validation· of the [children] of an element information item; such a particle is called a **content model**.

> **NOTE:** *XML Schema: Structures* ·content models· are similar to but more expressive than

[XML 1.0 (Second Edition)] content models; unlike [XML 1.0 (Second Edition)], *XML Schema: Structures* applies ·content models· to the ·validation· of both mixed and element-only content.

For detailed information on particles, see Particles (§3.9).

2.2.3.3 Attribute Use

An attribute use plays a role similar to that of a particle, but for attribute declarations: an attribute declaration within a complex type definition is embedded within an attribute use, which specifies whether the declaration requires or merely allows its attribute, and whether it has a default or fixed value.

2.2.3.4 Wildcard

A wildcard is a special kind of particle which matches element and attribute information items dependent on their namespace name, independently of their local names.

For detailed information on wildcards, see Wildcards (§3.10).

2.2.4 Identity-constraint Definition Components

An identity-constraint definition is an association between a name and one of several varieties of identity-constraint related to uniqueness and reference. All the varieties use [XPath] expressions to pick out sets of information items relative to particular target element information items which are unique, or a key, or a ·valid· reference, within a specified scope. An element information item is only ·valid· with respect to an element declaration with identity-constraint definitions if those definitions are all satisfied for all the descendants of that element information item which they pick out.

For detailed information on identity-constraint definitions, see Identity-constraint Definitions (§3.11).

2.2.5 Group Definition Components

There are two kinds of convenience definitions provided to enable the re-use of pieces of complex type definitions: model group definitions and attribute group definitions.

2.2.5.1 Model Group Definition

A model group definition is an association between a name and a model group, enabling re-use of the same model group in several complex type definitions.

For detailed information on model group definitions, see Model Group Definitions (§3.7).

2.2.5.2 Attribute Group Definition

An attribute group definition is an association between a name and a set of attribute declarations, enabling re-use of the same set in several complex type definitions.

For detailed information on attribute group definitions, see Attribute Group Definitions (§3.6).

2.2.6 Annotation Components

An annotation is information for human and/or mechanical consumers. The interpretation of such information is not defined in this specification.

For detailed information on annotations, see Annotations (§3.13).

2.3 Constraints and Validation Rules

The [XML 1.0 (Second Edition)] specification describes two kinds of constraints on XML documents: *well-formedness* and *validity* constraints. Informally, the well-formedness constraints are those imposed by the definition of XML itself (such as the rules for the use of the < and > characters and the rules for proper nesting of elements), while validity constraints are the further constraints on document structure provided by a particular DTD.

The preceding section focused on ·validation·, that is the constraints on information items which schema components supply. In fact however this specification provides four different kinds of normative statements about schema components, their representations in XML and their contribution to the ·validation· of information items:

Schema Component Constraint
: [Definition:] Constraints on the schema components themselves, i.e. conditions components must satisfy to be components at all. Located in the sixth sub-section of the per-component sections of Schema Component Details (§3) and tabulated in Schema Component Constraints (§C.4).

Schema Representation Constraint
: [Definition:] Constraints on the representation of schema components in XML beyond those which are expressed in Schema for Schemas (normative) (§A). Located in the third sub-section of the per-component sections of Schema Component Details (§3) and tabulated in Schema Representation Constraints (§C.3).

Validation Rules
: [Definition:] Contributions to ·validation· associated with schema components. Located in the fourth sub-section of the per-component sections of Schema Component Details (§3) and tabulated in Validation Rules (§C.1).

Schema Information Set Contribution
: [Definition:] Augmentations to post-schema-validation infosets expressed by schema components, which follow as a consequence of ·validation· and/or ·assessment·. Located in the fifth sub-section of the per-component sections of Schema Component Details (§3) and tabulated in Contributions to the post-schema-validation infoset (§C.2).

The last of these, schema information set contributions, are not as new as they might at first seem. XML 1.0 validation augments the XML 1.0 information set in similar ways, for example by providing values for attributes not present in instances, and by implicitly exploiting type information for normalization or access. (As an example of the latter case, consider the effect of `NMTOKENS` on attribute white space, and the semantics of `ID` and `IDREF`.) By including schema information set contributions, this specification makes explicit some features that XML 1.0 left implicit.

2.4 Conformance

This specification describes three levels of conformance for schema aware processors. The first is required of all processors. Support for the other two will depend on the application environments for which the processor is intended.

[Definition:] **Minimally conforming** processors must completely and correctly implement the ·Schema Component Constraints·, ·Validation Rules·, and ·Schema Information Set Contributions· contained in this specification.

[Definition:] ·Minimally conforming· processors which accept schemas represented in the form of XML documents as described in Layer 2: Schema Documents, Namespaces and Composition (§4.2) are additionally said to provide **conformance to the XML Representation of Schemas**. Such processors must, when processing schema documents, completely and correctly implement all ·Schema Representation Constraints· in this specification, and must adhere exactly to the specifications in Schema Component Details (§3) for mapping the contents of such documents to ·schema components· for use in ·validation· and ·assessment·.

> **NOTE:** By separating the conformance requirements relating to the concrete syntax of XML schema documents, this specification admits processors which use schemas stored in optimized binary representations, dynamically created schemas represented as programming language data structures, or implementations in which particular schemas are compiled into executable code such as C or Java. Such processors can be said to be ·minimally conforming· but not necessarily in ·conformance to the XML Representation of Schemas·.

[Definition:] **Fully conforming** processors are network-enabled processors which are not only both ·minimally conforming· and ·in conformance to the XML Representation of Schemas·, but which additionally must be capable of accessing schema documents from the World Wide Web according to Representation of Schemas on the World Wide Web (§2.7) and How schema definitions are located on the Web (§4.3.2). .

> **NOTE:** Although this specification provides just these three standard levels of conformance, it is anticipated that other conventions can be established in the future. For example, the World Wide Web Consortium is considering conventions for packaging on the Web a variety of resources relating to individual documents and namespaces. Should such developments lead to new conventions for representing schemas, or for accessing them on the Web, new levels of conformance can be established and named at that time. There is no need to modify or republish this specification to define such additional levels of conformance.

See Schemas and Namespaces: Access and Composition (§4) for a more detailed explanation of the mechanisms supporting these levels of conformance.

2.5 Names and Symbol Spaces

As discussed in XML Schema Abstract Data Model (§2.2), most schema components (may) have ·names·. If all such names were assigned from the same "pool", then it would be impossible to have, for example, a simple type definition and an element declaration both with the name "title" in a given ·target namespace·.

Therefore [Definition:] this specification introduces the term **symbol space** to denote a collection of names, each of which is unique with respect to the others. A symbol space is similar to the non-normative concept of namespace partition introduced in [XML-Namespaces]. There is a single distinct symbol space within a given ·target namespace· for each kind of definition and declaration component identified in XML Schema Abstract Data Model (§2.2), except that within a target namespace, simple type definitions and complex type definitions share a symbol space. Within a given symbol space, names are unique, but the same name may appear in more than one symbol space without conflict. For example, the same name can appear in both a type definition and an element declaration, without conflict or necessary relation between the two.

Locally scoped attribute and element declarations are special with regard to symbol spaces. Every complex type definition defines its own local attribute and element declaration symbol spaces, where these symbol spaces are distinct from each other and from any of the other symbol spaces. So, for example, two complex type definitions having the same target namespace can contain a local attribute declaration for the unqualified name "priority", or contain a local element declaration for the name "address", without conflict or necessary relation between the two.

2.6 Schema-Related Markup in Documents Being Validated

The XML representation of schema components uses a vocabulary identified by the namespace name `http://www.w3.org/2001/XMLSchema`. For brevity, the text and examples in this specification use

the prefix `xs:` to stand for this namespace; in practice, any prefix can be used.

XML Schema: Structures also defines several attributes for direct use in any XML documents. These attributes are in a different namespace, which has the namespace name `http://www.w3.org/2001/XMLSchema-instance`. For brevity, the text and examples in this specification use the prefix `xsi:` to stand for this latter namespace; in practice, any prefix can be used. All schema processors have appropriate attribute declarations for these attributes built in, see Attribute Declaration for the 'type' attribute (§3.2.7), Attribute Declaration for the 'nil' attribute (§3.2.7), Attribute Declaration for the 'schemaLocation' attribute (§3.2.7) and Attribute Declaration for the 'noNamespaceSchemaLocation' attribute (§3.2.7).

2.6.1 xsi:type

The Simple Type Definition (§2.2.1.2) or Complex Type Definition (§2.2.1.3) used in ·validation· of an element is usually determined by reference to the appropriate schema components. An element information item in an instance may, however, explicitly assert its type using the attribute `xsi:type`. The value of this attribute is a ·QName·; see QName Interpretation (§3.15.3) for the means by which the ·QName· is associated with a type definition.

2.6.2 xsi:nil

XML Schema: Structures introduces a mechanism for signaling that an element should be accepted as ·valid· when it has no content despite a content type which does not require or even necessarily allow empty content. An element may be ·valid· without content if it has the attribute `xsi:nil` with the value `true`. An element so labeled must be empty, but can carry attributes if permitted by the corresponding complex type.

2.6.3 xsi:schemaLocation, xsi:noNamespaceSchemaLocation

The `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes can be used in a document to provide hints as to the physical location of schema documents which may be used for ·assessment·. See How schema definitions are located on the Web (§4.3.2) for details on the use of these attributes.

2.7 Representation of Schemas on the World Wide Web

On the World Wide Web, schemas are conventionally represented as XML documents (preferably of MIME type `application/xml` or `text/xml`, but see clause 1.1 of Inclusion Constraints and Semantics (§4.2.1)), conforming to the specifications in Layer 2: Schema Documents, Namespaces and Composition (§4.2). For more information on the representation and use of schema documents on the World Wide Web see Standards for representation of schemas and retrieval of schema documents on the Web (§4.3.1) and How schema definitions are located on the Web (§4.3.2).

3 Schema Component Details

3.1 Introduction

The following sections provide full details on the composition of all schema components, together with their XML representations and their contributions to ·assessment·. Each section is devoted to a single component, with separate subsections for

1. properties: their values and significance
2. XML representation and the mapping to properties
3. constraints on representation
4. validation rules
5. post-schema-validation infoset contributions
6. constraints on the components themselves

The sub-sections immediately below introduce conventions and terminology used throughout the component sections.

3.1.1 Components and Properties

Components are defined in terms of their properties, and each property in turn is defined by giving its range, that is the values it may have. This can be understood as defining a schema as a labeled directed graph, where the root is a schema, every other vertex is a schema component or a literal (string, boolean, number) and every labeled edge is a property. The graph is *not* acyclic: multiple copies of components with the same name in the same ·symbol space· may not exist, so in some cases re-entrant chains of properties must exist. Equality of components for the purposes of this specification is always defined as equality of names (including target namespaces) within symbol spaces.

> **NOTE:** A schema and its components as defined in this chapter are an idealization of the information a schema-aware processor requires: implementations are not constrained in how they provide it. In particular, no implications about literal embedding versus indirection follow from the use below of language such as "properties . . . having . . . components as values".

[Definition:] Throughout this specification, the term **absent** is used as a distinguished property value denoting absence.

Any property not identified as optional is required to be present; optional properties which are not present are taken to have ·absent· as their value. Any property identified as a having a set, subset or list value may have an empty value unless this is explicitly ruled out: this is *not* the same as ·absent·. Any property value identified as a superset or subset of some set may be equal to that set, unless a proper superset or subset is explicitly called for. By 'string' in Part 1 of this specification is meant a sequence of ISO 10646 characters identified as legal XML characters in [XML 1.0 (Second Edition)].

3.1.2 XML Representations of Components

The principal purpose of *XML Schema: Structures* is to define a set of schema components that constrain the contents of instances and augment the information sets thereof. Although no external representation of schemas is required for this purpose, such representations will obviously be widely used. To provide for this in an appropriate and interoperable way, this specification provides a normative XML representation for schemas which makes provision for every kind of schema component. [Definition:] A document in this form (i.e. a <schema> element information item) is a **schema document**. For the schema document as a whole, and its constituents, the sections below define correspondences between element information items (with declarations in Schema for Schemas (normative) (§A) and DTD for Schemas (non-normative) (§G)) and schema components. All the element information items in the XML representation of a schema must be in the XML Schema namespace, that is their [namespace name] must be `http://www.w3.org/2001/XMLSchema`. Although a common way of creating the XML Infosets which are or contain ·schema documents· will be using an XML parser, this is not required: any mechanism which constructs conformant infosets as defined in [XML-Infoset] is a possible starting point.

Two aspects of the XML representations of components presented in the following sections are constant across them all:

1. All of them allow attributes qualified with namespace names other than the XML Schema namespace itself: these appear as annotations in the corresponding schema component;
2. All of them allow an <annotation> as their first child, for human-readable documentation and/or

machine-targeted information.

3.1.3 The Mapping between XML Representations and Components

For each kind of schema component there is a corresponding normative XML representation. The sections below describe the correspondences between the properties of each kind of schema component on the one hand and the properties of information items in that XML representation on the other, together with constraints on that representation above and beyond those implicit in the Schema for Schemas (normative) (§A).

The language used is as if the correspondences were mappings from XML representation to schema component, but the mapping in the other direction, and therefore the correspondence in the abstract, can always be constructed therefrom.

In discussing the mapping from XML representations to schema components below, the value of a component property is often determined by the value of an attribute information item, one of the [attributes] of an element information item. Since schema documents are constrained by the Schema for Schemas (normative) (§A), there is always a simple type definition associated with any such attribute information item. [Definition:] The phrase **actual value** is used to refer to the member of the value space of the simple type definition associated with an attribute information item which corresponds to its ·normalized value·. This will often be a string, but may also be an integer, a boolean, a URI reference, etc. This term is also occasionally used with respect to element or attribute information items in a document being ·validated·.

Many properties are identified below as having other schema components or sets of components as values. For the purposes of exposition, the definitions in this section assume that (unless the property is explicitly identified as optional) all such values are in fact present. When schema components are constructed from XML representations involving reference by name to other components, this assumption may be violated if one or more references cannot be resolved. This specification addresses the matter of missing components in a uniform manner, described in Missing Sub-components (§5.3): no mention of handling missing components will be found in the individual component descriptions below.

Forward reference to named definitions and declarations *is* allowed, both within and between ·schema documents·. By the time the component corresponding to an XML representation which contains a forward reference is actually needed for ·validation· an appropriately-named component may have become available to discharge the reference: see Schemas and Namespaces: Access and Composition (§4) for details.

3.1.4 White Space Normalization during Validation

Throughout this specification, [Definition:] the **initial value** of some attribute information item is the value of the [normalized value] property of that item. Similarly, the **initial value** of an element information item is the string composed of, in order, the [character code] of each character information item in the [children] of that element information item.

The above definition means that comments and processing instructions, even in the midst of text, are ignored for all ·validation· purposes.

[Definition:] The **normalized value** of an element or attribute information item is an ·initial value· whose white space, if any, has been normalized according to the value of the whiteSpace facet of the simple type definition used in its ·validation·:

preserve
: No normalization is done, the value is the ·normalized value·

replace
: All occurrences of `#x9` (tab), `#xA` (line feed) and `#xD` (carriage return) are replaced with `#x20`

(space).

collapse
Subsequent to the replacements specified above under **replace**, contiguous sequences of `#x20`s are collapsed to a single `#x20`, and initial and/or final `#x20`s are deleted.

There are three alternative validation rules which may supply the necessary background for the above: Attribute Locally Valid (§3.2.4) (clause 3), Element Locally Valid (Type) (§3.3.4) (clause 3.1.3) or Element Locally Valid (Complex Type) (§3.4.4) (clause 2.2).

These three levels of normalization correspond to the processing mandated in XML 1.0 for element content, CDATA attribute content and tokenized attributed content, respectively. See Attribute Value Normalization in [XML 1.0 (Second Edition)] for the precedent for **replace** and **collapse** for attributes. Extending this processing to element content is necessary to ensure a consistent ·validation· semantics for simple types, regardless of whether they are applied to attributes or elements. Performing it twice in the case of attributes whose [normalized value] has already been subject to replacement or collapse on the basis of information in a DTD is necessary to ensure consistent treatment of attributes regardless of the extent to which DTD-based information has been made use of during infoset construction.

> **NOTE:** Even when DTD-based information *has* been appealed to, and Attribute Value Normalization has taken place, the above definition of ·normalized value· may mean *further* normalization takes place, as for instance when character entity references in attribute values result in white space characters other than spaces in their ·initial value·s.

3.2 Attribute Declarations

Attribute declarations provide for:

- Local ·validation· of attribute information item values using a simple type definition;
- Specifying default or fixed values for attribute information items.

Example

```
<xs:attribute name="age" type="xs:positiveInteger" use="required"/>
```

The XML representation of an attribute declaration.

3.2.1 The Attribute Declaration Schema Component

The attribute declaration schema component has the following properties:

Schema Component: Attribute Declaration

{name}
An NCName as defined by [XML-Namespaces].

{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].

{type definition}
A simple type definition.

{scope}

Optional. Either *global* or a complex type definition.

{value constraint}

Optional. A pair consisting of a value and one of *default*, *fixed*.

{annotation}

Optional. An annotation.

The {name} property must match the local part of the names of attributes being ·validated·.

The value of the attribute must conform to the supplied {type definition}.

A non-·absent· value of the {target namespace} property provides for ·validation· of namespace-qualified attribute information items (which must be explicitly prefixed in the character-level form of XML documents). ·Absent· values of {target namespace} ·validate· unqualified (unprefixed) items.

A {scope} of *global* identifies attribute declarations available for use in complex type definitions throughout the schema. Locally scoped declarations are available for use only within the complex type definition identified by the {scope} property. This property is ·absent· in the case of declarations within attribute group definitions: their scope will be determined when they are used in the construction of complex type definitions.

{value constraint} reproduces the functions of XML 1.0 default and `#FIXED` attribute values. *default* specifies that the attribute is to appear unconditionally in the post-schema-validation infoset, with the supplied value used whenever the attribute is not actually present; *fixed* indicates that the attribute value if present must equal the supplied constraint value, and if absent receives the supplied value as for *default*. Note that it is *values* that are supplied and/or checked, not strings.

See Annotations (§3.13) for information on the role of the {annotation} property.

> **NOTE:** A more complete and formal presentation of the semantics of {name}, {target namespace} and {value constraint} is provided in conjunction with other aspects of complex type ·validation· (see Element Locally Valid (Complex Type) (§3.4.4).)

[XML-Infoset] distinguishes attributes with names such as `xmlns` or `xmlns:xsl` from ordinary attributes, identifying them as [namespace attributes]. Accordingly, it is unnecessary and in fact not possible for schemas to contain attribute declarations corresponding to such namespace declarations, see xmlns Not Allowed (§3.2.6). No means is provided in this specification to supply a default value for a namespace declaration.

3.2.2 XML Representation of Attribute Declaration Schema Components

The XML representation for an attribute declaration schema component is an <attribute> element information item. It specifies a simple type definition for an attribute either by reference or explicitly, and may provide default information. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `attribute` Element Information Item

```
<attribute
  default = string
  fixed = string
  form = (qualified | unqualified)
  id = ID
  name = NCName
  ref = QName
  type = QName
  use = (optional | prohibited | required) : optional
```

```
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleType?))
</attribute>
```

If the <attribute> element information item has <schema> as its parent, the corresponding schema component is as follows:

Attribute Declaration Schema Component

Property	Representation
{name}	The ·actual value· of the `name` [attribute]
{target namespace}	The ·actual value· of the `targetNamespace` [attribute] of the parent <schema> element information item, or ·absent· if there is none.
{type definition}	The simple type definition corresponding to the <simpleType> element information item in the [children], if present, otherwise the simple type definition ·resolved· to by the ·actual value· of the `type` [attribute], if present, otherwise the ·simple ur-type definition·.
{scope}	*global*.
{value constraint}	If there is a `default` or a `fixed` [attribute], then a pair consisting of the ·actual value· (with respect to the {type definition}) of that [attribute] and either *default* or *fixed*, as appropriate, otherwise ·absent·.
{annotation}	The annotation corresponding to the <annotation> element information item in the [children], if present, otherwise ·absent·.

otherwise if the <attribute> element information item has <complexType> or <attributeGroup> as an ancestor and the `ref` [attribute] is absent, it corresponds to an attribute use with properties as follows (unless `use='prohibited'`, in which case the item corresponds to nothing at all):

Attribute Use Schema Component

Property	Representation
{required}	*true* if the `use` [attribute] is present with ·actual value· `required`, otherwise *false*.
{attribute declaration}	See the Attribute Declaration mapping immediately below.
{value constraint}	If there is a `default` or a `fixed` [attribute], then a pair consisting of the ·actual value· (with respect to the {type definition} of the {attribute declaration}) of that [attribute] and either *default* or *fixed*, as appropriate, otherwise ·absent·.

Attribute Declaration Schema Component

Property	Representation
{name}	The ·actual value· of the `name` [attribute]
{target namespace}	If `form` is present and its ·actual value· is `qualified`, or if `form` is absent and the ·actual value· of `attributeFormDefault` on the <schema> ancestor is `qualified`, then the ·actual value· of the `targetNamespace` [attribute] of the parent <schema> element information item, or ·absent· if there is none, otherwise ·absent·.

{type definition}	The simple type definition corresponding to the <simpleType> element information item in the [children], if present, otherwise the simple type definition ·resolved· to by the ·actual value· of the `type` [attribute], if present, otherwise the ·simple ur-type definition·.
{scope}	If the <attribute> element information item has <complexType> as an ancestor, the complex definition corresponding to that item, otherwise (the <attribute> element information item is within an <attributeGroup> definition), ·absent·.
{value constraint}	·absent·.
{annotation}	The annotation corresponding to the <annotation> element information item in the [children], if present, otherwise ·absent·.

otherwise (the <attribute> element information item has <complexType> or <attributeGroup> as an ancestor and the `ref` [attribute] is present), it corresponds to an attribute use with properties as follows (unless `use='prohibited'`, in which case the item corresponds to nothing at all):

Attribute Use Schema Component

Property	**Representation**
{required}	*true* if the `use` [attribute] is present with ·actual value· `required`, otherwise *false*.
{attribute declaration}	The (top-level) attribute declaration ·resolved· to by the ·actual value· of the `ref` [attribute]
{value constraint}	If there is a `default` or a `fixed` [attribute], then a pair consisting of the ·actual value· (with respect to the {type definition} of the {attribute declaration}) of that [attribute] and either *default* or *fixed*, as appropriate, otherwise ·absent·.

Attribute declarations can appear at the top level of a schema document, or within complex type definitions, either as complete (local) declarations, or by reference to top-level declarations, or within attribute group definitions. For complete declarations, top-level or local, the `type` attribute is used when the declaration can use a built-in or pre-declared simple type definition. Otherwise an anonymous <simpleType> is provided inline.

The default when no simple type definition is referenced or provided is the simple ·ur-type definition·, which imposes no constraints at all.

Attribute information items ·validated· by a top-level declaration must be qualified with the {target namespace} of that declaration (if this is ·absent·, the item must be unqualified). Control over whether attribute information items ·validated· by a local declaration must be similarly qualified or not is provided by the `form` [attribute], whose default is provided by the `attributeFormDefault` [attribute] on the enclosing <schema>, via its determination of {target namespace}.

The names for top-level attribute declarations are in their own ·symbol space·. The names of locally-scoped attribute declarations reside in symbol spaces local to the type definition which contains them.

3.2.3 Constraints on XML Representations of Attribute Declarations

Schema Representation Constraint: Attribute Declaration Representation OK

In addition to the conditions imposed on <attribute> element information items by the schema for schemas, **all** of the following must be true:

1 `default` and `fixed` must not both be present.

2 If `default` and `use` are both present, `use` must have the ·actual value· `optional`.
3 If the item's parent is not <schema>, then **all** of the following must be true:
3.1 One of `ref` or `name` must be present, but not both.
3.2 If `ref` is present, then all of <simpleType>, `form` and `type` must be absent.
4 `type` and <simpleType> must not both be present.
5 The corresponding attribute declaration must satisfy the conditions set out in Constraints on Attribute Declaration Schema Components (§3.2.6).

3.2.4 Attribute Declaration Validation Rules

Validation Rule: Attribute Locally Valid

For an attribute information item to be locally ·valid· with respect to an attribute declaration **all** of the following must be true:
1 The declaration must not be ·absent· (see Missing Sub-components (§5.3) for how this can fail to be the case).
2 Its {type definition} must not be absent.
3 The item's ·normalized value· must be locally ·valid· with respect to that {type definition} as per String Valid (§3.14.4).
4 The item's ·actual value· must match the value of the {value constraint}, if it is present and *fixed*.

Validation Rule: Schema-Validity Assessment (Attribute)

The schema-validity assessment of an attribute information item depends on its ·validation· alone.

[Definition:] During ·validation·, associations between element and attribute information items among the [children] and [attributes] on the one hand, and element and attribute declarations on the other, are established as a side-effect. Such declarations are called the **context-determined declarations**. See clause 3.1 (in Element Locally Valid (Complex Type) (§3.4.4)) for attribute declarations, clause 2 (in Element Sequence Locally Valid (Particle) (§3.9.4)) for element declarations.

For an attribute information item's schema-validity to have been assessed **all** of the following must be true:
1 A non-·absent· attribute declaration must be known for it, namely **one** of the following:
1.1 A declaration which has been established as its ·context-determined declaration·;
1.2 A declaration resolved to by its [local name] and [namespace name] as defined by QName resolution (Instance) (§3.15.4), provided its ·context-determined declaration· is not *skip*.
2 Its ·validity· with respect to that declaration must have been evaluated as per Attribute Locally Valid (§3.2.4).
3 Both clause 1 and clause 2 of Attribute Locally Valid (§3.2.4) must be satisfied.

[Definition:] For attributes, there is no difference between assessment and strict assessment, so if the above holds, the attribute information item has been **strictly assessed**.

3.2.5 Attribute Declaration Information Set Contributions

Schema Information Set Contribution: Assessment Outcome (Attribute)

If the schema-validity of an attribute information item has been assessed as per Schema-Validity Assessment (Attribute) (§3.2.4), then in the post-schema-validation infoset it has properties as follows:

PSVI Contributions for attribute information items
`[validation context]` The nearest ancestor element information item with a [schema information] property. **`[validity]`** The appropriate **case** among the following: 1 **If** it was ·strictly assessed·, **then** the appropriate **case** among the following: 1.1 **If** it was ·valid· as defined by Attribute Locally Valid (§3.2.4), **then** *valid*;

1.2 **otherwise** *invalid*.
2 **otherwise** *notKnown*.

[validation attempted]
The appropriate **case** among the following:
1 **If** it was ·strictly assessed·, **then** *full*;
2 **otherwise** *none*.

[schema specified]
infoset. See Attribute Default Value (§3.4.5) for the other possible value.

Schema Information Set Contribution: Validation Failure (Attribute)
If the local ·validity·, as defined by Attribute Locally Valid (§3.2.4) above, of an attribute information item has been assessed, in the post-schema-validation infoset the item has a property:

PSVI Contributions for attribute information items

[schema error code]
The appropriate **case** among the following:
1 **If** the item is not ·valid·, **then** a list. Applications wishing to provide information as to the reason(s) for the ·validation· failure are encouraged to record one or more error codes (see Outcome Tabulations (normative) (§C)) herein.
2 **otherwise** ·absent·.

Schema Information Set Contribution: Attribute Declaration
If an attribute information item is ·valid· with respect to an attribute declaration as per Attribute Locally Valid (§3.2.4) then in the post-schema-validation infoset the attribute information item may, at processor option, have a property:

PSVI Contributions for attribute information items

[attribute declaration]
An ·item isomorphic· to the declaration component itself.

Schema Information Set Contribution: Attribute Validated by Type
If clause 3 of Attribute Locally Valid (§3.2.4) applies with respect to an attribute information item, in the post-schema-validation infoset the attribute information item has a property:

PSVI Contributions for attribute information items

[schema normalized value]
The ·normalized value· of the item as ·validated·.

Furthermore, the item has one of the following alternative sets of properties:

Either

PSVI Contributions for attribute information items

[type definition]
An ·item isomorphic· to the relevant attribute declaration's {type definition} component.

[member type definition]
If and only if that type definition has {variety} *union*, then an ·item isomorphic· to that member of its {member type definitions} which actually ·validated· the attribute item's [normalized value].

or

PSVI Contributions for attribute information items

[type definition type]
simple.

[type definition namespace]
The {target namespace} of the ·type definition·.

[type definition anonymous]
true if the {name} of the ·type definition· is ·absent·, otherwise *false*.

[type definition name]
The {name} of the ·type definition·, if it is not ·absent·. If it is ·absent·, schema processors may, but need not, provide a value unique to the definition.

If the ·type definition· has {variety} *union*, then calling [Definition:] that member of the {member type definitions} which actually ·validated· the attribute item's ·normalized value· the **actual member type definition**, there are three additional properties:

PSVI Contributions for attribute information items

[member type definition namespace]
The {target namespace} of the ·actual member type definition·.

[member type definition anonymous]
true if the {name} of the ·actual member type definition· is ·absent·, otherwise *false*.

[member type definition name]
The {name} of the ·actual member type definition·, if it is not ·absent·. If it is ·absent·, schema processors may, but need not, provide a value unique to the definition.

The first (·item isomorphic·) alternative above is provided for applications such as query processors which need access to the full range of details about an item's ·assessment·, for example the type hierarchy; the second, for lighter-weight processors for whom representing the significant parts of the type hierarchy as information items might be a significant burden.

Also, if the declaration has a {value constraint}, the item has a property:

PSVI Contributions for attribute information items

[schema default]
The canonical lexical representation of the declaration's {value constraint} value.

If the attribute information item was not ·strictly assessed·, then instead of the values specified above,
1 The item's [schema normalized value] property has the ·initial value· of the item as its value;
2 The [type definition] and [member type definition] properties, or their alternatives, are based on the ·simple ur-type definition·.

3.2.6 Constraints on Attribute Declaration Schema Components

All attribute declarations (see Attribute Declarations (§3.2)) must satisfy the following constraints.

Schema Component Constraint: Attribute Declaration Properties Correct
All of the following must be true:
1 The values of the properties of an attribute declaration must be as described in the property tableau in The Attribute Declaration Schema Component (§3.2.1), modulo the impact of Missing Sub-components (§5.3).
2 if there is a {value constraint}, the canonical lexical representation of its value must be ·valid· with respect to the {type definition} as defined in String Valid (§3.14.4).
3 If the {type definition} is or is derived from ID then there must not be a {value constraint}.

Schema Component Constraint: xmlns Not Allowed

The {name} of an attribute declaration must not match `xmlns`.

> **NOTE:** The {name} of an attribute is an ·NCName·, which implicitly prohibits attribute declarations of the form `xmlns:*`.

Schema Component Constraint: xsi: Not Allowed

The {target namespace} of an attribute declaration, whether local or top-level, must not match `http://www.w3.org/2001/XMLSchema-instance` (unless it is one of the four built-in declarations given in the next section).

> **NOTE:** This reinforces the special status of these attributes, so that they not only *need* not be declared to be allowed in instances, but *must* not be declared. It also removes any temptation to experiment with supplying global or fixed values for e.g. `xsi:type` or `xsi:nil`, which would be seriously misleading, as they would have no effect.

3.2.7 Built-in Attribute Declarations

There are four attribute declarations present in every schema by definition:

Attribute Declaration for the 'type' attribute

Property	Value
{name}	`type`
{target namespace}	`http://www.w3.org/2001/XMLSchema-instance`
{type definition}	The built-in QName simple type definition
{scope}	*global*
{value constraint}	·absent·
{annotation}	·absent·

Attribute Declaration for the 'nil' attribute

Property	Value
{name}	`nil`
{target namespace}	`http://www.w3.org/2001/XMLSchema-instance`
{type definition}	The built-in boolean simple type definition
{scope}	*global*
{value constraint}	·absent·
{annotation}	·absent·

Attribute Declaration for the 'schemaLocation' attribute

Property	Value
{name}	`schemaLocation`
{target namespace}	`http://www.w3.org/2001/XMLSchema-instance`
{type definition}	An anonymous simple type definition, as follows:

Property	Value
{name}	·absent·
{target namespace}	`http://www.w3.org/2001/XMLSchema-instance`
{base type definition}	The built in ·simple ur-type definition·
{facets}	·absent·
{variety}	*list*
{item type definition}	The built-in anyURI simple type definition
{annotation}	·absent·

{scope}	*global*
{value constraint}	·absent·
{annotation}	·absent·

Attribute Declaration for the 'noNamespaceSchemaLocation' attribute

Property	Value
{name}	`noNamespaceSchemaLocation`
{target namespace}	`http://www.w3.org/2001/XMLSchema-instance`
{type definition}	The built-in anyURI simple type definition
{scope}	*global*
{value constraint}	·absent·
{annotation}	·absent·

3.3 Element Declarations

Element declarations provide for:

- Local ·validation· of element information item values using a type definition;
- Specifying default or fixed values for an element information items;
- Establishing uniquenesses and reference constraint relationships among the values of related elements and attributes;
- Controlling the substitutability of elements through the mechanism of ·element substitution groups·.

Example

```
<xs:element name="PurchaseOrder" type="PurchaseOrderType"/>
```

```
<xs:element name="gift">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="birthday" type="xs:date"/>
   <xs:element ref="PurchaseOrder"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

XML representations of several different types of element declaration

3.3.1 The Element Declaration Schema Component

The element declaration schema component has the following properties:

Schema Component: Element Declaration

{name}
An NCName as defined by [XML-Namespaces].

{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].

{type definition}
Either a simple type definition or a complex type definition.

{scope}
Optional. Either *global* or a complex type definition.

{value constraint}
Optional. A pair consisting of a value and one of *default*, *fixed*.

{nillable}
A boolean.

{identity-constraint definitions}
A set of constraint definitions.

{substitution group affiliation}
Optional. A top-level element definition.

{substitution group exclusions}
A subset of {*extension*, *restriction*}.

{disallowed substitutions}
A subset of {*substitution*, *extension*, *restriction*}.

{abstract}
A boolean.

{annotation}
Optional. An annotation.

The {name} property must match the local part of the names of element information items being ·validated·.

A {scope} of *global* identifies element declarations available for use in content models throughout the schema. Locally scoped declarations are available for use only within the complex type identified by the {scope} property. This property is ·absent· in the case of declarations within named model groups: their scope is determined when they are used in the construction of complex type definitions.

A non-·absent· value of the {target namespace} property provides for ·validation· of namespace-qualified element information items. ·Absent· values of {target namespace} ·validate· unqualified items.

An element information item is ·valid· if it satisfies the {type definition}. For such an item, schema information set contributions appropriate to the {type definition} are added to the corresponding element information item in the post-schema-validation infoset.

If {nillable} is *true*, then an element may also be ·valid· if it carries the namespace qualified attribute with [local name] `nil` from namespace `http://www.w3.org/2001/XMLSchema-instance` and value `true` (see xsi:nil (§2.6.2)) even if it has no text or element content despite a {content type} which would otherwise require content. Formal details of element ·validation· are described in Element Locally Valid (Element) (§3.3.4).

{value constraint} establishes a default or fixed value for an element. If *default* is specified, and if the element being ·validated· is empty, then the canonical form of the supplied constraint value becomes the [schema normalized value] of the ·validated· element in the post-schema-validation infoset. If *fixed* is specified, then the element's content must either be empty, in which case *fixed* behaves as *default*, or its value must match the supplied constraint value.

> **NOTE:** The provision of defaults for elements goes beyond what is possible in XML 1.0 DTDs, and does not exactly correspond to defaults for attributes. In particular, an element with a non-empty {value constraint} whose simple type definition includes the empty string in its lexical space will nonetheless never receive that value, because the {value constraint} will override it.

{identity-constraint definitions} express constraints establishing uniquenesses and reference relationships among the values of related elements and attributes. See Identity-constraint Definitions (§3.11).

Element declarations are members of the substitution group, if any, identified by {substitution group affiliation}. Membership is transitive but not symmetric; an element declaration is a member of any group of which its {substitution group affiliation} is a member.

An empty {substitution group exclusions} allows a declaration to be nominated as the {substitution group affiliation} of other element declarations having the same {type definition} or types derived therefrom. The explicit values of {substitution group exclusions} rule out element declarations having types which are *extension*s or *restriction*s respectively of {type definition}. If both values are specified, then the declaration may not be nominated as the {substitution group affiliation} of any other declaration.

The supplied values for {disallowed substitutions} determine whether an element declaration appearing in a ·content model· will be prevented from additionally ·validating· elements (a) with anxsi:type (§2.6.1) that identifies an *extension* or *restriction* of the type of the declared element, and/or (b) from ·validating· elements which are in the substitution group headed by the declared element. If {disallowed substitutions} is empty, then all derived types and substitution group members are allowed.

Element declarations for which {abstract} is *true* can appear in content models only when substitution is allowed; such declarations may not themselves ever be used to ·validate· element content.

See Annotations (§3.13) for information on the role of the {annotation} property.

3.3.2 XML Representation of Element Declaration Schema Components

The XML representation for an element declaration schema component is an <element> element information item. It specifies a type definition for an element either by reference or explicitly, and may provide occurrence and default information. The correspondences between the properties of the information item and properties of the component(s) it corresponds to are as follows:

XML Representation Summary: `element` Element Information Item

```
<element
  abstract = boolean : false
  block = (#all | List of (extension | restriction | substitution))
  default = string
```

```
  final = (#all | List of (extension | restriction))
  fixed = string
  form = (qualified | unqualified)
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded)  : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  nillable = boolean : false
  ref = QName
  substitutionGroup = QName
  type = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, ((simpleType | complexType)?, (unique | key | keyref)
*))
</element>
```

If the <element> element information item has <schema> as its parent, the corresponding schema component is as follows:

Element Declaration Schema Component

Property	Representation
{name}	The ·actual value· of the `name` [attribute].
{target namespace}	The ·actual value· of the `targetNamespace` [attribute] of the parent <schema> element information item, or ·absent· if there is none.
{scope}	*global*.
{type definition}	The type definition corresponding to the <simpleType> or <complexType> element information item in the [children], if either is present, otherwise the type definition ·resolved· to by the ·actual value· of the `type` [attribute], otherwise the {type definition} of the element declaration ·resolved· to by the ·actual value· of the `substitutionGroup` [attribute], if present, otherwise the ·ur-type definition·.
{nillable}	The ·actual value· of the `nillable` [attribute], if present, otherwise *false*.
{value constraint}	If there is a `default` or a `fixed` [attribute], then a pair consisting of the ·actual value· (with respect to the {type definition}, if it is a simple type definition, or the {type definition}'s {content type}, if that is a simple type definition, or else with respect to the built-in string simple type definition) of that [attribute] and either *default* or *fixed*, as appropriate, otherwise ·absent·.
{identity-constraint definitions}	A set consisting of the identity-constraint-definitions corresponding to all the <key>, <unique> and <keyref> element information items in the [children], if any, otherwise the empty set.
{substitution group affiliation}	The element declaration ·resolved· to by the ·actual value· of the `substitutionGroup` [attribute], if present, otherwise ·absent·.

{disallowed substitutions}	A set depending on the ·actual value· of the `block` [attribute], if present, otherwise on the ·actual value· of the `blockDefault` [attribute] of the ancestor <schema> element information item, if present, otherwise on the empty string. Call this the **EBV** (for effective block value). Then the value of this property is the appropriate **case** among the following: 1 **If** the **EBV** is the empty string, **then** the empty set; 2 **If** the **EBV** is `#all`, **then** {*extension*, *restriction*, *substitution*}; 3 **otherwise** a set with members drawn from the set above, each being present or absent depending on whether the ·actual value· (which is a list) contains an equivalently named item. **NOTE:** Although the `blockDefault` [attribute] of <schema> may include values other than *extension*, *restriction* or *substitution*, those values are ignored in the determination of {disallowed substitutions} for element declarations (they *are* used elsewhere).
{substitution group exclusions}	As for {disallowed substitutions} above, but using the `final` and `finalDefault` [attributes] in place of the `block` and `blockDefault` [attributes] and with the relevant set being {*extension*, *restriction*}.
{abstract}	The ·actual value· of the `abstract` [attribute], if present, otherwise *false*.
{annotation}	The annotation corresponding to the <annotation> element information item in the [children], if present, otherwise ·absent·.

otherwise if the <element> element information item has <complexType> or <group> as an ancestor and the `ref` [attribute] is absent, the corresponding schema components are as follows (unless `minOccurs=maxOccurs=0`, in which case the item corresponds to no component at all):

Particle Schema Component

Property	Representation
{min occurs}	The ·actual value· of the `minOccurs` [attribute], if present, otherwise `1`.
{max occurs}	*unbounded*, if the `maxOccurs` [attribute] equals *unbounded*, otherwise the ·actual value· of the `maxOccurs` [attribute], if present, otherwise `1`.
{term}	A (local) element declaration as given below.

An element declaration as in the first case above, with the exception of its {target namespace} and {scope} properties, which are as below:

Element Declaration Schema Component

Property	Representation
{target namespace}	If `form` is present and its ·actual value· is `qualified`, or if `form` is absent and the ·actual value· of `elementFormDefault` on the <schema> ancestor is `qualified`, then the ·actual value· of the `targetNamespace` [attribute] of the parent <schema> element information item, or ·absent· if there is none, otherwise ·absent·.
{scope}	If the <element> element information item has <complexType> as an ancestor, the complex definition corresponding to that item, otherwise (the <element> element information item is within a named <group> definition), ·absent·.

otherwise (the <element> element information item has <complexType> or <group> as an ancestor and the `ref` [attribute] is present), the corresponding schema component is as follows (unless `minOccurs=maxOccurs=0`, in which case the item corresponds to no component at all):

Particle Schema Component

Property	Representation
{min occurs}	The ·actual value· of the `minOccurs` [attribute], if present, otherwise `1`.
{max occurs}	*unbounded*, if the `maxOccurs` [attribute] equals *unbounded*, otherwise the ·actual value· of the `maxOccurs` [attribute], if present, otherwise `1`.
{term}	The (top-level) element declaration ·resolved· to by the ·actual value· of the `ref` [attribute].

<element> corresponds to an element declaration, and allows the type definition of that declaration to be specified either by reference or by explicit inclusion.

<element>s within <schema> produce *global* element declarations; <element>s within <group> or <complexType> produce either particles which contain *global* element declarations (if there's a `ref` attribute) or local declarations (otherwise). For complete declarations, top-level or local, the `type` attribute is used when the declaration can use a built-in or pre-declared type definition. Otherwise an anonymous <simpleType> or <complexType> is provided inline.

Element information items ·validated· by a top-level declaration must be qualified with the {target namespace} of that declaration (if this is ·absent·, the item must be unqualified). Control over whether element information items ·validated· by a local declaration must be similarly qualified or not is provided by the `form` [attribute], whose default is provided by the `elementFormDefault` [attribute] on the enclosing <schema>, via its determination of {target namespace}.

As noted above the names for top-level element declarations are in a separate ·symbol space· from the symbol spaces for the names of type definitions, so there can (but need not be) a simple or complex type definition with the same name as a top-level element. As with attribute names, the names of locally-scoped element declarations with no {target namespace} reside in symbol spaces local to the type definition which contains them.

Note that the above allows for two levels of defaulting for unspecified type definitions. An <element> with no referenced or included type definition will correspond to an element declaration which has the same type definition as the head of its substitution group if it identifies one, otherwise the ·ur-type definition·. This has the important consequence that the minimum valid element declaration, that is, one with only a `name` attribute and no contents, is also the most general, validating any combination of text and element content and allowing any attributes.

See below at XML Representation of Identity-constraint Definition Schema Components (§3.11.2) for <key>, <unique> and <keyref>.

Example

```
<xs:element name="unconstrained"/>

<xs:element name="emptyElt">
 <xs:complexType>
  <xs:attribute ...>. . .</xs:attribute>
 </xs:complexType>
</xs:element>

<xs:element name="contextOne">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="myLocalElement" type="myFirstType"/>
   <xs:element ref="globalElement"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>

<xs:element name="contextTwo">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="myLocalElement" type="mySecondType"/>
   <xs:element ref="globalElement"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

The first example above declares an element whose type, by default, is the ·ur-type definition·. The second uses an embedded anonymous complex type definition.

The last two examples illustrate the use of local element declarations. Instances of `myLocalElement` within `contextOne` will be constrained by `myFirstType`, while those within `contextTwo` will be constrained by `mySecondType`.

NOTE: The possibility that differing attribute declarations and/or content models would apply to elements with the same name in different contexts is an extension beyond the expressive power of a DTD in XML 1.0.

Example

```
<xs:complexType name="facet">
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:attribute name="value" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="facet" type="xs:facet" abstract="true"/>

<xs:element name="encoding" substitutionGroup="xs:facet">
 <xs:complexType>
  <xs:complexContent>
   <xs:restriction base="xs:facet">
    <xs:sequence>
```

```
      <xs:element ref="annotation" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="value" type="xs:encodings"/>
    </xs:restriction>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>

 <xs:element name="period" substitutionGroup="xs:facet">
  <xs:complexType>
   <xs:complexContent>
    <xs:restriction base="xs:facet">
     <xs:sequence>
      <xs:element ref="annotation" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="value" type="xs:duration"/>
    </xs:restriction>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>

 <xs:complexType name="datatype">
  <xs:sequence>
   <xs:element ref="facet" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:NCName" use="optional"/>
  . . .
 </xs:complexType>
```

An example from a previous version of the schema for datatypes. The `facet` type is defined and the `facet` element is declared to use it. The `facet` element is abstract -- it's *only* defined to stand as the head for a substitution group. Two further elements are declared, each a member of the `facet` substitution group. Finally a type is defined which refers to `facet`, thereby allowing *either* `period` or `encoding` (or any other member of the group).

3.3.3 Constraints on XML Representations of Element Declarations

Schema Representation Constraint: Element Declaration Representation OK

In addition to the conditions imposed on <element> element information items by the schema for schemas: **all** of the following must be true:

1 `default` and `fixed` must not both be present.

2 If the item's parent is not <schema>, then **all** of the following must be true:

2.1 One of `ref` or `name` must be present, but not both.

2.2 If `ref` is present, then all of <complexType>, <simpleType>, <key>, <keyref>, <unique>, `nillable`, `default`, `fixed`, `form`, `block` and `type` must be absent, i.e. only `minOccurs`, `maxOccurs`, `id` are allowed in addition to `ref`, along with <annotation>.

3 `type` and either <simpleType> or <complexType> are mutually exclusive.

4 The corresponding particle and/or element declarations must satisfy the conditions set out in Constraints on Element Declaration Schema Components (§3.3.6) and Constraints on Particle Schema Components (§3.9.6).

3.3.4 Element Declaration Validation Rules

Validation Rule: Element Locally Valid (Element)

For an element information item to be locally ·valid· with respect to an element declaration **all** of the following must be true:

1 The declaration must not be ·absent·.

2 Its {abstract} must be *false*.

3 The appropriate **case** among the following must be true:

3.1 **If** {nillable} is *false*, **then** there must be no attribute information item among the element information item's [attributes] whose [namespace name] is identical to `http://www.w3.org/2001/XMLSchema-instance` and whose [local name] is `nil`.

3.2 **If** {nillable} is *true* and there is such an attribute information item and its ·actual value· is `true` , **then all** of the following must be true:

3.2.1 The element information item must have no character or element information item [children].

3.2.2 There must be no *fixed* {value constraint}.

4 If there is an attribute information item among the element information item's [attributes] whose [namespace name] is identical to `http://www.w3.org/2001/XMLSchema-instance` and whose [local name] is `type`, then **all** of the following must be true:

4.1 The ·normalized value· of that attribute information item must be ·valid· with respect to the built-in QName simple type, as defined by String Valid (§3.14.4);

4.2 The ·local name· and ·namespace name· (as defined in QName Interpretation (§3.15.3)), of the ·actual value· of that attribute information item must resolve to a type definition, as defined in QName resolution (Instance) (§3.15.4) -- [Definition:] call this type definition the **local type definition**;

4.3 The ·local type definition· must be validly derived from the {type definition} given the union of the {disallowed substitutions} and the {type definition}'s {prohibited substitutions}, as defined in Type Derivation OK (Complex) (§3.4.6) (if it is a complex type definition), or given {disallowed substitutions} as defined in Type Derivation OK (Simple) (§3.14.6) (if it is a simple type definition).

[Definition:] The phrase **actual type definition** occurs below. If the above three clauses are satisfied, this should be understood as referring to the ·local type definition·, otherwise to the {type definition}.

5 The appropriate **case** among the following must be true:

5.1 **If** the declaration has a {value constraint}, the item has neither element nor character [children] and clause 3.2 has not applied, **then all** of the following must be true:

5.1.1 If the ·actual type definition· is a ·local type definition· then the canonical lexical representation of the {value constraint} value must be a valid default for the ·actual type definition· as defined in Element Default Valid (Immediate) (§3.3.6).

5.1.2 The element information item with the canonical lexical representation of the {value constraint} value used as its ·normalized value· must be ·valid· with respect to the ·actual type definition· as defined by Element Locally Valid (Type) (§3.3.4).

5.2 **If** the declaration has no {value constraint} or the item has either element or character [children] or clause 3.2 has applied, **then all** of the following must be true:

5.2.1 The element information item must be ·valid· with respect to the ·actual type definition· as defined by Element Locally Valid (Type) (§3.3.4).

5.2.2 If there is a *fixed* {value constraint} and clause 3.2 has not applied, **all** of the following must be true:

5.2.2.1 The element information item must have no element information item [children].

5.2.2.2 The appropriate **case** among the following must be true:

5.2.2.2.1 **If** the {content type} of the ·actual type definition· is *mixed*, **then** the ·initial value· of the item must match the canonical lexical representation of the {value constraint} value.

5.2.2.2.2 **If** the {content type} of the ·actual type definition· is a simple type definition, **then** the ·actual value· of the item must match the canonical lexical representation of the {value constraint} value.

6 The element information item must be ·valid· with respect to each of the {identity-constraint definitions} as per Identity-constraint Satisfied (§3.11.4).

7 If the element information item is the ·validation root·, it must be ·valid· per Validation Root Valid (ID/IDREF) (§3.3.4).

Validation Rule: Element Locally Valid (Type)

For an element information item to be locally ·valid· with respect to a type definition **all** of the following

must be true:

1 The type definition must not be ·absent·;

2 Its {abstract} must be *false*.

3 The appropriate **case** among the following must be true:

3.1 **If** the type definition is a simple type definition, **then all** of the following must be true:

3.1.1 The element information item's [attributes] must be empty, excepting those whose [namespace name] is identical to `http://www.w3.org/2001/XMLSchema-instance` and whose [local name] is one of `type`, `nil`, `schemaLocation` or `noNamespaceSchemaLocation`.

3.1.2 The element information item must have no element information item [children].

3.1.3 If clause 3.2 of Element Locally Valid (Element) (§3.3.4) did not apply, then the ·normalized value· must be ·valid· with respect to the type definition as defined by String Valid (§3.14.4).

3.2 **If** the type definition is a complex type definition, **then** the element information item must be ·valid· with respect to the type definition as per Element Locally Valid (Complex Type) (§3.4.4);

Validation Rule: Validation Root Valid (ID/IDREF)

For an element information item which is the ·validation root· to be ·valid· **all** of the following must be true:

1 There must be no **ID/IDREF binding** in the item's [ID/IDREF table] whose [binding] is the empty set.

2 There must be no **ID/IDREF binding** in the item's [ID/IDREF table] whose [binding] has more than one member.

See ID/IDREF Table (§3.15.5) for the definition of **ID/IDREF binding**.

NOTE: The first clause above applies when there is a reference to an undefined ID. The second applies when there is a multiply-defined ID. They are separated out to ensure that distinct error codes (see Outcome Tabulations (normative) (§C)) are associated with these two cases.

NOTE: Although this rule applies at the ·validation root·, in practice processors, particularly streaming processors, may wish to detect and signal the clause 2 case as it arises.

NOTE: This reconstruction of [XML 1.0 (Second Edition)]'s `ID/IDREF` functionality is imperfect in that if the ·validation root· is not the document element of an XML document, the results will not necessarily be the same as those a validating parser would give were the document to have a DTD with equivalent declarations.

Validation Rule: Schema-Validity Assessment (Element)

The schema-validity assessment of an element information item depends on its ·validation· and the ·assessment· of its element information item children and associated attribute information items, if any.

So for an element information item's schema-validity to be assessed **all** of the following must be true:

1 **One** of the following must be true:

1.1 **All** of the following must be true:

1.1.1 A non-·absent· element declaration must be known for it, because **one** of the following is true

1.1.1.1 A declaration was stipulated by the processor (see Assessing Schema-Validity (§5.2)).

1.1.1.2 A declaration has been established as its ·context-determined declaration·.

1.1.1.3 **All** of the following must be true:

1.1.1.3.1 Its ·context-determined declaration· is not *skip*.

1.1.1.3.2 Its [local name] and [namespace name] resolve to an element declaration as defined by QName resolution (Instance) (§3.15.4).

1.1.2 Its ·validity· with respect to that declaration must have been evaluated as per Element Locally Valid (Element) (§3.3.4).
1.1.3 If that evaluation involved the evaluation of Element Locally Valid (Type) (§3.3.4), clause 1 thereof must be satisfied.
1.2 **All** of the following must be true:
1.2.1 A non-·absent· type definition is known for it because **one** of the following is true
1.2.1.1 A type definition was stipulated by the processor (see Assessing Schema-Validity (§5.2)).
1.2.1.2 **All** of the following must be true:
1.2.1.2.1 There is an attribute information item among the element information item's [attributes] whose [namespace name] is identical to `http://www.w3.org/2001/XMLSchema-instance` and whose [local name] is `type`.
1.2.1.2.2 The ·normalized value· of that attribute information item is ·valid· with respect to the built-in QName simple type, as defined by String Valid (§3.14.4).
1.2.1.2.3 The ·local name· and ·namespace name· (as defined in QName Interpretation (§3.15.3)), of the ·actual value· of that attribute information item resolve to a type definition, as defined in QName resolution (Instance) (§3.15.4) -- [Definition:] call this type definition the **local type definition**.
1.2.1.2.4 If there is also a processor-stipulated type definition, the ·local type definition· must be validly derived from that type definition given its {prohibited substitutions}, as defined in Type Derivation OK (Complex) (§3.4.6) (if it is a complex type definition), or given the empty set, as defined in Type Derivation OK (Simple) (§3.14.6) (if it is a simple type definition).
1.2.2 The element information item's ·validity· with respect to the ·local type definition· (if present and validly derived) or the processor-stipulated type definition (if no ·local type definition· is present) has been evaluated as per Element Locally Valid (Type) (§3.3.4).
2 The schema-validity of all the element information items among its [children] has been assessed as per Schema-Validity Assessment (Element) (§3.3.4), and the schema-validity of all the attribute information items among its [attributes] has been assessed as per Schema-Validity Assessment (Attribute) (§3.2.4).

[Definition:] If either case of clause 1 above holds, the element information item has been **strictly assessed**.

If the item cannot be ·strictly assessed·, because neither clause 1.1 nor clause 1.2 above are satisfied, [Definition:] an element information item's schema validity may be **laxly assessed** if its ·context-determined declaration· is not *skip* by ·validating· with respect to the ·ur-type definition· as per Element Locally Valid (Type) (§3.3.4).

NOTE: In general if clause 1.1 above holds clause 1.2 does not, and vice versa. When an `xsi:type` [attribute] is involved, however, clause 1.2 takes precedence, as is made clear in Element Locally Valid (Element) (§3.3.4).

NOTE: The {name} and {target namespace} properties are not mentioned above because they are checked during particle ·validation·, as per Element Sequence Locally Valid (Particle) (§3.9.4).

3.3.5 Element Declaration Information Set Contributions

Schema Information Set Contribution: Assessment Outcome (Element)
If the schema-validity of an element information item has been assessed as per Schema-Validity Assessment (Element) (§3.3.4), then in the post-schema-validation infoset it has properties as follows:

PSVI Contributions for element information items

[validation context]
The nearest ancestor element information item with a [schema information] property (or this element item itself if it has such a property).

[validity]
The appropriate **case** among the following:

1 **If** it was ·strictly assessed·, **then** the appropriate **case** among the following:

1.1 **If all** of the following are true

1.1.1

1.1.1.1 clause 1.1 of Schema-Validity Assessment (Element) (§3.3.4) applied and the item was ·valid· as defined by Element Locally Valid (Element) (§3.3.4);

1.1.1.2 clause 1.2 of Schema-Validity Assessment (Element) (§3.3.4) applied and the item was ·valid· as defined by Element Locally Valid (Type) (§3.3.4).

1.1.2 Neither its [children] nor its [attributes] contains an information item (element or attribute respectively) whose [validity] is *invalid*.

1.1.3 Neither its [children] nor its [attributes] contains an information item (element or attribute respectively) with a ·context-determined declaration· of *mustFind* whose [validity] is *unknown*.

, **then** *valid*;

1.2 **otherwise** *invalid*..

2 **otherwise** *notKnown*.

[validation attempted]
The appropriate **case** among the following:

1 **If** it was ·strictly assessed· and neither its [children] nor its [attributes] contains an information item (element or attribute respectively) whose [validation attempted] is not *full*, **then** *full*;

2 **If** it was not ·strictly assessed· and neither its [children] nor its [attributes] contains an information item (element or attribute respectively) whose [validation attempted] is not *none*, **then** *none*;

3 **otherwise** *partial*.

Schema Information Set Contribution: Validation Failure (Element)

If the local ·validity·, as defined by Element Locally Valid (Element) (§3.3.4) above and/or Element Locally Valid (Type) (§3.3.4) below, of an element information item has been assessed, in the post-schema-validation infoset the item has a property:

PSVI Contributions for element information items

[schema error code]
The appropriate **case** among the following:

1 **If** the item is not ·valid·, **then** a list. Applications wishing to provide information as to the reason(s) for the ·validation· failure are encouraged to record one or more error codes (see Outcome Tabulations (normative) (§C)) herein.

2 **otherwise** ·absent·.

Schema Information Set Contribution: Element Declaration

If an element information item is ·valid· with respect to an element declaration as per Element Locally Valid (Element) (§3.3.4) then in the post-schema-validation infoset the element information item must, at processor option, have either:

PSVI Contributions for element information items

[element declaration]
an ·item isomorphic· to the declaration component itself

or

PSVI Contributions for element information items

[nil]
true if clause 3.2 of Element Locally Valid (Element) (§3.3.4) above is satisfied, otherwise *false*

Schema Information Set Contribution: Element Validated by Type

If an element information item is ·valid· with respect to a ·type definition· as per Element Locally Valid (Type) (§3.3.4), in the post-schema-validation infoset the item has a property:

PSVI Contributions for element information items

[schema normalized value]
The appropriate **case** among the following:
1 **If** clause 3.2 of Element Locally Valid (Element) (§3.3.4) and Element Default Value (§3.3.5) above have *not* applied and either the ·type definition· is a simple type definition or its {content type} is a simple type definition, **then** the ·normalized value· of the item as ·validated·.
2 **otherwise** ·absent·.

Furthermore, the item has one of the following alternative sets of properties:

Either

PSVI Contributions for element information items

[type definition]
An ·item isomorphic· to the ·type definition· component itself.
[member type definition]
If and only if that type definition is a simple type definition with {variety} *union*, or a complex type definition whose {content type} is a simple thype definition with {variety} *union*, then an ·item isomorphic· to that member of the union's {member type definitions} which actually ·validated· the element item's ·normalized value·.

or

PSVI Contributions for element information items

[type definition type]
simple or *complex*, depending on the ·type definition·.
[type definition namespace]
The {target namespace} of the ·type definition·.
[type definition anonymous]
true if the {name} of the ·type definition· is ·absent·, otherwise *false*.
[type definition name]
The {name} of the ·type definition·, if it is not ·absent·. If it is ·absent·, schema processors may, but need not, provide a value unique to the definition.

If the ·type definition· is a simple type definition or its {content type} is a simple type definition, and that type definition has {variety} *union*, then calling [Definition:] that member of the {member type definitions} which actually ·validated· the element item's ·normalized value· the **actual member type definition**, there are three additional properties:

PSVI Contributions for element information items

[member type definition namespace]

The {target namespace} of the ·actual member type definition·.

[member type definition anonymous]
true if the {name} of the ·actual member type definition· is ·absent·, otherwise *false*.

[member type definition name]
The {name} of the ·actual member type definition·, if it is not ·absent·. If it is ·absent·, schema processors may, but need not, provide a value unique to the definition.

The first (·item isomorphic·) alternative above is provided for applications such as query processors which need access to the full range of details about an item's ·assessment·, for example the type hierarchy; the second, for lighter-weight processors for whom representing the significant parts of the type hierarchy as information items might be a significant burden.

Also, if the declaration has a {value constraint}, the item has a property:

PSVI Contributions for element information items

[schema default]
The canonical lexical representation of the declaration's {value constraint} value.

Note that if an element is ·laxly assessed·, then the [type definition] and [member type definition] properties, or their alternatives, are based on the ·ur-type definition·.

Schema Information Set Contribution: Element Default Value

If the local ·validity·, as defined by Element Locally Valid (Element) (§3.3.4) above, of an element information item has been assessed, in the post-schema-validation infoset the item has a property:

PSVI Contributions for element information items

[schema specified]
The appropriate **case** among the following:
1 **If** the item is ·valid· with respect to an element declaration as per Element Locally Valid (Element) (§3.3.4) and the {value constraint} is present, but clause 3.2 of Element Locally Valid (Element) (§3.3.4) above is not satisfied and the item has no element or character information item [children], **then** *schema*. Furthermore, the post-schema-validation infoset has the canonical lexical representation of the {value constraint} value as the item's [schema normalized value] property.
2 **otherwise** *infoset*.

3.3.6 Constraints on Element Declaration Schema Components

All element declarations (see Element Declarations (§3.3)) must satisfy the following constraint.

Schema Component Constraint: Element Declaration Properties Correct

All of the following must be true:
1 The values of the properties of an element declaration must be as described in the property tableau in The Element Declaration Schema Component (§3.3.1), modulo the impact of Missing Sub-components (§5.3).
2 If there is a {value constraint}, the canonical lexical representation of its value must be ·valid· with respect to the {type definition} as defined in Element Default Valid (Immediate) (§3.3.6).
3 If there is an {substitution group affiliation}, the {type definition} of the element declaration must be validly derived from the {type definition} of the {substitution group affiliation}, given the value of the {substitution group exclusions} of the {substitution group affiliation}, as defined in Type Derivation OK (Complex) (§3.4.6) (if the {type definition} is complex) or as defined in Type Derivation OK (Simple) (§3.14.6) (if the {type definition} is simple).
4 If the {type definition} or {type definition}'s {content type} is or is derived from ID then there must

not be a {value constraint}.

> **NOTE:** The use of ID as a type definition for elements goes beyond XML 1.0, and should be avoided if backwards compatibility is desired.

The following constraints define relations appealed to elsewhere in this specification.

Schema Component Constraint: Element Default Valid (Immediate)

For a string to be a valid default with respect to a type definition the appropriate **case** among the following must be true:

1 **If** the type definition is a simple type definition, **then** the string must be ·valid· with respect to that definition as defined by String Valid (§3.14.4).

2 **If** the type definition is a complex type definition, **then all** of the following must be true:

2.1 its {content type} must be a simple type definition or *mixed*.

2.2 The appropriate **case** among the following must be true:

2.2.1 **If** the {content type} is a simple type definition, **then** the string must be ·valid· with respect to that simple type definition as defined by String Valid (§3.14.4).

2.2.2 **If** the {content type} is *mixed*, **then** the {content type}'s particle must be ·emptiable· as defined by Particle Emptiable (§3.9.6).

Schema Component Constraint: Substitution Group OK (Transitive)

For an element declaration (call it **D**) together with a blocking constraint (a subset of {*substitution*, *extension*, *restriction*}, the value of a {disallowed substitutions}) to be validly substitutable for another element declaration (call it **C**) **all** of the following must be true:

1 The blocking constraint does not contain *substitution*.

2 There is a chain of {substitution group affiliation}s from **D** to **C**, that is, either **D**'s {substitution group affiliation} is **C**, or **D**'s {substitution group affiliation}'s {substitution group affiliation} is **C**, or . . .

3 The set of all {derivation method}s involved in the derivation of **D**'s {type definition} from **C**'s {type definition} does not intersect with the union of the blocking constraint, **C**'s {prohibited substitutions} (if **C** is complex, otherwise the empty set) and the {prohibited substitutions} (respectively the empty set) of any intermediate {type definition}s in the derivation of **D**'s {type definition} from **C**'s {type definition}.

Schema Component Constraint: Substitution Group

[Definition:] Every element declaration in the {element declarations} of a schema defines a **substitution group**, a subset of those {element declarations}, as follows:

1 The element declaration itself is in the group;

2 The group is closed with respect to {substitution group affiliation}, that is, if any element declaration in the {element declarations} has a {substitution group affiliation} in the group, then it is also in the group itself.

3.4 Complex Type Definitions

Complex Type Definitions provide for:

- Constraining element information items by providing Attribute Declaration (§2.2.2.3)s governing the appearance and content of [attributes]
- Constraining element information item [children] to be empty, or to conform to a specified element-

only or mixed content model, or else constraining the character information item [children] to conform to a specified simple type definition.
- Using the mechanisms of Type Definition Hierarchy (§2.2.1.1) to derive a complex type from another simple or complex type.
- Specifying ·post-schema-validation infoset contributions· for elements.
- Limiting the ability to derive additional types from a given complex type.
- Controlling the permission to substitute, in an instance, elements of a derived type for elements declared in a content model to be of a given complex type.

Example

```
<xs:complexType name="PurchaseOrderType">
  <xs:sequence>
   <xs:element name="shipTo" type="USAddress"/>
   <xs:element name="billTo" type="USAddress"/>
   <xs:element ref="comment" minOccurs="0"/>
   <xs:element name="items"  type="Items"/>
  </xs:sequence>
  <xs:attribute name="orderDate" type="xs:date"/>
 </xs:complexType>
```

The XML representation of a complex type definition.

3.4.1 The Complex Type Definition Schema Component

A complex type definition schema component has the following properties:

Schema Component: Complex Type Definition

{name}
Optional. An NCName as defined by [XML-Namespaces].

{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].

{base type definition}
Either a simple type definition or a complex type definition.

{derivation method}
Either *extension* or *restriction*.

{final}
A subset of {*extension*, *restriction*}.

{abstract}
A boolean

{attribute uses}
A set of attribute uses.

{attribute wildcard}
Optional. A wildcard.

{content type}
One of *empty*, a simple type definition or a pair consisting of a ·content model· (I.e. a Particle (§2.2.3.2)) and one of *mixed*, *element-only*.

{prohibited substitutions}
A subset of {*extension*, *restriction*}.

{annotations}
A set of annotations.

Complex types definitions are identified by their {name} and {target namespace}. Except for anonymous complex type definitions (those with no {name}), since type definitions (i.e. both simple and complex type definitions taken together) must be uniquely identified within an ·XML Schema·, no complex type definition can have the same name as another simple or complex type definition. Complex type {name}s and {target

namespace}s are provided for reference from instances (see xsi:type (§2.6.1)), and for use in the XML representation of schema components (specifically in <element>). See References to schema components across namespaces (§4.2.3) for the use of component identifiers when importing one schema into another.

> **NOTE:** The {name} of a complex type is not *ipso facto* the [(local) name] of the element information items ·validated· by that definition. The connection between a name and a type definition is described in Element Declarations (§3.3).

As described in Type Definition Hierarchy (§2.2.1.1), each complex type is derived from a {base type definition} which is itself either a Simple Type Definition (§2.2.1.2) or a Complex Type Definition (§2.2.1.3). {derivation method} specifies the means of derivation as either *extension* or *restriction* (see Type Definition Hierarchy (§2.2.1.1)).

A complex type with an empty specification for {final} can be used as a {base type definition} for other types derived by either of extension or restriction; the explicit values *extension*, and *restriction* prevent further derivations by extension and restriction respectively. If all values are specified, then [Definition:] the complex type is said to be **final**, because no further derivations are possible. Finality is *not* inherited, that is, a type definition derived by restriction from a type definition which is final for extension is not itself, in the absence of any explicit `final` attribute of its own, final for anything.

Complex types for which {abstract} is *true* must not be used as the {type definition} for the ·validation· of element information items. It follows that they must not be referenced from an xsi:type (§2.6.1) attribute in an instance document. Abstract complex types can be used as {base type definition}s, or even as the {type definition}s of element declarations, provided in every case a concrete derived type definition is used for ·validation·, either via xsi:type (§2.6.1) or the operation of a substitution group.

{attribute uses} are a set of attribute uses. See Element Locally Valid (Complex Type) (§3.4.4) and Attribute Locally Valid (§3.2.4) for details of attribute ·validation·.

{attribute wildcard}s provide a more flexible specification for ·validation· of attributes not explicitly included in {attribute uses}. Informally, the specific values of {attribute wildcard} are interpreted as follows:

- *any*: [attributes] can include attributes with any qualified or unqualified name.
- a set whose members are either namespace names or ·absent·: [attributes] can include any attribute(s) from the specified namespace(s). If ·absent· is included in the set, then any unqualified attributes are (also) allowed.
- *'not'* and a namespace name: [attributes] cannot include attributes from the specified namespace.
- *'not'* and ·absent·: [attributes] cannot include unqualified attributes.

See Element Locally Valid (Complex Type) (§3.4.4) and Wildcard allows Namespace Name (§3.10.4) for formal details of attribute wildcard ·validation·.

{content type} determines the ·validation· of [children] of element information items. Informally:

- A {content type} with the distinguished value *empty* ·validates· elements with no character or element information item [children].
- A {content type} which is a Simple Type Definition (§2.2.1.2) ·validates· elements with character-only [children].
- An *element-only* {content type} ·validates· elements with [children] that conform to the supplied ·content model·.
- A *mixed* {content type} ·validates· elements whose element [children] (i.e. specifically ignoring other [children] such as character information items) conform to the supplied ·content model·.

{prohibited substitutions} determine whether an element declaration appearing in a · content model· is prevented from additionally ·validating· element items with an xsi:type (§2.6.1) attribute that identifies a complex type definition derived by *extension* or *restriction* from this definition, or element items in a substitution group whose type definition is similarly derived: If {prohibited substitutions} is empty, then all such substitutions are allowed, otherwise, the derivation method(s) it names are disallowed.

See Annotations (§3.13) for information on the role of the {annotations} property.

3.4.2 XML Representation of Complex Type Definitions

The XML representation for a complex type definition schema component is a <complexType> element information item.

The XML representation for complex type definitions with a simple type definition {content type} is significantly different from that of those with other {content type}s, and this is reflected in the presentation below, which displays first the elements involved in the first case, then those for the second. The property mapping is shown once for each case.

XML Representation Summary: `complexType` Element Information Item

```
<complexType
  abstract = boolean : false
  block = (#all | List of (extension | restriction))
  final = (#all | List of (extension | restriction))
  id = ID
  mixed = boolean : false
  name = NCName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleContent | complexContent | ((group | all | choice
| sequence)?, ((attribute | attributeGroup)*, anyAttribute?))))
</complexType>
```

Whichever alternative for the content of <complexType> is chosen, the following property mappings apply:

Complex Type Definition Schema Component

Property	Representation
{name}	The ·actual value· of the `name` [attribute] if present, otherwise ·absent·.
{target namespace}	The ·actual value· of the `targetNamespace` [attribute] of the <schema> ancestor element information item if present, otherwise ·absent·.
{abstract}	The ·actual value· of the `abstract` [attribute], if present, otherwise *false*.

{prohibited substitutions}	A set corresponding to the ·actual value· of the `block` [attribute], if present, otherwise on the ·actual value· of the `blockDefault` [attribute] of the ancestor <schema> element information item, if present, otherwise on the empty string. Call this the **EBV** (for effective block value). Then the value of this property is the appropriate **case** among the following: 1 **If** the **EBV** is the empty string, **then** the empty set; 2 **If** the **EBV** is `#all`, **then** {*extension*, *restriction*}; 3 **otherwise** a set with members drawn from the set above, each being present or absent depending on whether the ·actual value· (which is a list) contains an equivalently named item. **NOTE:** Although the `blockDefault` [attribute] of <schema> may include values other than *restriction* or*extension*, those values are ignored in the determination of {prohibited substitutions} for complex type definitions (they *are* used elsewhere).
{final}	As for {prohibited substitutions} above, but using the `final` and `finalDefault` [attributes] in place of the `block` and `blockDefault` [attributes].
{annotations}	The annotations corresponding to the <annotation> element information item in the [children], if present, in the <simpleContent> and <complexContent> [children], if present, and in their <restriction> and <extension> [children], if present, otherwise ·absent·.

When the <simpleContent> alternative is chosen, the following elements are relevant, and the remaining property mappings are as below. Note that either <restriction> or <extension> must be chosen as the content of <simpleContent>.

```
<simpleContent
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (restriction | extension))
</simpleContent>
```

```
<restriction
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleType?, (minExclusive | minInclusive |
maxExclusive | maxInclusive | totalDigits | fractionDigits | length | minLength |
maxLength | enumeration | whiteSpace | pattern)*)?, ((attribute | attributeGroup)
*, anyAttribute?))
</restriction>
```

```
<extension
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, ((attribute | attributeGroup)*, anyAttribute?))
</extension>
```

```
<attributeGroup
  id = ID
```

```
  ref = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</attributeGroup>
```

```
<anyAttribute
  id = ID
  namespace = ((##any | ##other) | List of (anyURI |
(##targetNamespace | ##local)) )  : ##any
  processContents = (lax | skip | strict) : strict
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</anyAttribute>
```

Complex Type Definition with simple content Schema Component

Property	Representation
{base type definition}	The type definition ·resolved· to by the ·actual value· of the `base` [attribute]
{derivation method}	If the <restriction> alternative is chosen, then *restriction*, otherwise (the <extension> alternative is chosen) *extension*.
{attribute uses}	A union of sets of attribute uses as follows 1 The set of attribute uses corresponding to the <attribute> [children], if any. 2 The {attribute uses} of the attribute groups ·resolved· to by the ·actual value·s of the `ref` [attribute] of the <attributeGroup> [children], if any. 3 if the type definition ·resolved· to by the ·actual value· of the `base` [attribute] is a complex type definition, the {attribute uses} of that type definition, unless the <restriction> alternative is chosen, in which case some members of that type definition's {attribute uses} may not be included, namely those whose {attribute declaration}'s {name} and {target namespace} are the same as **one** of the following: 3.1 the {name} and {target namespace} of the {attribute declaration} of an attribute use in the set per clause 1 or clause 2 above; 3.2 what would have been the {name} and {target namespace} of the {attribute declaration} of an attribute use in the set per clause 1 above but for the ·actual value· of the `use` [attribute] of the relevant <attribute> among the [children] of <restriction> being *prohibited*.
{attribute wildcard}	[Definition:] Let the **local wildcard** be defined as the appropriate **case** among the following: 1 **If** there is an <anyAttribute> present, **then** a wildcard based on the ·actual value·s of the `namespace` and `processContents` [attributes] and the <annotation> [children], exactly as for the wildcard corresponding to an <any> element as set out in XML Representation of Wildcard Schema Components (§3.10.2); 2 **otherwise** ·absent·. [Definition:] Let the **complete wildcard** be defined as the appropriate **case** among the following: 1 **If** there are no <attributeGroup> [children] corresponding to attribute groups with non-·absent· {attribute wildcard}s, **then** the ·local wildcard·.

2 **If** there are one or more <attributeGroup> [children] corresponding to attribute groups with non-·absent· {attribute wildcard}s, **then** the appropriate **case** among the following:

2.1 **If** there is an <anyAttribute> present, **then** a wildcard whose {process contents} and {annotation} are those of the ·local wildcard·, and whose {namespace constraint} is the intensional intersection of the {namespace constraint} of the ·local wildcard· and of the {namespace constraint}s of all the non-·absent· {attribute wildcard}s of the attribute groups corresponding to the <attributeGroup> [children], as defined in Attribute Wildcard Intersection (§3.10.6).

2.2 **If** there is no <anyAttribute> present, **then** a wildcard whose properties are as follows:

{process contents}

The {process contents} of the first non-·absent· {attribute wildcard} of an attribute group among the attribute groups corresponding to the <attributeGroup> [children].

{namespace constraint}

The intensional intersection of the {namespace constraint}s of all the non-·absent· {attribute wildcard}s of the attribute groups corresponding to the <attributeGroup> [children], as defined in Attribute Wildcard Intersection (§3.10.6).

{annotation}

·absent·.

The value is then determined by the appropriate **case** among the following:

1 **If** the <restriction> alternative is chosen, **then** the ·complete wildcard·;

2 **If** the <extension> alternative is chosen, **then** [Definition:] let the **base wildcard** be defined as the appropriate **case** among the following:

2.1 **If** the type definition ·resolved· to by the ·actual value· of the `base` [attribute] is a complex type definition with an {attribute wildcard}, **then** that {attribute wildcard}.

2.2 **otherwise** ·absent·.

The value is then determined by the appropriate **case** among the following:

2.1 **If** the ·base wildcard· is non-·absent·, **then** the appropriate **case** among the following:

2.1.1 **If** the ·complete wildcard· is ·absent·, **then** the ·base wildcard·.

2.1.2 **otherwise** a wildcard whose {process contents} and {annotation} are those of the ·complete wildcard·, and whose {namespace constraint} is the intensional union of the {namespace constraint} of the ·effective wildcard· and of the ·base wildcard·, as defined in Attribute Wildcard Union (§3.10.6).

{content type}	1 if the type definition ·resolved· to by the ·actual value· of the `base` [attribute] is a complex type definition (whose own {content type} must be a simple type definition, see below) and the <restriction> alternative is chosen, then starting from either 1.1 the simple type definition corresponding to the <simpleType> among the [children] of <restriction> if there is one; 1.2 otherwise (<restriction> has no <simpleType> among its [children]), the simple type definition which is the {content type} of the type definition ·resolved· to by the ·actual value· of the `base` [attribute] a simple type definition which restricts that simple type definition with a set of facet components corresponding to the appropriate element information items among the <restriction>'s [children] (i.e. those which specify facets, if any), as defined in Simple Type Restriction (Facets) (§3.14.3); 2 otherwise if the type definition ·resolved· to by the ·actual value· of the `base` [attribute] is a complex type definition (whose own {content type} must be a simple type definition, see below) and the <extension> alternative is chosen, then the {content type} of that complex type definition; 3 otherwise (the type definition ·resolved· to by the ·actual value· of the `base` [attribute] is a simple type definition and the <extension> alternative is chosen), then that simple type definition.

When the <complexContent> alternative is chosen, the following elements are relevant (as are the <attributeGroup> and <anyAttribute> elements, not repeated here), and the additional property mappings are as below. Note that either <restriction> or <extension> must be chosen as the content of <complexContent>, but their content models are different in this case from the case above when they occur as children of <simpleContent>.

The property mappings below are *also* used in the case where the third alternative (neither <simpleContent> nor <complexContent>) is chosen. This case is understood as shorthand for complex content restricting the ·ur-type definition·, and the details of the mappings should be modified as necessary.

```
<complexContent
  id = ID
  mixed = boolean
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (restriction | extension))
</complexContent>
```

```
<restriction
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (group | all | choice | sequence)?, ((attribute |
attributeGroup)*, anyAttribute?))
</restriction>
```

```
<extension
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, ((group | all | choice | sequence)?, ((attribute |
attributeGroup)*, anyAttribute?)))
</extension>
```

Complex Type Definition with complex content Schema Component

Property	Representation
{base type definition}	The type definition ·resolved· to by the ·actual value· of the `base` [attribute]
{derivation method}	If the <restriction> alternative is chosen, then *restriction*, otherwise (the <extension> alternative is chosen) *extension*.
{attribute uses}	A union of sets of attribute uses as follows: 1 The set of attribute uses corresponding to the <attribute> [children], if any. 2 The {attribute uses} of the attribute groups ·resolved· to by the ·actual value·s of the `ref` [attribute] of the <attributeGroup> [children], if any. 3 The {attribute uses} of the type definition ·resolved· to by the ·actual value· of the `base` [attribute], unless the <restriction> alternative is chosen, in which case some members of that type definition's {attribute uses} may not be included, namely those whose {attribute declaration}'s {name} and {target namespace} are the same as **one** of the following: 3.1 The {name} and {target namespace} of the {attribute declaration} of an attribute use in the set per clause 1 or clause 2 above; 3.2 what would have been the {name} and {target namespace} of the {attribute declaration} of an attribute use in the set per clause 1 above but for the ·actual value· of the `use` [attribute] of the relevant <attribute> among the [children] of <restriction> being *prohibited*.
{attribute wildcard}	As above for the <simpleContent> alternative.
{content type}	The appropriate **case** among the following: 1 **If** the <restriction> alternative is chosen, **then** the appropriate **case** among the following: 1.1 **If one** of the following is true 1.1.1 There is no <group>, <all>, <choice> or <sequence> among the [children]; 1.1.2 There is an <all> or <sequence> among the [children] with no [children] of its own excluding <annotation>; 1.1.3 There is a <choice> among the [children] with no [children] of its own excluding <annotation> whose `minOccurs` [attribute] has the ·actual value· `0`; , **then** *empty*; 1.2 **otherwise** a pair consisting of 1.2.1 the appropriate **case** among the following: 1.2.1.1 **If** the `mixed` [attribute] is present on <complexContent>, **then** *mixed* if its ·actual value· is `true`, otherwise *elementOnly*; 1.2.1.2 **If** the `mixed` [attribute] is present on <complexType> and its ·actual value· is `true`, **then** *mixed*; 1.2.1.3 **otherwise** *elementOnly*. 1.2.2 The particle corresponding to the <all>, <choice>, <group> or <sequence> among the [children]. 2 **If** the <extension> alternative is chosen, **then** [Definition:] let the **explicit content** be *empty* if any of the sub-clauses of clause 1.1 above applies, otherwise the particle corresponding to the <all>,

<choice>, <group> or <sequence> among the [children], and then take the appropriate **case** among the following:

2.1 **If** the ·explicit content· is *empty*, **then** the {content type} of the type definition ·resolved· to by the ·actual value· of the `base` [attribute]

2.2 **If** the type definition ·resolved· to by the ·actual value· of the `base` [attribute] has a {content type} of *empty*, **then** a pair of *mixed* or *elementOnly* (determined as per clause 1.2.1 above) and the ·explicit content· itself;

2.3 **otherwise** a pair of *mixed* or *elementOnly* (determined as per clause 1.2.1 above) and a particle whose properties are as follows:

{min occurs}
1

{max occurs}
1

{term}
A model group whose {compositor} is *sequence* and whose {particles} are the particle of the {content type} of the type definition ·resolved· to by the ·actual value· of the `base` [attribute] followed by the ·explicit content·.

NOTE: Aside from the simple coherence requirements enforced above, constraining type definitions identified as restrictions to actually *be* restrictions, that is, to ·validate· a subset of the items which are ·validated· by their base type definition, is enforced in Constraints on Complex Type Definition Schema Components (§3.4.6).

NOTE: The *only* substantive function of the value *prohibited* for the `use` attribute of an <attribute> is in establishing the correspondence between a complex type defined by restriction and its XML representation. It serves to prevent inheritance of an identically named attribute use from the {base type definition}. Such an <attribute> does not correspond to any component, and hence there is no interaction with either explicit or inherited wildcards in the operation of Complex Type Definition Validation Rules (§3.4.4) or Constraints on Complex Type Definition Schema Components (§3.4.6).

Careful consideration of the above concrete syntax reveals that a type definition need consist of no more than a name, i.e. that `<complexType name="anyThing"/>` is allowed.

Example

```
<xs:complexType name="length1">
 <xs:simpleContent>
  <xs:extension base="xs:nonNegativeInteger">
   <xs:attribute name="unit" type="xs:NMTOKEN"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>

<xs:element name="width" type="length1"/>

  <width unit="cm">25</width>

<xs:complexType name="length2">
```

```
 <xs:complexContent>
  <xs:restriction base="xs:anyType">
   <xs:sequence>
    <xs:element name="size" type="xs:nonPositiveInteger"/>
    <xs:element name="unit" type="xs:NMTOKEN"/>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:element name="depth" type="length2"/>

  <depth>
   <size>25</size><unit>cm</unit>
  </depth>

<xs:complexType name="length3">
 <xs:sequence>
  <xs:element name="size" type="xs:non-positive-integer"/>
  <xs:element name="unit" type="xs:NMTOKEN"/>
 </xs:sequence>
</xs:complexType>
```

Three approaches to defining a type for length: one with character data content constrained by reference to a built-in datatype, and one attribute, the other two using two elements. `length3` is the abbreviated alternative to `length2`: they correspond to identical type definition components.

Example

```
<xs:complexType name="personName">
 <xs:sequence>
  <xs:element name="title" minOccurs="0"/>
  <xs:element name="forename" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="surname"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="extendedName">
 <xs:complexContent>
  <xs:extension base="personName">
   <xs:sequence>
    <xs:element name="generation" minOccurs="0"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="addressee" type="extendedName"/>

  <addressee>
   <forename>Albert</forename>
   <forename>Arnold</forename>
   <surname>Gore</surname>
   <generation>Jr</generation>
  </addressee>
```

A type definition for personal names, and a definition derived by extension which adds a single element; an element declaration referencing the derived definition, and a ·valid· instance thereof.

Example

```
<xs:complexType name="simpleName">
 <xs:complexContent>
  <xs:restriction base="personName">
   <xs:sequence>
    <xs:element name="forename" minOccurs="1" maxOccurs="1"/>
    <xs:element name="surname"/>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:element name="who" type="simpleName"/>

   <who>
    <forename>Bill</forename>
    <surname>Clinton</surname>
   </who>
```

A simplified type definition derived from the base type from the previous example by restriction, eliminating one optional daughter and fixing another to occur exactly once; an element declared by reference to it, and a ·valid· instance thereof.

Example

```
<xs:complexType name="paraType" mixed="true">
 <xs:choice minOccurs="0" maxOccurs="unbounded">
  <xs:element ref="emph"/>
  <xs:element ref="strong"/>
 </xs:choice>
 <xs:attribute name="version" type="xs:number"/>
</xs:complexType>
```

A further illustration of the abbreviated form, with the `mixed` attribute appearing on `complexType` itself.

3.4.3 Constraints on XML Representations of Complex Type Definitions

Schema Representation Constraint: Complex Type Definition Representation OK

In addition to the conditions imposed on <complexType> element information items by the schema for schemas, **all** of the following must be true:

1 If the <complexContent> alternative is chosen, the type definition ·resolved· to by the ·actual value· of the `base` [attribute] must be a complex type definition;

2 If the <simpleContent> alternative is chosen, the type definition ·resolved· to by the ·actual value· of the `base` [attribute] must be either a complex type definition whose {content type} is a simple type definition or, only if the <extension> alternative is also chosen, a simple type definition;

3 The corresponding complex type definition component must satisfy the conditions set out in Constraints on Complex Type Definition Schema Components (§3.4.6);

4 If clause 2.1 or clause 2.2 in the correspondence specification above for {attribute wildcard} is satisfied, the intensional intersection must be expressible, as defined in Attribute Wildcard Intersection (§3.10.6).

3.4.4 Complex Type Definition Validation Rules

Validation Rule: Element Locally Valid (Complex Type)

For an element information item to be locally ·valid· with respect to a complex type definition **all** of the following must be true:

1 {abstract} is *false*.

2 If clause 3.2 of Element Locally Valid (Element) (§3.3.4) did not apply, then the appropriate **case** among the following must be true:

2.1 **If** the {content type} is *empty*, **then** the element information item has no character or element

information item [children].

2.2 **If** the {content type} is a simple type definition, **then** the element information item has no element information item [children], and the ·normalized value· of the element information item is ·valid· with respect to that simple type definition as defined by String Valid (§3.14.4).

2.3 **If** the {content type} is *element-only*, **then** the element information item has no character information item [children] other than those whose [character code] is defined as a white space in [XML 1.0 (Second Edition)].

2.4 **If** the {content type} is *element-only* or *mixed*, **then** the sequence of the element information item's element information item [children], if any, taken in order, is ·valid· with respect to the {content type}'s particle, as defined in Element Sequence Locally Valid (Particle) (§3.9.4).

3 For each attribute information item in the element information item's [attributes] excepting those whose [namespace name] is identical to `http://www.w3.org/2001/XMLSchema-instance` and whose [local name] is one of `type`, `nil`, `schemaLocation` or `noNamespaceSchemaLocation`, the appropriate **case** among the following must be true:

3.1 **If** there is among the {attribute uses} an attribute use with an {attribute declaration} whose {name} matches the attribute information item's [local name] and whose {target namespace} is identical to the attribute information item's [namespace name] (where an ·absent· {target namespace} is taken to be identical to a [namespace name] with no value), **then** the attribute information must be ·valid· with respect to that attribute use as per Attribute Locally Valid (Use) (§3.5.4). In this case the {attribute declaration} of that attribute use is the ·context-determined declaration· for the attribute information item with respect to Schema-Validity Assessment (Attribute) (§3.2.4) and Assessment Outcome (Attribute) (§3.2.5).

3.2 **otherwise all** of the following must be true:

3.2.1 There must be an {attribute wildcard}.

3.2.2 The attribute information item must be ·valid· with respect to it as defined in Item Valid (Wildcard) (§3.10.4).

4 The {attribute declaration} of each attribute use in the {attribute uses} whose {required} is *true* matches one of the attribute information items in the element information item's [attributes] as per clause 3.1 above.

5 Let [Definition:] the **wild IDs** be the set of all attribute information item to which clause 3.2 applied and whose ·validation· resulted in a ·context-determined declaration· of *mustFind* or no ·context-determined declaration· at all, and whose [local name] and [namespace name] resolve (as defined by QName resolution (Instance) (§3.15.4)) to an attribute declaration whose {type definition} is or is derived from ID. Then **all** of the following must be true:

5.1 There must be no more than one item in ·wild IDs·.

5.2 If ·wild IDs· is non-empty, there must not be any attribute uses among the {attribute uses} whose {attribute declaration}'s {type definition} is or is derived from ID.

> **NOTE:** This clause serves to ensure that even via attribute wildcards no element has more than one attribute of type ID, and that even when an element legitimately lacks a declared attribute of type ID, a wildcard-validated attribute must not supply it. That is, if an element has a type whose attribute declarations include one of type ID, it either has that attribute or no attribute of type ID.

NOTE: When an {attribute wildcard} is present, this does *not* introduce any ambiguity with respect to how attribute information items for which an attribute use is present amongst the {attribute uses} whose name and target namespace match are ·assessed·. In such cases the attribute use *always* takes precedence, and the ·assessment· of such items stands or falls entirely on the basis of the attribute use and its {attribute declaration}. This follows from the details of clause 3.

3.4.5 Complex Type Definition Information Set Contributions

Schema Information Set Contribution: Attribute Default Value

For each attribute use in the {attribute uses} whose {required} is *false* and whose {value constraint} is

not ·absent· but whose {attribute declaration} does not match one of the attribute information items in the element information item's [attributes] as per clause 3.1 of Element Locally Valid (Complex Type) (§3.4.4) above, the post-schema-validation infoset has an attribute information item whose properties are as below added to the [attributes] of the element information item.

[local name]
The {attribute declaration}'s {name}.
[namespace name]
The {attribute declaration}'s {target namespace}.
[schema normalized value]
The canonical lexical representation of the {value constraint} value.
[schema default]
The canonical lexical representation of the {value constraint} value.
[validation context]
The nearest ancestor element information item with a [schema information] property.
[validity]
valid.
[validation attempted]
full.
[schema specified]
schema.

The added items should also either have [type definition] (and [member type definition] if appropriate) properties, or their lighter-weight alternatives, as specified in Attribute Validated by Type (§3.2.5).

3.4.6 Constraints on Complex Type Definition Schema Components

All complex type definitions (see Complex Type Definitions (§3.4)) must satisfy the following constraints.

Schema Component Constraint: Complex Type Definition Properties Correct
All of the following must be true:
1 The values of the properties of a complex type definition must be as described in the property tableau in The Complex Type Definition Schema Component (§3.4.1), modulo the impact of Missing Sub-components (§5.3).
2 If the {base type definition} is a simple type definition, the {derivation method} must be *extension*.
3 Circular definitions are disallowed, except for the ·ur-type definition·. That is, it must be possible to reach the ·ur-type definition· by repeatedly following the {base type definition}.
4 Two distinct attribute declarations in the {attribute uses} must not have identical {name}s and {target namespace}s.
5 Two distinct attribute declarations in the {attribute uses} must not have {type definition}s which are or are derived from ID.

Schema Component Constraint: Derivation Valid (Extension)
If the {derivation method} is *extension*, the appropriate **case** among the following must be true:
1 **If** the {base type definition} is a complex type definition, **then all** of the following must be true:
1.1 The {final} of the {base type definition} must not contain *extension*.
1.2 Its {attribute uses} must be a subset of the {attribute uses} of the complex type definition itself, that is, for every attribute use in the {attribute uses} of the {base type definition}, there must be an attribute use in the {attribute uses} of the complex type definition itself whose {attribute declaration} has the same {name}, {target namespace} and {type definition} as its attribute declaration.
1.3 If it has an {attribute wildcard}, the complex type definition must also have one, and the base type definition's {attribute wildcard}'s {namespace constraint} must be a subset of the complex type definition's {attribute wildcard}'s {namespace constraint}, as defined by Wildcard Subset

(§3.10.6).

1.4 **One** of the following must be true:

1.4.1 The {content type} of the {base type definition} and the {content type} of the complex type definition itself must be the same simple type definition.

1.4.2 **All** of the following must be true:

1.4.2.1 The {content type} of the complex type definition itself must specify a particle.

1.4.2.2 **One** of the following must be true:

1.4.2.2.1 The {content type} of the {base type definition} must be *empty*.

1.4.2.2.2 **All** of the following must be true:

1.4.2.2.2.1 Both {content type}s must be *mixed* or both must be *element-only*.

1.4.2.2.2.2 The particle of the complex type definition must be a ·valid extension· of the {base type definition}'s particle, as defined in Particle Valid (Extension) (§3.9.6).

1.5 It must in principle be possible to derive the complex type definition in two steps, the first an extension and the second a restriction (possibly vacuous), from that type definition among its ancestors whose {base type definition} is the ·ur-type definition·.

NOTE: This requirement ensures that nothing removed by a restriction is subsequently added back by an extension. It is trivial to check if the extension in question is the only extension in its derivation, or if there are no restrictions bar the first from the ·ur-type definition·.

Constructing the intermediate type definition to check this constraint is straightforward: simply re-order the derivation to put all the extension steps first, then collapse them into a single extension. If the resulting definition can be the basis for a valid restriction to the desired definition, the constraint is satisfied.

2 **If** the {base type definition} is a simple type definition, **then all** of the following must be true:

2.1 The {content type} must be the same simple type definition.

2.2 The {final} of the {base type definition} must not contain *extension*.

[Definition:] If this constraint Derivation Valid (Extension) (§3.4.6) holds of a complex type definition, it is a **valid extension** of its {base type definition}.

Schema Component Constraint: Derivation Valid (Restriction, Complex)

If the {derivation method} is *restriction* **all** of the following must be true:

1 The {base type definition} must be a complex type definition whose {final} does not contain *restriction*.

2 For each attribute use (call this **R**) in the {attribute uses} the appropriate **case** among the following must be true:

2.1 **If** there is an attribute use in the {attribute uses} of the {base type definition} (call this **B**) whose {attribute declaration} has the same {name} and {target namespace}, **then all** of the following must be true:

2.1.1 **one** of the following must be true:

2.1.1.1 **B**'s {required} is *false*.

2.1.1.2 **R**'s {required} is *true*.

2.1.2 **R**'s {attribute declaration}'s {type definition} must be validly derived from **B**'s {type definition} given the empty set as defined in Type Derivation OK (Simple) (§3.14.6).

2.1.3 [Definition:] Let the **effective value constraint** of an attribute use be its {value constraint}, if present, otherwise its {attribute declaration}'s {value constraint} . Then **one** of the following must be true:

2.1.3.1 **B**'s ·effective value constraint· is ·absent· or *default*.

2.1.3.2 **R**'s ·effective value constraint· is *fixed* with the same string as **B**'s.

2.2 **otherwise** the {base type definition} must have an {attribute wildcard} and the {target namespace} of the **R**'s {attribute declaration} must be ·valid· with respect to that wildcard, as defined in Wildcard allows Namespace Name (§3.10.4).

3 For each attribute use in the {attribute uses} of the {base type definition} whose {required} is *true*, there must be an attribute use with an {attribute declaration} with the same {name} and {target

namespace} as its {attribute declaration} in the {attribute uses} of the complex type definition itself whose {required} is *true*.

4 If there is an {attribute wildcard}, **all** of the following must be true:

4.1 The {base type definition} must also have one.

4.2 The complex type definition's {attribute wildcard}'s {namespace constraint} must be a subset of the {base type definition}'s {attribute wildcard}'s {namespace constraint}, as defined by Wildcard Subset (§3.10.6).

5 The appropriate **case** among the following must be true:

5.1 **If** the {content type} of the complex type definition is a simple type definition, **then one** of the following must be true:

5.1.1 The {content type} of the {base type definition} must be a simple type definition of which the {content type} is a ·valid restriction· as defined in Derivation Valid (Restriction, Simple) (§3.14.6).

5.1.2 The {base type definition} must be *mixed* and have a particle which is ·emptiable· as defined in Particle Emptiable (§3.9.6).

5.2 **If** the {content type} of the complex type itself is *empty* , **then one** of the following must be true:

5.2.1 The {content type} of the {base type definition} must also be *empty*.

5.2.2 The {content type} of the {base type definition} must be *elementOnly* or *mixed* and have a particle which is ·emptiable· as defined in Particle Emptiable (§3.9.6).

5.3 **If** the {content type} of the {base type definition} is *mixed* or the {content type} of the complex type definition itself is *element-only*, **then** the particle of the complex type definition itself must be a ·valid restriction· of the particle of the {content type} of the {base type definition} as defined in Particle Valid (Restriction) (§3.9.6).

[Definition:] If this constraint Derivation Valid (Restriction, Complex) (§3.4.6) holds of a complex type definition, it is a **valid restriction** of its {base type definition}.

NOTE: To restrict a complex type definition with a simple base type definition to *empty*, use a simple type definition with a *fixed* value of the empty string: this preserves the type information.

The following constraint defines a relation appealed to elsewhere in this specification.

Schema Component Constraint: Type Derivation OK (Complex)

For a complex type definition (call it **D**, for derived) to be validly derived from a type definition (call this **B**, for base) given a subset of {*extension*, *restriction*} **all** of the following must be true:

1 If **B** and **D** are not the same type definition, then the {derivation method} of **D** must not be in the subset.

2 **One** of the following must be true:

2.1 **B** and **D** must be the same type definition.

2.2 **B** must be **D**'s {base type definition}.

2.3 **All** of the following must be true:

2.3.1 **D**'s {base type definition} must not be the ·ur-type definition·.

2.3.2 The appropriate **case** among the following must be true:

2.3.2.1 **If D**'s {base type definition} is complex, **then** it must be validly derived from **B** given the subset as defined by this constraint.

2.3.2.2 **If D**'s {base type definition} is simple, **then** it must be validly derived from **B** given the subset as defined in Type Derivation OK (Simple) (§3.14.6).

NOTE: This constraint is used to check that when someone uses a type in a context where another type was expected (either via `xsi:type` or substitution groups), that the type used is actually derived from the expected type, and that that derivation does not involve a form of derivation which was ruled out by the expected type.

3.4.7 Built-in Complex Type Definition

There is a complex type definition nearly equivalent to the ·ur-type definition· present in every schema by definition. It has the following properties:

Complex Type Definition of the Ur-Type

Property	Value
{name}	anyType
{target namespace}	http://www.w3.org/2001/XMLSchema
{base type definition}	Itself
{derivation method}	*restriction*
{content type}	A pair consisting of *mixed* and a particle with the following properties:

Property	Value
{min occurs}	1
{max occurs}	1
{term}	a model group with the following properties:

Property	Value
{compositor}	*sequence*
{particles}	a list containing one particle with the following properties:

Property	Value
{min occurs}	0
{max occurs}	*unbounded*
{term}	a wildcard with an *any* {namespace constraint}

{attribute uses}	The empty set
{attribute wildcard}	{namespace constraint} is *any*
{final}	The empty set
{prohibited substitutions}	The empty set
{abstract}	*false*

The `mixed` content specification together with the unconstrained wildcard content model and attribute specification produce the defining property for the ·ur-type definition·, namely that *every* complex type definition is (eventually) a restriction of the ·ur-type definition·: its permissions and requirements are the least restrictive possible.

> **NOTE:** This specification does not provide an inventory of built-in complex type definitions for use in user schemas. A preliminary library of complex type definitions is available which includes both mathematical (e.g. `rational`) and utility (e.g. `array`) type definitions. In particular, there is a `text` type definition which is recommended for use as the type definition in element declarations intended for general text content, as it makes sensible provision for various aspects of internationalization. For more details, see the schema document for the type library at its namespace name: http://www.w3.org/2001/03/XMLSchema/TypeLibrary.xsd.

3.5 AttributeUses

An attribute use is a utility component which controls the occurrence and defaulting behavior of attribute declarations. It plays the same role for attribute declarations in complex types that particles play for element declarations.

Example

```
<xs:complexType>
 . . .
 <xs:attribute ref="xml:lang" use="required"/>
 <xs:attribute ref="xml:space" default="preserve"/>
 <xs:attribute name="version" type="xs:number" fixed="1.0"/>
</xs:complexType>
```

XML representations which all involve attribute uses, illustrating some of the possibilities for controlling occurrence.

3.5.1 The Attribute Use Schema Component

The attribute use schema component has the following properties:

Schema Component: Attribute Use

{required}
A boolean.
{attribute declaration}
An attribute declaration.
{value constraint}
Optional. A pair consisting of a value and one of *default*, *fixed*.

{required} determines whether this use of an attribute declaration requires an appropriate attribute information item to be present, or merely allows it.

{attribute declaration} provides the attribute declaration itself, which will in turn determine the simple type definition used.

{value constraint} allows for local specification of a default or fixed value. This must be consistent with that of the {attribute declaration}, in that if the {attribute declaration} specifies a fixed value, the only allowed {value constraint} is the same fixed value.

3.5.2 XML Representation of Attribute Use Components

Attribute uses correspond to all uses of <attribute> which allow a `use` attribute. These in turn correspond to *two* components in each case, an attribute use and its {attribute declaration} (although note the latter is not new when the attribute use is a reference to a top-level attribute declaration). The appropriate mapping is described in XML Representation of Attribute Declaration Schema Components (§3.2.2).

3.5.3 Constraints on XML Representations of Attribute Uses

None as such.

3.5.4 Attribute Use Validation Rules

Validation Rule: Attribute Locally Valid (Use)

For an attribute information item to be·valid· with respect to an attribute use its ·normalized value· must match the canonical lexical representation of the attribute use's {value constraint} value, if it is present and *fixed*.

3.5.5 Attribute Use Information Set Contributions

None as such.

3.5.6 Constraints on Attribute Use Schema Components

All attribute uses (see AttributeUses (§3.5)) must satisfy the following constraints.

Schema Component Constraint: Attribute Use Correct

All of the following must be true:

1 The values of the properties of an attribute use must be as described in the property tableau in The Attribute Use Schema Component (§3.5.1), modulo the impact of Missing Sub-components (§5.3).

2 If the {attribute declaration} has a *fixed* {value constraint}, then if the attribute use itself has a {value constraint}, it must also be *fixed* and its value must match that of the {attribute declaration}'s {value constraint}.

3.6 Attribute Group Definitions

A schema can name a group of attribute declarations so that they may be incorporated as a group into complex type definitions.

Attribute group definitions do not participate in ·validation· as such, but the {attribute uses} and {attribute wildcard} of one or more complex type definitions may be constructed in whole or part by reference to an attribute group. Thus, attribute group definitions provide a replacement for some uses of XML's parameter entity facility. Attribute group definitions are provided primarily for reference from the XML representation of schema components (see <complexType> and <attributeGroup>).

Example

```
<xs:attributeGroup name="myAttrGroup">
    <xs:attribute . . ./>
    . . .
</xs:attributeGroup>

<xs:complexType name="myelement">
    . . .
    <xs:attributeGroup ref="myAttrGroup"/>
</xs:complexType>
```

XML representations for attribute group definitions. The effect is as if the attribute declarations in the

group were present in the type definition.

3.6.1 The Attribute Group Definition Schema Component

The attribute group definition schema component has the following properties:

Schema Component: Attribute Group Definition

{name}
An NCName as defined by [XML-Namespaces].

{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].

{attribute uses}
A set of attribute uses.

{attribute wildcard}
Optional. A wildcard.

{annotation}
Optional. An annotation.

Attribute groups are identified by their {name} and {target namespace}; attribute group identities must be unique within an ·XML Schema·. See References to schema components across namespaces (§4.2.3) for the use of component identifiers when importing one schema into another.

{attribute uses} is a set attribute uses, allowing for local specification of occurrence and default or fixed values.

{attribute wildcard} provides for an attribute wildcard to be included in an attribute group. See above under Complex Type Definitions (§3.4) for the interpretation of attribute wildcards during ·validation·.

See Annotations (§3.13) for information on the role of the {annotation} property.

3.6.2 XML Representation of Attribute Group Definition Schema Components

The XML representation for an attribute group definition schema component is an <attributeGroup> element information item. It provides for naming a group of attribute declarations and an attribute wildcard for use by reference in the XML representation of complex type definitions and other attribute group definitions. The correspondences between the properties of the information item and properties of the component it corresponds to are as follows:

XML Representation Summary: attributeGroup Element Information Item

```
<attributeGroup
  id = ID
  name = NCName
  ref = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, ((attribute | attributeGroup)*, anyAttribute?))
</attributeGroup>
```

When an <attributeGroup> appears as a daughter of <schema> or <redefine>, it corresponds to an attribute group definition as below. When it appears as a daughter of <complexType> or <attributeGroup>, it does not correspond to any component as such.

Attribute Group Definition Schema Component

Property	Representation
{name}	The ·actual value· of the `name` [attribute]
{target namespace}	The ·actual value· of the `targetNamespace` [attribute] of the parent `schema` element information item.
{attribute uses}	The union of the set of attribute uses corresponding to the <attribute> [children], if any, with the {attribute uses} of the attribute groups ·resolved· to by the ·actual value·s of the `ref` [attribute] of the <attributeGroup> [children], if any.
{attribute wildcard}	As for the ·complete wildcard· as described in XML Representation of Complex Type Definitions (§3.4.2).
{annotation}	The annotation corresponding to the <annotation> element information item in the [children], if present, otherwise ·absent·.

The example above illustrates a pattern which recurs in the XML representation of schemas: The same element, in this case `attributeGroup`, serves both to define and to incorporate by reference. In the first case the `name` attribute is required, in the second the `ref` attribute is required, and the element must be empty. These two are mutually exclusive, and also conditioned by context: the defining form, with a `name`, must occur at the top level of a schema, whereas the referring form, with a `ref`, must occur within a complex type definition or an attribute group definition.

3.6.3 Constraints on XML Representations of Attribute Group Definitions

Schema Representation Constraint: Attribute Group Definition Representation OK

In addition to the conditions imposed on <attributeGroup> element information items by the schema for schemas, **all** of the following must be true:

1 The corresponding attribute group definition, if any, must satisfy the conditions set out in Constraints on Attribute Group Definition Schema Components (§3.6.6).

2 If clause 2.1 or clause 2.2 in the correspondence specification in XML Representation of Complex Type Definitions (§3.4.2) for {attribute wildcard}, as referenced above, is satisfied, the intensional intersection must be expressible, as defined in Attribute Wildcard Intersection (§3.10.6).

3 Circular group reference is disallowed outside <redefine>. That is, unless this element information item's parent is <redefine>, then among the [children], if any, there must not be an <attributeGroup> with `ref` [attribute] which resolves to the component corresponding to this <attributeGroup>.

3.6.4 Attribute Group Definition Validation Rules

None as such.

3.6.5 Attribute Group Definition Information Set Contributions

None as such.

3.6.6 Constraints on Attribute Group Definition Schema Components

All attribute group definitions (see Attribute Group Definitions (§3.6)) must satisfy the following constraint.

Schema Component Constraint: Attribute Group Definition Properties Correct

All of the following must be true:

1 The values of the properties of an attribute group definition must be as described in the property tableau in The Attribute Group Definition Schema Component (§3.6.1), modulo the impact of Missing Sub-components (§5.3);

2 Two distinct members of the {attribute uses} must not have {attribute declaration}s both of whose

{name}s match and whose {target namespace}s are identical.
3 Two distinct members of the {attribute uses} must not have {attribute declaration}s both of whose {type definition}s are or are derived from ID.

3.7 Model Group Definitions

A model group definition associates a name and optional annotations with a Model Group (§2.2.3.1). By reference to the name, the entire model group can be incorporated by reference into a {term}.

Model group definitions are provided primarily for reference from the XML Representation of Complex Type Definitions (§3.4.2) (see <complexType> and <group>). Thus, model group definitions provide a replacement for some uses of XML's parameter entity facility.

Example

```
<xs:group name="myModelGroup">
 <xs:sequence>
  <xs:element ref="someThing"/>
  . . .
 </xs:sequence>
</xs:group>

<xs:complexType name="trivial">
 <xs:group ref="myModelGroup"/>
 <xs:attribute .../>
</xs:complexType>

<xs:complexType name="moreSo">
 <xs:choice>
  <xs:element ref="anotherThing"/>
  <xs:group ref="myModelGroup"/>
 </xs:choice>
 <xs:attribute .../>
</xs:complexType>
```

A minimal model group is defined and used by reference, first as the whole content model, then as one alternative in a choice.

3.7.1 The Model Group Definition Schema Component

The model group definition schema component has the following properties:

Schema Component: Model Group Definition

{name}
An NCName as defined by [XML-Namespaces].
{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].
{model group}
A model group.
{annotation}

Optional. An annotation.

Model group definitions are identified by their {name} and {target namespace}; model group identities must be unique within an ·XML Schema·. See References to schema components across namespaces (§4.2.3) for the use of component identifiers when importing one schema into another.

Model group definitions *per se* do not participate in ·validation·, but the {term} of a particle may correspond in whole or in part to a model group from a model group definition.

{model group} is the Model Group (§2.2.3.1) for which the model group definition provides a name.

See Annotations (§3.13) for information on the role of the {annotation} property.

3.7.2 XML Representation of Model Group Definition Schema Components

The XML representation for a model group definition schema component is a <group> element information item. It provides for naming a model group for use by reference in the XML representation of complex type definitions and model groups. The correspondences between the properties of the information item and properties of the component it corresponds to are as follows:

XML Representation Summary: `group` Element Information Item

```
<group
  name = NCName>
  Content: (annotation?, (all | choice | sequence))
</group>
```

If there is a `name` [attribute] (in which case the item will have <schema> or <redefine> as parent), then the item corresponds to a model group definition component with properties as follows:

Model Group Definition Schema Component

Property	Representation
{name}	The ·actual value· of the `name` [attribute]
{target namespace}	The ·actual value· of the `targetNamespace` [attribute] of the parent `schema` element information item.
{model group}	A model group which is the {term} of a particle corresponding to the <all>, <choice> or <sequence> among the [children] (there must be one).
{annotation}	The annotation corresponding to the <annotation> element information item in the [children], if present, otherwise ·absent·.

Otherwise, the item will have a `ref` [attribute], in which case it corresponds to a particle component with properties as follows (unless `minOccurs=maxOccurs=0`, in which case the item corresponds to no component at all):

Particle Schema Component

Property	Representation
{min occurs}	The ·actual value· of the `minOccurs` [attribute], if present, otherwise `1`.
{max occurs}	*unbounded*, if the `maxOccurs` [attribute] equals *unbounded*, otherwise the ·actual value· of the `maxOccurs` [attribute], if present, otherwise `1`.
{term}	The {model group} of the model group definition ·resolved· to by the ·actual value· of the `ref` [attribute]

The name of this section is slightly misleading, in that the second, un-named, case above (with a `ref` and no `name`) is not really a named model group at all, but a reference to one. Also note that in the first (named) case above no reference is made to `minOccurs` or `maxOccurs`: this is because the schema for schemas does not allow them on the child of <group> when it is named. This in turn is because the {min occurs} and {max occurs} of the particles which *refer* to the definition are what count.

Given the constraints on its appearance in content models, an <all> should only occur as the only item in the [children] of a named model group definition or a content model: see Constraints on Model Group Schema Components (§3.8.6).

3.7.3 Constraints on XML Representations of Model Group Definitions

Schema Representation Constraint: Model Group Definition Representation OK
In addition to the conditions imposed on <group> element information items by the schema for schemas, the corresponding model group definition, if any, must satisfy the conditions set out in Constraints on Model Group Schema Components (§3.8.6).

3.7.4 Model Group Definition Validation Rules

None as such.

3.7.5 Model Group Definition Information Set Contributions

None as such.

3.7.6 Constraints on Model Group Definition Schema Components

All model group definitions (see Model Group Definitions (§3.7)) must satisfy the following constraint.

Schema Component Constraint: Model Group Definition Properties Correct
The values of the properties of a model group definition must be as described in the property tableau in The Model Group Definition Schema Component (§3.7.1), modulo the impact of Missing Sub-components (§5.3).

3.8 Model Groups

When the [children] of element information items are not constrained to be *empty* or by reference to a simple type definition (Simple Type Definitions (§3.14)), the sequence of element information item [children] content may be specified in more detail with a model group. Because the {term} property of a

particle can be a model group, and model groups contain particles, model groups can indirectly contain other model groups; the grammar for content models is therefore recursive.

Example

```
<xs:all>
 <xs:element ref="cats"/>
 <xs:element ref="dogs"/>
</xs:all>

<xs:sequence>
 <xs:choice>
  <xs:element ref="left"/>
  <xs:element ref="right"/>
 </xs:choice>
 <xs:element ref="landmark"/>
</xs:sequence>
```

XML representations for the three kinds of model group, the third nested inside the second.

3.8.1 The Model Group Schema Component

The model group schema component has the following properties:

Schema Component: Model Group

{compositor}
One of *all*, *choice* or *sequence*.
{particles}
A list of particles
{annotation}
Optional. An annotation.

specifies a sequential (*sequence*), disjunctive (*choice*) or conjunctive (*all*) interpretation of the {particles}. This in turn determines whether the element information item [children] ·validated· by the model group must:

- (*sequence*) correspond, in order, to the specified {particles};
- (*choice*) corresponded to exactly one of the specified {particles};
- (*all*) contain all and only exactly zero or one of each element specified in {particles}. The elements can occur in any order. In this case, to reduce implementation complexity, {particles} is restricted to contain local and top-level element declarations only, with {min occurs}=0 or 1, {max occurs}=1.

When two or more particles contained directly or indirectly in the {particles} of a model group have identically named element declarations as their {term}, the type definitions of those declarations must be the same. By 'indirectly' is meant particles within the {particles} of a group which is itself the {term} of a directly contained particle, and so on recursively.

See Annotations (§3.13) for information on the role of the {annotation} property.

3.8.2 XML Representation of Model Group Schema Components

The XML representation for a model group schema component is either an <all>, a <choice> or a <sequence> element information item. The correspondences between the properties of those information items and properties of the component they correspond to are as follows:

XML Representation Summary: `all` Element Information Item

```
<all
  id = ID
  maxOccurs = 1 : 1
  minOccurs = (0 | 1) : 1
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, element*)
</all>
```

```
<choice
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded)  : 1
  minOccurs = nonNegativeInteger : 1
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (element | group | choice | sequence | any)*)
</choice>
```

```
<sequence
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded)  : 1
  minOccurs = nonNegativeInteger : 1
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (element | group | choice | sequence | any)*)
</sequence>
```

Each of the above items corresponds to a particle containing a model group, with properties as follows (unless `minOccurs=maxOccurs=0`, in which case the item corresponds to no component at all):

Particle Schema Component

Property	Representation
{min occurs}	The ·actual value· of the `minOccurs` [attribute], if present, otherwise `1`.
{max occurs}	*unbounded*, if the `maxOccurs` [attribute] equals *unbounded*, otherwise the ·actual value· of the `maxOccurs` [attribute], if present, otherwise `1`.
{term}	A model group as given below:

Model Group Schema Component

Property	Representation
{compositor}	One of *all*, *choice*, *sequence* depending on the element information item.
{particles}	A sequence of particles corresponding to all the <all>, <choice>, <sequence>, <any>, <group> or <element> items among the [children], in order.
{annotation}	The annotation corresponding to the <annotation> element information item in the [children], if present, otherwise ·absent·.

3.8.3 Constraints on XML Representations of Model Groups

Schema Representation Constraint: Model Group Representation OK

In addition to the conditions imposed on <all>, <choice> and <sequence> element information items by the schema for schemas, the corresponding particle and model group must satisfy the conditions set out in Constraints on Model Group Schema Components (§3.8.6) and Constraints on Particle Schema Components (§3.9.6).

3.8.4 Model Group Validation Rules

Validation Rule: Element Sequence Valid

[Definition:] Define a **partition** of a sequence as a sequence of sub-sequences, some or all of which may be empty, such that concatenating all the sub-sequences yields the original sequence.

For a sequence (possibly empty) of element information items to be locally ·valid· with respect to a model group the appropriate **case** among the following must be true:

1 **If** the {compositor} is *sequence*, **then** there must be a ·partition· of the sequence into n sub-sequences where n is the length of {particles} such that each of the sub-sequences in order is ·valid· with respect to the corresponding particle in the {particles} as defined in Element Sequence Locally Valid (Particle) (§3.9.4).

2 **If** the {compositor} is *choice*, **then** there must be a particle among the {particles} such that the sequence is ·valid· with respect to that particle as defined in Element Sequence Locally Valid (Particle) (§3.9.4).

3 **If** the {compositor} is *all*, **then** there must be a ·partition· of the sequence into n sub-sequences where n is the length of {particles} such that there is a one-to-one mapping between the sub-sequences and the {particles} where each sub-sequence is ·valid· with respect to the corresponding particle as defined in Element Sequence Locally Valid (Particle) (§3.9.4).

Nothing in the above should be understood as ruling out groups whose {particles} is empty: although no sequence can be ·valid· with respect to such a group whose {compositor} is *choice*, the empty sequence *is* ·valid· with respect to empty groups whose {compositor} is *sequence* or *all*.

> **NOTE:** The above definition is implicitly non-deterministic, and should not be taken as a recipé for implementations. Note in particular that when {compositor} is *all*, particles is restricted to a list of local and top-level element declarations (see Constraints on Model Group Schema Components (§3.8.6)). A much simpler implementation is possible than would arise from a literal interpretation of the definition above; informally, the content is ·valid· when each declared element occurs exactly once (or at most once, if {min occurs} is `0`), and each is ·valid· with respect to its corresponding declaration. The elements can occur in arbitrary order.

3.8.5 Model Group Information Set Contributions

None as such.

3.8.6 Constraints on Model Group Schema Components

All model groups (see Model Groups (§3.8)) must satisfy the following constraints.

Schema Component Constraint: Model Group Correct

All of the following must be true:

1 The values of the properties of a model group must be as described in the property tableau in The Model Group Schema Component (§3.8.1), modulo the impact of Missing Sub-components (§5.3).

2 Circular groups are disallowed. That is, within the {particles} of a group there must not be at any depth a particle whose {term} is the group itself.

Schema Component Constraint: All Group Limited

When a model group has {compositor} *all* **all** of the following must be true:

1 **one** of the following must be true:

1.1 It appears as the model group of a model group definition.

1.2 It appears in a particle with {min occurs}={max occurs}=`1`, and that particle must be part of a pair which constitutes the {content type} of a complex type definition.

2 The {max occurs} of all the particles in the {particles} of the group must be `0` or `1`.

Schema Component Constraint: Element Declarations Consistent

If the {particles} contains, either directly, indirectly (that is, within the {particles} of a contained model group, recursively) or ·implicitly· two or more element declaration particles with the same {name} and {target namespace}, then all their type definitions must be the same top-level definition, that is, **all** of the following must be true:

1 all their {type definition}s must have a non-·absent· name.
2 all their {type definition}s must have the same name.
3 all their {type definition}s must have the same target namespace.

[Definition:] A list of particles **implicitly contains** an element declaration if a member of the list contains that element declaration in its ·substitution group·.

Schema Component Constraint: Unique Particle Attribution

A content model must be formed such that during ·validation· of an element information item sequence, the particle contained directly, indirectly or ·implicitly· therein with which to attempt to ·validate· each item in the sequence in turn can be uniquely determined without examining the content or attributes of that item, and without any information about the items in the remainder of the sequence.

> **NOTE:** This constraint reconstructs for XML Schema the equivalent constraints of [XML 1.0 (Second Edition)] and SGML. Given the presence of element substitution groups and wildcards, the concise expression of this constraint is difficult, see Analysis of the Unique Particle Attribution Constraint (non-normative) (§H) for further discussion.

> **NOTE:** Because locally-scoped element declarations may or may not have a {target namespace}, the scope of declarations is *not* relevant to enforcing either of the two preceding constraints.

The following constraints define relations appealed to elsewhere in this specification.

Schema Component Constraint: Effective Total Range (all and sequence)

The effective total range of a particle whose {term} is a group whose {compositor} is *all* or *sequence* is a pair of minimum and maximum, as follows:

minimum
The product of the particle's {min occurs} and the sum of the {min occurs} of every wildcard or element declaration particle in the group's {particles} and the minimum part of the effective total range of each of the group particles in the group's {particles} (or `0` if there are no {particles}).

maximum
unbounded if the {max occurs} of any wildcard or element declaration particle in the group's {particles} or the maximum part of the effective total range of any of the group particles in the group's {particles} is *unbounded*, or if any of those is non-zero and the {max occurs} of the particle itself is *unbounded*, otherwise the product of the particle's {max occurs} and the sum of the {max occurs} of every wildcard or element declaration particle in the group's {particles} and the maximum part of the effective total range of each of the group particles in the group's {particles} (or `0` if there are no {particles}).

Schema Component Constraint: Effective Total Range (choice)

The effective total range of a particle whose {term} is a group whose {compositor} is *choice* is a pair of minimum and maximum, as follows:

minimum
The product of the particle's {min occurs} and the minimum of the {min occurs} of every wildcard or element declaration particle in the group's {particles} and the minimum part of the effective total range of each of the group particles in the group's {particles} (or `0` if there are no {particles}).

maximum

unbounded if the {max occurs} of any wildcard or element declaration particle in the group's {particles} or the maximum part of the effective total range of any of the group particles in the group's {particles} is *unbounded*, or if any of those is non-zero and the {max occurs} of the particle itself is *unbounded*, otherwise the product of the particle's {max occurs} and the maximum of the {max occurs} of every wildcard or element declaration particle in the group's {particles} and the maximum part of the effective total range of each of the group particles in the group's {particles} (or `0` if there are no {particles}).

3.9 Particles

As described in Model Groups (§3.8), particles contribute to the definition of content models.

Example

```
<xs:element ref="egg" minOccurs="12" maxOccurs="12"/>

<xs:group ref="omelette" minOccurs="0"/>

<xs:any maxOccurs="unbounded"/>
```

XML representations which all involve particles, illustrating some of the possibilities for controlling occurrence.

3.9.1 The Particle Schema Component

The particle schema component has the following properties:

Schema Component: Particle

{min occurs}
A non-negative integer.
{max occurs}
Either a non-negative integer or *unbounded*.
{term}
One of a model group, a wildcard, or an element declaration.

In general, multiple element information item [children], possibly with intervening character [children] if the content type is *mixed*, can be ·validated· with respect to a single particle. When the {term} is an element declaration or wildcard, {min occurs} determines the minimum number of such element [children] that can occur. The number of such children must be greater than or equal to {min occurs}. If {min occurs} is *0*, then occurrence of such children is optional.

Again, when the {term} is an element declaration or wildcard, the number of such element [children] must be less than or equal to any numeric specification of {max occurs}; if {max occurs} is *unbounded*, then there is no upper bound on the number of such children.

When the {term} is a model group, the permitted occurrence range is determined by a combination of {min occurs} and {max occurs} and the occurrence ranges of the {term}'s {particles}.

3.9.2 XML Representation of Particle Components

Particles correspond to all three elements (<element> not immediately within <schema>, <group> not immediately within <schema> and <any>) which allow `minOccurs` and `maxOccurs` attributes. These in turn correspond to *two* components in each case, a particle and its {term}. The appropriate mapping is described in XML Representation of Element Declaration Schema Components (§3.3.2), XML Representation of Model Group Schema Components (§3.8.2) and XML Representation of Wildcard Schema Components (§3.10.2) respectively.

3.9.3 Constraints on XML Representations of Particles

None as such.

3.9.4 Particle Validation Rules

Validation Rule: Element Sequence Locally Valid (Particle)

For a sequence (possibly empty) of element information items to be locally ·valid· with respect to a particle the appropriate **case** among the following must be true:

1 **If** the {term} is a wildcard, **then all** of the following must be true:

1.1 The length of the sequence must be greater than or equal to the {min occurs}.

1.2 If {max occurs} is a number, the length of the sequence must be less than or equal to the {max occurs}.

1.3 Each element information item in the sequence must be ·valid· with respect to the wildcard as defined by Item Valid (Wildcard) (§3.10.4).

2 **If** the {term} is an element declaration, **then all** of the following must be true:

2.1 The length of the sequence must be greater than or equal to the {min occurs}.

2.2 If {max occurs} is a number, the length of the sequence must be less than or equal to the {max occurs}.

2.3 For each element information item in the sequence **one** of the following must be true:

2.3.1 The element declaration is local (i.e. its {scope} must not be *global*), its {abstract} is *false*, the element information item's [namespace name] is identical to the element declaration's {target namespace} (where an ·absent· {target namespace} is taken to be identical to a [namespace name] with no value) and the element information item's [local name] matches the element declaration's {name}.

In this case the element declaration is the ·context-determined declaration· for the element information item with respect to Schema-Validity Assessment (Element) (§3.3.4) and Assessment Outcome (Element) (§3.3.5).

2.3.2 The element declaration is top-level (i.e. its {scope} is *global*), {abstract} is *false*, the element information item's [namespace name] is identical to the element declaration's {target namespace} (where an ·absent· {target namespace} is taken to be identical to a [namespace name] with no value) and the element information item's [local name] matches the element declaration's {name}.

In this case the element declaration is the ·context-determined declaration· for the element information item with respect to Schema-Validity Assessment (Element) (§3.3.4) and Assessment Outcome (Element) (§3.3.5).

2.3.3 The element declaration is top-level (i.e. its {scope} is *global*), its {disallowed substitutions} does not contain *substitution*, the [local] and [namespace name] of the element information item resolve to an element declaration, as defined in QName resolution (Instance) (§3.15.4) -- [Definition:] call this declaration the **substituting declaration** and the ·substituting declaration· together with the particle's element declaration's {disallowed substitutions} is validly substitutable for the particle's element declaration as defined in Substitution Group OK

(Transitive) (§3.3.6).

In this case the ·substituting declaration· is the ·context-determined declaration· for the element information item with respect to Schema-Validity Assessment (Element) (§3.3.4) and Assessment Outcome (Element) (§3.3.5).

3 **If** the {term} is a model group, **then all** of the following must be true:

3.1 There is a ·partition· of the sequence into n sub-sequences such that n is greater than or equal to {min occurs}.

3.2 If {max occurs} is a number, n must be less than or equal to {max occurs}.

3.3 Each sub-sequence in the ·partition· is ·valid· with respect to that model group as defined in Element Sequence Valid (§3.8.4).

NOTE: Clauses clause 1 and clause 2.3.3 do not interact: an element information item validatable by a declaration with a substitution group head in a different namespace is *not* validatable by a wildcard which accepts the head's namespace but not its own.

3.9.5 Particle Information Set Contributions

None as such.

3.9.6 Constraints on Particle Schema Components

All particles (see Particles (§3.9)) must satisfy the following constraints.

Schema Component Constraint: Particle Correct

All of the following must be true:

1 The values of the properties of a particle must be as described in the property tableau in The Particle Schema Component (§3.9.1), modulo the impact of Missing Sub-components (§5.3).

2 If {max occurs} is not *unbounded*, that is, it has a numeric value, then **all** of the following must be true:

2.1 {min occurs} must not be greater than {max occurs}.

2.2 {max occurs} must be greater than or equal to 1.

The following constraints define relations appealed to elsewhere in this specification.

Schema Component Constraint: Particle Valid (Extension)

[Definition:] For a particle (call it **E**, for extension) to be a **valid extension** of another particle (call it **B**, for base) **one** of the following must be true:

1 They are the same particle.

2 **E**'s {min occurs}={max occurs}=1 and its {term} is a *sequence* group whose {particles}' first member is a particle all of whose properties, recursively, are identical to those of **B**, with the exception of {annotation} properties.

The approach to defining a type by restricting another type definition set out here is designed to ensure that types defined in this way are guaranteed to be a subset of the type they restrict. This is accomplished by requiring a clear mapping between the components of the base type definition and the restricting type definition. Permissible mappings are set out below via a set of recursive definitions, bottoming out in the obvious cases, e.g. where an (restricted) element declaration corresponds to another (base) element declaration with the same name and type but the same or wider range of occurrence.

NOTE: The structural correspondence approach to guaranteeing the subset relation set out here is necessarily verbose, but has the advantage of being checkable in a straightforward way. The working group solicits feedback on how difficult this is in practice, and on whether other approaches are found to be viable.

Schema Component Constraint: Particle Valid (Restriction)

[Definition:] For a particle (call it **R**, for restriction) to be a **valid restriction** of another particle (call it **B**, for base) **one** of the following must be true:

1 They are the same particle.

2 depending on the kind of particle, per the table below, with the qualifications that **all** of the following must be true:

2.1 Any top-level element declaration particle (in **R** or **B**) which is the {substitution group affiliation} of one or more other element declarations is treated as if it were a *choice* group whose {min occurs} and {max occurs} are those of the particle, and whose {particles} consists of one particle with {min occurs} and {max occurs} of 1 for the top-level element declaration and for each of the declarations in its ·substitution group·.

2.2 Any pointless occurrences of <sequence>, <choice> or <all> are ignored, where pointlessness is understood as follows:

<sequence>

One of the following must be true:

2.2.1 {particles} is empty.

2.2.2 **All** of the following must be true:

2.2.2.1 The particle within which this <sequence> appears has {max occurs} and {min occurs} of 1.

2.2.2.2 **One** of the following must be true:

2.2.2.2.1 The <sequence>'s {particles} has only one member.

2.2.2.2.2 The particle within which this <sequence> appears is itself among the {particles} of a <sequence>.

<all>

One of the following must be true:

2.2.1 {particles} is empty.

2.2.2 {particles} has only one member.

<choice>

One of the following must be true:

2.2.1 {particles} is empty and the particle within which this <choice> appears has {min occurs} of 0.

2.2.2 **All** of the following must be true:

2.2.2.1 The particle within which this <choice> appears has {max occurs} and {min occurs} of 1.

2.2.2.2 **One** of the following must be true:

2.2.2.2.1 The <choice>'s {particles} has only one member.

2.2.2.2.2 The particle within which this <choice> appears is itself among the {particles} of a <choice>.

		Base Particle				
		elt	**any**	**all**	**choice**	**sequence**
Derived Particle	**elt**	NameAnd-TypeOK	NSCompat	Recurse-AsIfGroup	Recurse-AsIfGroup	RecurseAs-IfGroup
	any	Forbidden	NSSubset	Forbidden	Forbidden	Forbidden
	all	Forbidden	NSRecurse-CheckCardinality	Recurse	Forbidden	Forbidden
	choice	Forbidden	NSRecurse-CheckCardinality	Forbidden	RecurseLax	Forbidden
	seq-uence	Forbidden	NSRecurse-CheckCardinality	Recurse-Unordered	MapAndSum	Recurse

Schema Component Constraint: Occurrence Range OK

For a particle's occurrence range to be a valid restriction of another's occurrence range **all** of the following must be true:

1 Its {min occurs} is greater than or equal to the other's {min occurs}.

2 **one** of the following must be true:

2.1 The other's {max occurs} is *unbounded*.

2.2 Both {max occurs} are numbers, and the particle's is less than or equal to the other's.

Schema Component Constraint: Particle Restriction OK (Elt:Elt -- NameAndTypeOK)

For an element declaration particle to be a ·valid restriction· of another element declaration particle **all** of the following must be true:

1 The declarations' {name}s and {target namespace}s are the same.

2 Either **B**'s {nillable} is *true* or **R**'s {nillable} is *false*.

3 **R**'s occurrence range is a valid restriction of **B**'s occurrence range as defined by Occurrence Range OK (§3.9.6).

4 either **B**'s declaration's {value constraint} is absent, or is not *fixed*, or **R**'s declaration's {value constraint} is *fixed* with the same value.

5 **R**'s declaration's {identity-constraint definitions} is a subset of **B**'s declaration's {identity-constraint definitions}, if any.

6 **R**'s declaration's {disallowed substitutions} is a superset of **B**'s declaration's {disallowed substitutions}.

7 **R**'s {type definition} is validly derived given {*extension*, *list*, *union*} from **B**'s {type definition} as defined by Type Derivation OK (Complex) (§3.4.6) or Type Derivation OK (Simple) (§3.14.6), as appropriate.

> **NOTE:** The above constraint on {type definition} means that in deriving a type by restriction, any contained type definitions must themselves be explicitly derived by restriction from the corresponding type definitions in the base definition.

Schema Component Constraint: Particle Derivation OK (Elt:Any -- NSCompat)

For an element declaration particle to be a ·valid restriction· of a wildcard particle **all** of the following must be true:

1 The element declaration's {target namespace} is ·valid· with respect to the wildcard's {namespace constraint} as defined by Wildcard allows Namespace Name (§3.10.4).

2 **R**'s occurrence range is a valid restriction of **B**'s occurrence range as defined by Occurrence Range OK (§3.9.6).

Schema Component Constraint: Particle Derivation OK (Elt:All/Choice/Sequence -- RecurseAsIfGroup)

For an element declaration particle to be a ·valid restriction· of a group particle (*all*, *choice* or *sequence*) a group particle of the variety corresponding to **B**'s, with {min occurs} and {max occurs} of 1 and with {particles} consisting of a single particle the same as the element declaration must be a ·valid restriction· of the group as defined by Particle Derivation OK (All:All,Sequence:Sequence -- Recurse) (§3.9.6), Particle Derivation OK (Choice:Choice -- RecurseLax) (§3.9.6) or Particle Derivation OK (All:All,Sequence:Sequence -- Recurse) (§3.9.6), depending on whether the group is *all*, *choice* or *sequence*.

Schema Component Constraint: Particle Derivation OK (Any:Any -- NSSubset)

For a wildcard particle to be a ·valid restriction· of another wildcard particle **all** of the following must be true:

1 **R**'s occurrence range must be a valid restriction of **B**'s occurrence range as defined by Occurrence Range OK (§3.9.6).

2 **R**'s {namespace constraint} must be an intensional subset of **B**'s {namespace constraint} as defined by Wildcard Subset (§3.10.6).

Schema Component Constraint: Particle Derivation OK (All/Choice/Sequence:Any -- NSRecurseCheckCardinality)

For a group particle to be a ·valid restriction· of a wildcard particle **all** of the following must be true:

1 Every member of the {particles} of the group is a ·valid restriction· of the wildcard as defined by Particle Valid (Restriction) (§3.9.6).

2 The effective total range of the group, as defined by Effective Total Range (all and sequence) (§3.8.6) (if the group is *all* or *sequence*) or Effective Total Range (choice) (§3.8.6) (if it is *choice*) is a valid restriction of **B**'s occurrence range as defined by Occurrence Range OK (§3.9.6).

Schema Component Constraint: Particle Derivation OK (All:All,Sequence:Sequence -- Recurse)

For an *all* or *sequence* group particle to be a ·valid restriction· of another group particle with the same {compositor} **all** of the following must be true:

1 **R**'s occurrence range is a valid restriction of **B**'s occurrence range as defined by Occurrence Range OK (§3.9.6).

2 There is a complete ·order-preserving· functional mapping from the particles in the {particles} of **R** to the particles in the {particles} of **B** such that **all** of the following must be true:

2.1 Each particle in the {particles} of **R** is a ·valid restriction· of the particle in the {particles} of **B** it maps to as defined by Particle Valid (Restriction) (§3.9.6).

2.2 All particles in the {particles} of **B** which are not mapped to by any particle in the {particles} of **R** are ·emptiable· as defined by Particle Emptiable (§3.9.6).

> **NOTE:** Although the ·validation· semantics of an *all* group does not depend on the order of its particles, derived *all* groups are required to match the order of their base in order to simplify checking that the derivation is OK.

[Definition:] A complete functional mapping is **order-preserving** if each particle **r** in the domain **R** maps to a particle **b** in the range **B** which follows (not necessarily immediately) the particle in the range **B** mapped to by the predecessor of **r**, if any, where "predecessor" and "follows" are defined with respect to the order of the lists which constitute **R** and **B**.

Schema Component Constraint: Particle Derivation OK (Choice:Choice -- RecurseLax)

For a *choice* group particle to be a ·valid restriction· of another *choice* group particle **all** of the following must be true:

1 **R**'s occurrence range is a valid restriction of **B**'s occurrence range as defined by Occurrence Range OK (§3.9.6);

2 There is a complete ·order-preserving· functional mapping from the particles in the {particles} of **R** to the particles in the {particles} of **B** such that each particle in the {particles} of **R** is a ·valid restriction· of the particle in the {particles} of **B** it maps to as defined by Particle Valid (Restriction) (§3.9.6).

> **NOTE:** Although the ·validation· semantics of a *choice* group does not depend on the order of its particles, derived *choice* groups are required to match the order of their base in order to simplify checking that the derivation is OK.

Schema Component Constraint: Particle Derivation OK (Sequence:All -- RecurseUnordered)

For a *sequence* group particle to be a ·valid restriction· of an *all* group particle **all** of the following must be true:

1 **R**'s occurrence range is a valid restriction of **B**'s occurrence range as defined by Occurrence Range OK (§3.9.6).

2 There is a complete functional mapping from the particles in the {particles} of **R** to the particles in the {particles} of **B** such that **all** of the following must be true:

2.1 No particle in the {particles} of **B** is mapped to by more than one of the particles in the {particles} of **R**;

2.2 Each particle in the {particles} of **R** is a ·valid restriction· of the particle in the {particles} of **B** it maps to as defined by Particle Valid (Restriction) (§3.9.6);

2.3 All particles in the {particles} of **B** which are not mapped to by any particle in the {particles} of **R** are ·emptiable· as defined by Particle Emptiable (§3.9.6).

NOTE: Although this clause allows reordering, because of the limits on the contents of *all* groups the checking process can still be deterministic.

Schema Component Constraint: Particle Derivation OK (Sequence:Choice -- MapAndSum)

For a *sequence* group particle to be a ·valid restriction· of a *choice* group particle **all** of the following must be true:

1 There is a complete functional mapping from the particles in the {particles} of **R** to the particles in the {particles} of **B** such that each particle in the {particles} of **R** is a ·valid restriction· of the particle in the {particles} of **B** it maps to as defined by Particle Valid (Restriction) (§3.9.6).

2 The pair consisting of the product of the {min occurs} of **R** and the length of its {particles} and *unbounded* if {max occurs} is *unbounded* otherwise the product of the {max occurs} of **R** and the length of its {particles} is a valid restriction of **B**'s occurrence range as defined by Occurrence Range OK (§3.9.6).

NOTE: This clause is in principle more restrictive than absolutely necessary, but in practice will cover all the likely cases, and is much easier to specify than the fully general version.

NOTE: This case allows the "unfolding" of iterated disjunctions into sequences. It may be particularly useful when the disjunction is an implicit one arising from the use of substitution groups.

Schema Component Constraint: Particle Emptiable

[Definition:] For a particle to be **emptiable** **one** of the following must be true:

1 Its {min occurs} is `0`.

2 Its {term} is a group and the minimum part of the effective total range of that group, as defined by Effective Total Range (all and sequence) (§3.8.6) (if the group is *all* or *sequence*) or Effective Total Range (choice) (§3.8.6) (if it is *choice*), is `0`.

3.10 Wildcards

In order to exploit the full potential for extensibility offered by XML plus namespaces, more provision is needed than DTDs allow for targeted flexibility in content models and attribute declarations. A wildcard provides for ·validation· of attribute and element information items dependent on their namespace name, but independently of their local name.

Example

```
<xs:any processContents="skip"/>

<xs:any namespace="##other" processContents="lax"/>

<xs:any namespace="http://www.w3.org/1999/XSL/Transform"/>

<xs:any namespace="##targetNamespace"/>

<xs:anyAttribute namespace="http://www.w3.org/XML/1998/namespace"/>
```

XML representations of the four basic types of wildcard, plus one attribute wildcard.

3.10.1 The Wildcard Schema Component

The wildcard schema component has the following properties:

Schema Component: Wildcard

{namespace constraint}
One of *any*; a pair of *not* and a namespace name or ·absent·; or a set whose members are either namespace names or ·absent·.

{process contents}
One of *skip*, *lax* or *strict*.

{annotation}
Optional. An annotation.

{namespace constraint} provides for ·validation· of attribute and element items that:

1. (*any*) have any namespace or are not namespace qualified;
2. (*not* and a namespace name) have any namespace other than the specified namespace name, or are not namespace qualified;
3. (*not* and ·absent·) are namespace qualified;
4. (a set whose members are either namespace names or ·absent·) have any of the specified namespaces and/or, if ·absent· is included in the set, are unqualified.

{process contents} controls the impact on ·assessment· of the information items allowed by wildcards, as follows:

strict
There must be a top-level declaration for the item available, or the item must have an `xsi:type`, and the item must be ·valid· as appropriate.

skip
No constraints at all: the item must simply be well-formed XML.

lax
If the item, or any items among its [children] if it's an element information item, has a uniquely determined declaration available, it must be ·valid· with respect to that definition, that is, ·validate· where you can, don't worry when you can't.

See Annotations (§3.13) for information on the role of the {annotation} property.

3.10.2 XML Representation of Wildcard Schema Components

The XML representation for a wildcard schema component is an <any> or <anyAttribute> element information item. The correspondences between the properties of an <any> information item and properties of the components it corresponds to are as follows (see <complexType> and <attributeGroup> for the correspondences for <anyAttribute>):

XML Representation Summary: any Element Information Item

```
<any
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded)  : 1
  minOccurs = nonNegativeInteger : 1
  namespace = ((##any | ##other) | List of (anyURI | (##targetNamespace
| ##local)) )  : ##any
  processContents = (lax | skip | strict) : strict
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
```

```
</any>
```

A particle containing a wildcard, with properties as follows (unless `minOccurs=maxOccurs=0`, in which case the item corresponds to no component at all):

Particle Schema Component

Property	Representation
{min occurs}	The ·actual value· of the `minOccurs` [attribute], if present, otherwise `1`.
{max occurs}	*unbounded*, if the `maxOccurs` [attribute] equals *unbounded*, otherwise the ·actual value· of the `maxOccurs` [attribute], if present, otherwise `1`.
{term}	A wildcard as given below:

Wildcard Schema Component

Property	Representation
{namespace constraint}	Dependent on the ·actual value· of the `namespace` [attribute]: if absent, then *any*, otherwise as follows: **##any** *any* **##other** a pair of *not* and the ·actual value· of the `targetNamespace` [attribute] of the <schema> ancestor element information item if present, otherwise ·absent·. ***otherwise*** a set whose members are namespace names corresponding to the space-delimited substrings of the string, except 1 if one such substring is `##targetNamespace`, the corresponding member is the ·actual value· of the `targetNamespace` [attribute] of the <schema> ancestor element information item if present, otherwise ·absent·. 2 if one such substring is `##local`, the corresponding member is ·absent·.
{process contents}	The ·actual value· of the `processContents` [attribute], if present, otherwise *strict*.
{annotation}	The annotation corresponding to the <annotation> element information item in the [children], if present, otherwise ·absent·.

Wildcards are subject to the same ambiguity constraints (Unique Particle Attribution (§3.8.6)) as other content model particles: If an instance element could match either an explicit particle and a wildcard, or one of two wildcards, within the content model of a type, that model is in error.

3.10.3 Constraints on XML Representations of Wildcards

Schema Representation Constraint: Wildcard Representation OK

In addition to the conditions imposed on <any> element information items by the schema for schemas, the corresponding particle and model group must satisfy the conditions set out in Constraints on Model Group Schema Components (§3.8.6) and Constraints on Particle Schema Components (§3.9.6).

3.10.4 Wildcard Validation Rules

Validation Rule: Item Valid (Wildcard)

For an element or attribute information item to be locally ·valid· with respect to a wildcard constraint its [namespace name] must be ·valid· with respect to the wildcard constraint, as defined in Wildcard allows Namespace Name (§3.10.4).

When this constraint applies the appropriate **case** among the following must be true:

1 **If** {process contents} is *lax*, **then** the item has no ·context-determined declaration· with respect to Assessment Outcome (Element) (§3.3.5), Schema-Validity Assessment (Element) (§3.3.4) and Schema-Validity Assessment (Attribute) (§3.2.4).

2 **If** {process contents} is *strict*, **then** the item's ·context-determined declaration· is *mustFind*.

3 **If** {process contents} is *skip*, **then** the item's ·context-determined declaration· is *skip*.

Validation Rule: Wildcard allows Namespace Name

For a value which is either a namespace name or ·absent· to be ·valid· with respect to a wildcard constraint (the value of a {namespace constraint}) **one** of the following must be true:

1 The constraint must be *any*.

2 **All** of the following must be true:

2.1 The constraint is a pair of *not* and a namespace name or ·absent· ([Definition:] call this the **namespace test**).

2.2 The value must not be identical to the ·namespace test·.

2.3 The value must not be ·absent·.

3 The constraint is a set, and the value is identical to one of the members of the set.

3.10.5 Wildcard Information Set Contributions

None as such.

3.10.6 Constraints on Wildcard Schema Components

All wildcards (see Wildcards (§3.10)) must satisfy the following constraint.

Schema Component Constraint: Wildcard Properties Correct

The values of the properties of a wildcard must be as described in the property tableau in The Wildcard Schema Component (§3.10.1), modulo the impact of Missing Sub-components (§5.3).

The following constraints define a relation appealed to elsewhere in this specification.

Schema Component Constraint: Wildcard Subset

For a namespace constraint (call it **sub**) to be an intensional subset of another namespace constraint (call it **super**) **one** of the following must be true:

1 **super** must be *any*.

2 **All** of the following must be true:

2.1 **sub** must be a pair of *not* and a namespace name or ·absent·.

2.2 **super** must be a pair of *not* and the same value.

3 **All** of the following must be true:

3.1 **sub** must be a set whose members are either namespace names or ·absent·.

3.2 **One** of the following must be true:

3.2.1 **super** must be the same set or a superset thereof.

3.2.2 **super** must be a pair of *not* and a namespace name or ·absent· and that value must not be in **sub**'s set.

Schema Component Constraint: Attribute Wildcard Union

For a wildcard's {namespace constraint} value to be the intensional union of two other such values (call them **O1** and **O2**): the appropriate **case** among the following must be true:

1 **If O1** and **O2** are the same value, **then** that value must be the value.

2 **If** either **O1** or **O2** is *any*, **then** *any* must be the value.

3 **If** both **O1** and **O2** are sets of (namespace names or ·absent·), **then** the union of those sets must be the value.
4 **If** the two are negations of different namespace names, **then** the intersection is not expressible.
5 **If** either **O1** or **O2** is a pair of *not* and a namespace name and the other is a set of (namespace names or ·absent·), **then** The appropriate **case** among the following must be true:
5.1 **If** the set includes the negated namespace name, **then** *any* must be the value.
5.2 **If** the set does not include the negated namespace name, **then** whichever of **O1** or **O2** is a pair of *not* and a namespace name must be the value.

In the case where there are more than two values, the intensional intersection is determined by identifying the intensional intersection of two of the values as above, then the intensional intersection of that value with the third (providing the first intersection was expressible), and so on as required.

Schema Component Constraint: Attribute Wildcard Intersection

For a wildcard's {namespace constraint} value to be the intensional intersection of two other such values (call them **O1** and **O2**): the appropriate **case** among the following must be true:

1 **If O1** and **O2** are the same value, **then** that value must be the value.
2 **If** either **O1** or **O2** is *any*, **then** the other must be the value.
3 **If** either **O1** or **O2** is a pair of *not* and a namespace name and the other is a set of (namespace names or ·absent·), **then** that set, minus the negated namespace name if it was in the set, must be the value.
4 **If** both **O1** and **O2** are sets of (namespace names or ·absent·), **then** the intersection of those sets must be the value.
5 **If** the two are negations of different namespace names, **then** the intersection is not expressible.

In the case where there are more than two values, the intensional intersection is determined by identifying the intensional intersection of two of the values as above, then the intensional intersection of that value with the third (providing the first intersection was expressible), and so on as required.

3.11 Identity-constraint Definitions

Identity-constraint definition components provide for uniqueness and reference constraints with respect to the contents of multiple elements and attributes.

Example

```
<xs:key name="fullName">
 <xs:selector xpath=".//person"/>
 <xs:field xpath="forename"/>
 <xs:field xpath="surname"/>
</xs:key>

<xs:keyref name="personRef" refer="fullName">
 <xs:selector xpath=".//personPointer"/>
 <xs:field xpath="@first"/>
 <xs:field xpath="@last"/>
</xs:keyref>

<xs:unique name="nearlyID">
 <xs:selector xpath=".//*"/>
 <xs:field xpath="@id"/>
</xs:unique>
```

XML representations for the three kinds of identity-constraint definitions.

3.11.1 The Identity-constraint Definition Schema Component

The identity-constraint definition schema component has the following properties:

Schema Component: Identity-constraint Definition
{name} An NCName as defined by [XML-Namespaces]. **{target namespace}** Either ·absent· or a namespace name, as defined in [XML-Namespaces]. **{identity-constraint category}** One of *key*, *keyref* or *unique*. **{selector}** A restricted XPath ([XPath]) expression. **{fields}** A non-empty list of restricted XPath ([XPath]) expressions. **{referenced key}** Required if {identity-constraint category} is *keyref*, forbidden otherwise. An identity-constraint definition with {identity-constraint category} equal to *key* or *unique*. **{annotation}** Optional. An annotation.

Identity-constraint definitions are identified by their {name} and {target namespace}; Identity-constraint definition identities must be unique within an ·XML Schema·. See References to schema components across namespaces (§4.2.3) for the use of component identifiers when importing one schema into another.

Informally, {identity-constraint category} identifies the Identity-constraint definition as playing one of three roles:

- (*unique*) the Identity-constraint definition asserts uniqueness, with respect to the content identified by {selector}, of the tuples resulting from evaluation of the {fields} XPath expression(s).
- (*key*) the Identity-constraint definition asserts uniqueness as for *unique*. *key* further asserts that all selected content actually has such tuples.
- (*keyref*) the Identity-constraint definition asserts a correspondence, with respect to the content identified by {selector}, of the tuples resulting from evaluation of the {fields} XPath expression(s), with those of the {referenced key}.

These constraints are specified along side the specification of types for the attributes and elements involved, i.e. something declared as of type integer may also serve as a key. Each constraint declaration has a name, which exists in a single symbol space for constraints. The equality and inequality conditions appealed to in checking these constraints apply to the *value* of the fields selected, so that for example `3.0` and `3` would be conflicting keys if they were both number, but non-conflicting if they were both strings, or one was a string and one a number. Values of differing type can only be equal if one type is derived from the other, and the value is in the value space of both.

Overall the augmentations to XML's `ID/IDREF` mechanism are:

- Functioning as a part of an identity-constraint is in addition to, not instead of, having a type;
- Not just attribute values, but also element content and combinations of values and content can be declared to be unique;
- Identity-constraints are specified to hold within the scope of particular elements;
- (Combinations of) attribute values and/or element content can be declared to be keys, that is, not only unique, but always present and non-nillable;
- The comparison between *keyref* {fields} and *key* or *unique* {fields} is by value equality, not by string equality.

{selector} specifies a restricted XPath ([XPath]) expression relative to instances of the element being declared. This must identify a node set of subordinate elements (i.e. contained within the declared element) to which the constraint applies.

{fields} specifies XPath expressions relative to each element selected by a {selector}. This must identify a single node (element or attribute) whose content or value, which must be of a simple type, is used in the constraint. It is possible to specify an ordered list of {fields}s, to cater to multi-field keys, keyrefs, and uniqueness constraints.

In order to reduce the burden on implementers, in particular implementers of streaming processors, only restricted subsets of XPath expressions are allowed in {selector} and {fields}. The details are given in Constraints on Identity-constraint Definition Schema Components (§3.11.6).

NOTE: Provision for multi-field keys etc. goes beyond what is supported by `xsl:key`.

See Annotations (§3.13) for information on the role of the {annotation} property.

3.11.2 XML Representation of Identity-constraint Definition Schema Components

The XML representation for an identity-constraint definition schema component is either a <key>, a <keyref> or a <unique> element information item. The correspondences between the properties of those information items and properties of the component they correspond to are as follows:

XML Representation Summary: `unique` Element Information Item

```
<unique
  id = ID
  name = NCName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (selector, field+))
</unique>
```

```
<key
  id = ID
  name = NCName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (selector, field+))
</key>
```

```
<keyref
  id = ID
  name = NCName
  refer = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (selector, field+))
</keyref>
```

```
<selector
  id = ID
  xpath = a subset of XPath expression, see below
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</selector>
```

```
<field
  id = ID
  xpath = a subset of XPath expression, see below
  {any attributes with non-schema namespace . . .}>
```

Content: (annotation?)
</field>

Identity-constraint Definition Schema Component

Property	Representation
{name}	The ·actual value· of the name [attribute]
{target namespace}	The ·actual value· of the targetNamespace [attribute] of the parent schema element information item.
{identity-constraint category}	One of *key*, *keyref* or *unique*, depending on the item.
{selector}	A restricted XPath expression corresponding to the ·actual value· of the xpath [attribute] of the <selector> element information item among the [children]
{fields}	A sequence of XPath expressions, corresponding to the ·actual value·s of the xpath [attribute]s of the <field> element information item [children], in order.
{referenced key}	If the item is a <keyref>, the identity-constraint definition ·resolved· to by the ·actual value· of the refer [attribute], otherwise ·absent·.
{annotation}	The annotation corresponding to the <annotation> element information item in the [children], if present, otherwise ·absent·.

Example

```
<xs:element name="vehicle">
 <xs:complexType>
  . . .
  <xs:attribute name="plateNumber" type="xs:integer"/>
  <xs:attribute name="state" type="twoLetterCode"/>
 </xs:complexType>
</xs:element>

<xs:element name="state">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="code" type="twoLetterCode"/>
   <xs:element ref="vehicle" maxOccurs="unbounded"/>
   <xs:element ref="person" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <xs:key name="reg"> <!-- vehicles are keyed by their plate within states
-->
  <xs:selector xpath=".//vehicle"/>
  <xs:field xpath="@plateNumber"/>
 </xs:key>
</xs:element>

<xs:element name="root">
 <xs:complexType>
  <xs:sequence>
   . . .
   <xs:element ref="state" maxOccurs="unbounded"/>
   . . .
  </xs:sequence>
```

```
  </xs:complexType>

  <xs:key name="state"> <!-- states are keyed by their code -->
   <xs:selector xpath=".//state"/>
   <xs:field xpath="code"/>
  </xs:key>

  <xs:keyref name="vehicleState" refer="state">
   <!-- every vehicle refers to its state -->
   <xs:selector xpath=".//vehicle"/>
   <xs:field xpath="@state"/>
  </xs:keyref>

  <xs:key name="regKey"> <!-- vehicles are keyed by a pair of state and
 plate -->
   <xs:selector xpath=".//vehicle"/>
   <xs:field xpath="@state"/>
   <xs:field xpath="@plateNumber"/>
  </xs:key>

  <xs:keyref name="carRef" refer="regKey"> <!-- people's cars are a
 reference -->
   <xs:selector xpath=".//car"/>
   <xs:field xpath="@regState"/>
   <xs:field xpath="@regPlate"/>
  </xs:keyref>

 </xs:element>

 <xs:element name="person">
  <xs:complexType>
   <xs:sequence>
    . . .
    <xs:element name="car">
     <xs:complexType>
      <xs:attribute name="regState" type="twoLetterCode"/>
      <xs:attribute name="regPlate" type="xs:integer"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

A `state` element is defined, which contains a `code` child and some `vehicle` and `person` children. A `vehicle` in turn has a `plateNumber` attribute, which is an integer, and a `state` attribute. State's `code`s are a key for them within the document. Vehicle's `plateNumber`s are a key for them within states, and `state` and `plateNumber` is asserted to be a *key* for `vehicle` within the document as a whole. Furthermore, a `person` element has an empty `car` child, with `regState` and `regPlate` attributes, which are then asserted together to refer to `vehicle`s via the `carRef` constraint. The requirement that a `vehicle`'s `state` match its containing `state`'s `code` is not expressed here.

3.11.3 Constraints on XML Representations of Identity-constraint Definitions

Schema Representation Constraint: Identity-constraint Definition Representation OK

In addition to the conditions imposed on <key>, <keyref> and <unique> element information items by the schema for schemas, the corresponding identity-constraint definition must satisfy the conditions set out in Constraints on Identity-constraint Definition Schema Components (§3.11.6).

3.11.4 Identity-constraint Definition Validation Rules

Validation Rule: Identity-constraint Satisfied

For an element information item to be locally ·valid· with respect to an identity-constraint **all** of the following must be true:

1 The {selector}, with the element information item as the context node, evaluates to a node-set (as defined in [XPath]). [Definition:] Call this the **target node set**.

2 Each node in the ·target node set· is an element node among the descendants of the context node.

3 For each node in the ·target node set· all of the {fields}, with that node as the context node, evaluate to either an empty node-set or a node-set with exactly one member, which must have a simple type. [Definition:] Call the sequence of the type-determined values (as defined in [XML Schemas: Datatypes]) of the [schema normalized value] of the element and/or attribute information items in those node-sets in order the **key-sequence** of the node.

4 [Definition:] Call the subset of the ·target node set· for which all the {fields} evaluate to a node-set with exactly one member which is an element or attribute node with a simple type the **qualified node set**. The appropriate **case** among the following must be true:

4.1 **If** the {identity-constraint category} is *unique*, **then** no two members of the ·qualified node set· have ·key-sequences· whose members are pairwise equal, as defined by Equal in [XML Schemas: Datatypes].

4.2 **If** the {identity-constraint category} is *key*, **then all** of the following must be true:

4.2.1 The ·target node set· and the ·qualified node set· are equal, that is, every member of the ·target node set· is also a member of the ·qualified node set· and *vice versa*.

4.2.2 No two members of the ·qualified node set· have ·key-sequences· whose members are pairwise equal, as defined by Equal in [XML Schemas: Datatypes].

4.2.3 No element member of the ·key-sequence· of any member of the ·qualified node set· was assessed as ·valid· by reference to an element declaration whose {nillable} is *true*.

4.3 **If** the {identity-constraint category} is *keyref*, **then** for each member of the ·qualified node set· (call this the **keyref member**), there must be a ·node table· associated with the {referenced key} in the [identity-constraint table] of the element information item (see Identity-constraint Table (§3.11.5), which must be understood as logically prior to this clause of this constraint, below) and there must be an entry in that table whose ·key-sequence· is equal to the **keyref member's** ·key-sequence· member for member, as defined by Equal in [XML Schemas: Datatypes].

NOTE: The use of [schema normalized value] in the definition of ·key sequence· above means that *default* or *fixed* value constraints may play a part in ·key sequence·s.

NOTE: Although this specification defines a post-schema-validation infoset contribution which would enable schema-aware processors to implement clause 4.2.3 above (Element Declaration (§3.3.5)), processors are not required to provide it. This clause can be read as if in the absence of this infoset contribution, the value of the relevant {nillable} property must be available.

3.11.5 Identity-constraint Definition Information Set Contributions

Schema Information Set Contribution: Identity-constraint Table

[Definition:] An **eligible identity-constraint** of an element information item is one such that clause 4.1 or clause 4.2 of Identity-constraint Satisfied (§3.11.4) is satisfied with respect to that item and that constraint, or such that any of the element information item [children] of that item have an [identity-constraint table] property whose value has an entry for that constraint.

[Definition:] A **node table** is a set of pairs each consisting of a ·key-sequence· and an element node.

Whenever an element information item has one or more ·eligible identity-constraints·, in the post-schema-validation infoset that element information item has a property as follows:

PSVI Contributions for element information items

[identity-constraint table]
one **Identity-constraint Binding** information item for each ·eligible identity-constraint·, with properties as follows:

PSVI Contributions for Identity-constraint Binding information items

[definition]
The ·eligible identity-constraint·.

[node table]
A ·node table· with one entry for every ·key-sequence· (call it **k**) and node (call it **n**) such that **one** of the following must be true:

1 There is an entry in one of the ·node tables· associated with the [definition] in an **Identity-constraint Binding** information item in at least one of the [identity-constraint table]s of the element information item [children] of the element information item whose ·key-sequence· is **k** and whose node is **n**;

2 **n** appears with ·key-sequence· **k** in the ·qualified node set· for the [definition].

provided no two entries have the same ·key-sequence·but distinct nodes. Potential conflicts are resolved by not including any conflicting entries which would have owed their inclusion to clause 1above. Note that if all the conflicting entries arose under clause 1above, this means no entry at all will appear for the offending ·key-sequence·.

NOTE: The complexity of the above arises from the fact that *keyref* identity-constraints may be defined on domains distinct from the embedded domain of the identity-constraint they reference, or the domains may be the same but self-embedding at some depth. In either case the ·node table· for the referenced identity-constraint needs to propagate upwards, with conflict resolution.

The **Identity-constraint Binding** information item, unlike others in this specification, is essentially an internal bookkeeping mechanism. It is introduced to support the definition of Identity-constraint Satisfied (§3.11.4) above. Accordingly, conformant processors may, but are *not* required to, expose them via [identity-constraint table] properties in the post-schema-validation infoset. In other words, the above constraints may be read as saying ·validation· of identity-constraints proceeds *as if* such infoset items existed.

3.11.6 Constraints on Identity-constraint Definition Schema Components

All identity-constraint definitions (see Identity-constraint Definitions (§3.11)) must satisfy the following constraint.

Schema Component Constraint: Identity-constraint Definition Properties Correct

All of the following must be true:

1 The values of the properties of an identity-constraint definition must be as described in the property tableau in The Identity-constraint Definition Schema Component (§3.11.1), modulo the impact of Missing Sub-components (§5.3).

2 If the {identity-constraint category} is *keyref*, the cardinality of the {fields} must equal that of the {fields} of the {referenced key}.

Schema Component Constraint: Selector Value OK

All of the following must be true:

1 The {selector} must be a valid XPath expression, as defined in [XPath].

2 **One** of the following must be true:

2.1 It must conform to the following extended BNF:

Selector XPath expressions

```
[1]    Selector    ::=    Path ( '|' Path )*
[2]    Path        ::=    ('.//')? Step ( '/' Step )*
[3]    Step        ::=    '.' | NameTest
[4]    NameTest    ::=    QName | '*' | NCName ':' '*'
```

2.2 It must be an XPath expression involving the `child` axis whose abbreviated form is as given above.

Schema Component Constraint: Fields Value OK

All of the following must be true:

1 Each member of the {fields} must be a valid XPath expression, as defined in [XPath].

2 **One** of the following must be true:

2.1 It must conform to the extended BNF given above for Selector, with the following modification:

Path in Field XPath expressions

```
[5]    Path    ::=    ('.//')? ( Step '/' )* ( Step | '@' NameTest )
```

This production differs from the one above in allowing the final step to match an attribute node.

2.2 It must be an XPath expression involving the `child` and/or `attribute` axes whose abbreviated form is as given above.

3.12 Notation Declarations

Notation declarations reconstruct XML 1.0 NOTATION declarations.

Example

```
<xs:notation name="jpeg" public="image/jpeg" system="viewer.exe">
```

The XML representation of a notation declaration.

3.12.1 The Notation Declaration Schema Component

The notation declaration schema component has the following properties:

Schema Component: Notation Declaration

{name}
An NCName as defined by [XML-Namespaces].

{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].

{system identifier}
Optional if {public identifier} is present. A URI reference.

{public identifier}
Optional if {system identifier} is present. A public identifier, as defined in [XML 1.0 (Second Edition)].

{annotation}
Optional. An annotation.

Notation declarations do not participate in ·validation· as such. They are referenced in the course of ·validating· strings as members of the NOTATION simple type.

See Annotations (§3.13) for information on the role of the {annotation} property.

3.12.2 XML Representation of Notation Declaration Schema Components

The XML representation for a notation declaration schema component is a <notation> element information item. The correspondences between the properties of that information item and properties of the component it corresponds to are as follows:

XML Representation Summary: `notation` Element Information Item

```
<notation
  id = ID
  name = NCName
  public = anyURI
  system = anyURI
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</notation>
```

Notation Declaration Schema Component

Property	Representation
{name}	The ·actual value· of the `name` [attribute]
{target namespace}	The ·actual value· of the `targetNamespace` [attribute] of the parent `schema` element information item.
{system identifier}	The ·actual value· of the `system` [attribute], if present, otherwise ·absent·.
{public identifier}	The ·actual value· of the `public` [attribute]
{annotation}	The annotation corresponding to the <annotation> element information item in the [children], if present, otherwise ·absent·.

Example

```
<xs:notation name="jpeg"
             public="image/jpeg" system="viewer.exe" />

<xs:element name="picture">
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension base="xs:hexBinary">
    <xs:attribute name="pictype">
     <xs:simpleType>
      <xs:restriction base="xs:NOTATION">
       <xs:enumeration value="jpeg"/>
       <xs:enumeration value="png"/>
       . . .
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
```

```
  </xs:element>

  <picture pictype="jpeg">...</picture>
```

3.12.3 Constraints on XML Representations of Notation Declarations

Schema Representation Constraint: Notation Definition Representation OK
In addition to the conditions imposed on <notation> element information items by the schema for schemas, the corresponding notation definition must satisfy the conditions set out in Constraints on Notation Declaration Schema Components (§3.12.6).

3.12.4 Notation Declaration Validation Rules

None as such.

3.12.5 Notation Declaration Information Set Contributions

Schema Information Set Contribution: Validated with Notation
Whenever an attribute information item is ·valid· with respect to a NOTATION, in the post-schema-validation infoset its parent element information item either has a property as follows:

PSVI Contributions for element information items
`[notation]` An ·item isomorphic· to the notation declaration whose {name} and {target namespace} match the ·local name· and ·namespace name· (as defined in QName Interpretation (§3.15.3)) of the attribute item's ·actual value·

or has a pair of properties as follows:

PSVI Contributions for element information items
`[notation system]` The value of the {system identifier} of that notation declaration. **`[notation public]`** The value of the {public identifier} of that notation declaration.

NOTE: For compatibility, only one such attribute should appear on any given element. If more than one such attribute *does* appear, which one supplies the infoset property or properties above is not defined.

3.12.6 Constraints on Notation Declaration Schema Components

All notation declarations (see Notation Declarations (§3.12)) must satisfy the following constraint.

Schema Component Constraint: Notation Declaration Correct
The values of the properties of a notation declaration must be as described in the property tableau in The Notation Declaration Schema Component (§3.12.1), modulo the impact of Missing Sub-components (§5.3).

3.13 Annotations

Annotations provide for human- and machine-targeted annotations of schema components.

Example

```
<xs:simpleType fn:note="special">
  <xs:annotation>
   <xs:documentation>A type for experts only</xs:documentation>
   <xs:appinfo>
    <fn:specialHandling>checkForPrimes</fn:specialHandling>
   </xs:appinfo>
  </xs:annotation>
```

XML representations of three kinds of annotation.

3.13.1 The Annotation Schema Component

The annotation schema component has the following properties:

Schema Component: Annotation

{application information}
A sequence of element information items.
{user information}
A sequence of element information items.
{attributes}
A sequence of attribute information items.

{user information} is intended for human consumption, {application information} for automatic processing. In both cases, provision is made for an optional URI reference to supplement the local information, as the value of the `source` attribute of the respective element information items. ·Validation· does *not* involve dereferencing these URIs, when present. In the case of {user information}, indication should be given as to the identity of the (human) language used in the contents, using the `xml:lang` attribute.

{attributes} ensures that when schema authors take advantage of the provision for adding attributes from namespaces other than the XML Schema namespace to schema documents, they are available within the components corresponding to the element items where such attributes appear.

Annotations do not participate in ·validation· as such. Provided an annotation itself satisfies all relevant ·Schema Component Constraints· it *cannot* affect the ·validation· of element information items.

3.13.2 XML Representation of Annotation Schema Components

Annotation of schemas and schema components, with material for human or computer consumption, is provided for by allowing application information and human information at the beginning of most major schema elements, and anywhere at the top level of schemas. The XML representation for an annotation schema component is an <annotation> element information item. The correspondences between the properties of that information item and properties of the component it corresponds to are as follows:

XML Representation Summary: `annotation` Element Information Item

```
<annotation
  id = ID
  {any attributes with non-schema namespace . . .}>
```

```
  Content: (appinfo | documentation)*
</annotation>
```

```
<appinfo
  source = anyURI>
  Content: ({any})*
</appinfo>
```

```
<documentation
  source = anyURI
  xml:lang = language>
  Content: ({any})*
</documentation>
```

Annotation Schema Component

Property	Representation
{application information}	A sequence of the <appinfo> element information items from among the [children], in order, if any, otherwise the empty sequence.
{user information}	A sequence of the <documentation> element information items from among the [children], in order, if any, otherwise the empty sequence.
{attributes}	A sequence of attribute information items, namely those allowed by the attribute wildcard in the type definition for the <annotation> item itself or for the enclosing items which correspond to the component within which the annotation component is located.

The annotation component corresponding to the <annotation> element in the example above will have one element item in each of its {user information} and {application information} and one attribute item in its {attributes}.

3.13.3 Constraints on XML Representations of Annotations

Schema Representation Constraint: Annotation Definition Representation OK
In addition to the conditions imposed on <annotation> element information items by the schema for schemas, the corresponding annotation must satisfy the conditions set out in Constraints on Annotation Schema Components (§3.13.6).

3.13.4 Annotation Validation Rules

None as such.

3.13.5 Annotation Information Set Contributions

None as such: the addition of annotations to the post-schema-validation infoset is covered by the post-schema-validation infoset contributions of the enclosing components.

3.13.6 Constraints on Annotation Schema Components

All annotations (see Annotations (§3.13)) must satisfy the following constraint.

Schema Component Constraint: Annotation Correct
The values of the properties of an annotation must be as described in the property tableau in The Annotation Schema Component (§3.13.1), modulo the impact of Missing Sub-components (§5.3).

3.14 Simple Type Definitions

> **NOTE:** This section consists of a combination of non-normative versions of normative material from [XML Schemas: Datatypes], for local cross-reference purposes, and normative material relating to the interface between schema components defined in this specification and the simple type definition component.

Simple type definitions provide for constraining character information item [children] of element and attribute information items.

Example

```
<xs:simpleType name="farenheitWaterTemp">
 <xs:restriction base="xs:number">
  <xs:fractionDigits value="2"/>
  <xs:minExclusive value="0.00"/>
  <xs:maxExclusive value="100.00"/>
 </xs:restriction>
</xs:simpleType>
```

The XML representation of a simple type definition.

3.14.1 (non-normative) The Simple Type Definition Schema Component

The simple type definition schema component has the following properties:

Schema Component: Simple Type Definition

{name}
Optional. An NCName as defined by [XML-Namespaces].

{target namespace}
Either ·absent· or a namespace name, as defined in [XML-Namespaces].

{base type definition}
A simple type definition, which may be the ·simple ur-type definition·.

{facets}
A set of constraining facets.

{fundamental facets}
A set of fundamental facets.

{final}
A subset of {*extension*, *list*, *restriction*, *union*}.

{variety}
One of {*atomic*, *list*, *union*}. Depending on the value of {variety}, further properties are defined as follows:

atomic

{primitive type definition}
A built-in primitive simple type definition (or the ·simple ur-type definition·).

list

{item type definition}
A simple type definition.

union

{member type definitions}
A non-empty sequence of simple type definitions.

{annotation}
Optional. An annotation.

Simple types are identified by their {name} and {target namespace}. Except for anonymous simple types (those with no {name}), since type definitions (i.e. both simple and complex type definitions taken together) must be uniquely identified within an ·XML Schema·, no simple type definition can have the same name as another simple or complex type definition. Simple type {name}s and {target namespace}s are provided for reference from instances (see xsi:type (§2.6.1)), and for use in the XML representation of schema components (specifically in <element> and <attribute>). See References to schema components across namespaces (§4.2.3) for the use of component identifiers when importing one schema into another.

> **NOTE:** The {name} of a simple type is not *ipso facto* the [(local) name] of the element or attribute information items ·validated· by that definition. The connection between a name and a type definition is described in Element Declarations (§3.3) and Attribute Declarations (§3.2).

A simple type definition with an empty specification for {final} can be used as the {base type definition} for other types derived by either of extension or restriction, or as the {item type definition} in the definition of a list, or in the {member type definitions} of a union; the explicit values *extension*, *restriction*, *list* and *union* prevent further derivations by extension (to yield a complex type) and restriction (to yield a simple type) and use in constructing lists and unions respectively.

{variety} determines whether the simple type corresponds to an *atomic*, *list* or *union* type as defined by [XML Schemas: Datatypes].

As described in Type Definition Hierarchy (§2.2.1.1), every simple type definition is a ·restriction· of some other simple type (the {base type definition}), which is the simple ·ur-type definition· if and only if the type definition in question is one of the built-in primitive datatypes, or a list or union type definition. Each *atomic* type is ultimately a restriction of exactly one such built-in simple {primitive type definition}.

{facets} for each simple type definition are selected from those defined in [XML Schemas: Datatypes]. For *atomic* definitions, these are restricted to those appropriate for the corresponding {primitive type definition}. Therefore, the value space and lexical space (i.e. what is ·validated· by any atomic simple type) is determined by the pair ({primitive type definition}, {facets}).

As specified in [XML Schemas: Datatypes], *list* simple type definitions ·validate· space separated tokens, each of which conforms to a specified simple type definition, the {item type definition}. The item type specified must not itself be a *list* type, and must be one of the types identified in [XML Schemas: Datatypes] as a suitable item type for a list simple type. In this case the {facets} apply to the list itself, and are restricted to those appropriate for lists.

A *union* simple type definition ·validates· strings which satisfy at least one of its {member type definitions}. As in the case of *list*, the {facets} apply to the union itself, and are restricted to those appropriate for unions.

As discussed in Type Definition Hierarchy (§2.2.1.1), the ·ur-type definition· functions as a simple type when used as the ·base type definition· for the built-in primitive datatypes and for list and union type definitions. It is considered to have an unconstrained lexical space, and a value space consisting of the union of the value spaces of all the built-in primitive datatypes and the set of all lists of all members of the value spaces of all the built-in primitive datatypes.

The simple ·ur-type definition· must *not* be named as the ·base type definition· of any user-defined simple

types: as it has no constraining facets, this would be incoherent.

See Annotations (§3.13) for information on the role of the {annotation} property.

3.14.2 (non-normative) XML Representation of Simple Type Definition Schema Components

NOTE: This section reproduces a version of material from [XML Schemas: Datatypes], for local cross-reference purposes.

XML Representation Summary: `simpleType` Element Information Item

```
<simpleType
  final = (#all | (list | union | restriction))
  id = ID
  name = NCName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (restriction | list | union))
</simpleType>
```

```
<restriction
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleType?, (minExclusive | minInclusive |
maxExclusive | maxInclusive | totalDigits | fractionDigits | length | minLength |
maxLength | enumeration | whiteSpace | pattern)*))
</restriction>
```

```
<list
  id = ID
  itemType = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleType?))
</list>
```

```
<union
  id = ID
  memberTypes = List of QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleType*))
</union>
```

Simple Type Definition Schema Component

Property	Representation
{name}	The ·actual value· of the `name` [attribute] if present, otherwise ·absent·.
{target namespace}	The ·actual value· of the `targetNamespace` [attribute] of the <schema> ancestor element information item if present, otherwise ·absent·.
{base type definition}	The appropriate **case** among the following: 1 **If** the <restriction> alternative is chosen, **then** the type definition ·resolved· to by the ·actual value· of the `base` [attribute] of <restriction>, if present, otherwise the type definition corresponding to the <simpleType> among the [children] of <restriction>. 2 **If** the <list> or <union> alternative is chosen, **then** the ·simple ur-type definition·.

{final}	As for the {prohibited substitutions} property of complex type definitions, but using the `final` and `finalDefault` [attributes] in place of the `block` and `blockDefault` [attributes] and with the relevant set being {*extension*, *restriction*, *list*, *union*}.
{variety}	If the <list> alternative is chosen, then *list*, otherwise if the <union> alternative is chosen, then *union*, otherwise (the <restriction> alternative is chosen), then the {variety} of the {base type definition}.

If the {variety} is *atomic*, the following additional property mappings also apply:

Atomic Simple Type Definition Schema Component

Property	Representation
{primitive type definition}	The built-in primitive type definition from which the {base type definition} is derived.
{facets}	A set of facet components ·constituting a restriction· of the {facets} of the {base type definition} with respect to a set of facet components corresponding to the appropriate element information items among the [children] of <restriction> (i.e. those which specify facets, if any), as defined in Simple Type Restriction (Facets) (§3.14.3).

If the {variety} is *list*, the following additional property mappings also apply:

List Simple Type Definition Schema Component

Property	Representation
{item type definition}	The appropriate **case** among the following: 1 **If** the <list> alternative is chosen, **then** the type definition ·resolved· to by the ·actual value· of the `itemType` [attribute] of <list>, if present, otherwise the type definition corresponding to the <simpleType> among the [children] of <list>. 2 **If** the <restriction> option is chosen, **then** the {item type definition} of the {base type definition}.
{facets}	If the <restriction> alternative is chosen, a set of facet components ·constituting a restriction· of the {facets} of the {base type definition} with respect to a set of facet components corresponding to the appropriate element information items among the [children] of <restriction> (i.e. those which specify facets, if any), as defined in Simple Type Restriction (Facets) (§3.14.3), otherwise the empty set.

If the {variety} is *union*, the following additional property mappings also apply:

Union Simple Type Definition Schema Component

Property	Representation
{member type definitions}	The appropriate **case** among the following: 1 **If** the <union> alternative is chosen, **then** [Definition:] define the **explicit members** as the type definitions ·resolved· to by the items in the ·actual value· of the `memberTypes` [attribute], if any, followed by the type definitions corresponding to the <simpleType>s among the [children] of <union>, if any. The actual value is then formed by replacing any union type definition in the ·explicit members· with the members of their {member type definitions}, in order. 2 **If** the <restriction> option is chosen, **then** the {member type definitions} of the {base type definition}.
{facets}	If the <restriction> alternative is chosen, a set of facet components ·constituting a restriction· of the {facets} of the {base type definition} with respect to a set of facet components corresponding to the appropriate element information items among the [children] of <restriction> (i.e. those which specify facets, if any), as defined in Simple Type Restriction (Facets) (§3.14.3), otherwise the empty set.

3.14.3 (non-normative) Constraints on XML Representations of Simple Type Definitions

Schema Representation Constraint: Simple Type Definition Representation OK

In addition to the conditions imposed on <simpleType> element information items by the schema for schemas, **all** of the following must be true:

1 The corresponding simple type definition, if any, must satisfy the conditions set out in Constraints on Simple Type Definition Schema Components (§3.14.6).

2 If the <restriction> alternative is chosen, either it must have a `base` [attribute] or a <simpleType> among its [children], but not both.

3 If the <list> alternative is chosen, either it must have an `itemType` [attribute] or a <simpleType> among its [children], but not both.

4 Circular union type definition is disallowed. That is, if the <union> alternative is chosen, there must not be any entries in the `memberTypes` [attribute] at any depth which resolve to the component corresponding to the <simpleType>.

Schema Representation Constraint: Simple Type Restriction (Facets)

For a simple type definition (call it **R**) to restrict another simple type definition (call it **B**) with a set of facets (call this **S**) **all** of the following must be true:

1 The {variety} and {primitive type definition} of **R** are the same as those of **B**.

2 The {facets} of **R** are the union of **S** and the {facets} of **B**, eliminating duplicates. To eliminate duplicates, when a facet of the same kind occurs in both **S** and the {facets} of **B**, the one in the {facets} of **B** is not included, with the exception of enumeration and pattern facets, for which multiple occurrences with distinct values are allowed.

[Definition:] If clause 2 above holds, the {facets} of **R constitute a restriction** of the {facets} of **B** with respect to **S**.

3.14.4 Simple Type Definition Validation Rules

Validation Rule: String Valid

A string is locally ·valid· with respect to a simple type definition if it is schema-valid with respect to that definition as defined by Datatype Valid in [XML Schemas: Datatypes].

3.14.5 Simple Type Definition Information Set Contributions

None as such.

3.14.6 Constraints on Simple Type Definition Schema Components

All simple type definitions (see Simple Type Definitions (§3.14)) must satisfy the following constraints.

Schema Component Constraint: Simple Type Definition Properties Correct

All of the following must be true:

1 The values of the properties of a simple type definition must be as described in the property tableau in Datatype definition, modulo the impact of Missing Sub-components (§5.3).

2 Circular definitions are disallowed. That is, it must be possible to reach a built-in primitive datatype or the ·simple ur-type definition· by repeatedly following the {base type definition}.

3 The {final} of the {base type definition} must not contain *restriction*.

4 If the {base type definition} is not the ·simple ur-type definition·, **all** of the following must be true:

4.1 The definition must be a ·valid restriction· as defined in Derivation Valid (Restriction, Simple) (§3.14.6).

4.2 If {variety} is not *atomic*, then the appropriate **case** among the following must be true:

4.2.1 **If** the {variety} is *list*, **then** the {final} of the {base type definition} must not contain *list*.

4.2.2 **If** the {variety} is *union*, **then** the {final} of the {base type definition} must not contain *union*.

Schema Component Constraint: Derivation Valid (Restriction, Simple)

The appropriate **case** among the following must be true:

1 **If** the {variety} is *atomic*, **then all** of the following must be true:

1.1 The {base type definition} must be an atomic simple type definition or a built-in primitive datatype.

1.2 The {final} of the {base type definition} must not contain *restriction*.

1.3 For each facet in the {facets} there must be a facet of the same kind in the {facets} of the {base type definition} of whose {value} the facet in question's {value} must be a valid restriction as defined in [XML Schemas: Datatypes].

2 **If** the {variety} is *list*, **then all** of the following must be true:

2.1 The {item type definition} must have a {variety} of *atomic* or *union* (in which case all the {member type definitions} must be *atomic*).

2.2 Only *length*, *minLength*, *maxLength*, *pattern* and *enumeration* facet components are allowed among the {facets}.

2.3 If the {base type definition} is not the ·simple ur-type definition·, then **all** of the following must be true:

2.3.1 The {base type definition} must have a {variety} of *list*.

2.3.2 The {final} of the {base type definition} must not contain *restriction*.

2.3.3 for each facet in the {facets} there must be a facet of the same kind in the {facets} of the {base type definition} of whose {value} the facet in question's {value} must be a valid restriction as defined in [XML Schemas: Datatypes].

3 **If** the {variety} is *union*, **then all** of the following must be true:

3.1 The {member type definitions} must all have {variety} of *atomic* or *list*.

3.2 Only *pattern* and *enumeration* facet components are allowed among the {facets}.

3.3 If the {base type definition} is not the ·simple ur-type definition·, then **all** of the following must be true:

3.3.1 The {base type definition} must have a {variety} of *union*.

3.3.2 The {final} of the {base type definition} must not contain *restriction*.

3.3.3 for each facet in the {facets} there must be a facet of the same kind in the {facets} of the {base type definition} of whose {value} the facet in question's {value} must be a valid restriction as defined in [XML Schemas: Datatypes].

[Definition:] If this constraint Derivation Valid (Restriction, Simple) (§3.14.6) holds of a simple type

definition, it is a **valid restriction** of its ·base type definition·.

The following constraint defines relations appealed to elsewhere in this specification.

Schema Component Constraint: Type Derivation OK (Simple)

For a simple type definition (call it **D**, for derived) to be validly derived from a simple type definition (call this **B**, for base) given a subset of {*extension*, *restriction*, *list*, *union*} (of which only *restriction* is actually relevant) **one** of the following must be true:

1 They are the same type definition.

2 **All** of the following must be true:

2.1 *restriction* is not in the subset, or in the {final} of its own {base type definition};

2.2 **One** of the following must be true:

2.2.1 **D**'s ·base type definition· is **B**.

2.2.2 **D**'s ·base type definition· is not the ·simple ur-type definition· and is validly derived from **B** given the subset, as defined by this constraint.

2.2.3 **D**'s {variety} is *list* or *union* and **B** is the ·simple ur-type definition·.

2.2.4 **B**'s {variety} is *union* and **D** is validly derived from a type definition in **B**'s {member type definitions} given the subset, as defined by this constraint.

3.14.7 Built-in Simple Type Definition

There is a simple type definition nearly equivalent to the simple version of the ·ur-type definition· present in every schema by definition. It has the following properties:

Simple Type Definition of the Ur-Type

Property	Value
{name}	anySimpleType
{target namespace}	http://www.w3.org/2001/XMLSchema
{base type definition}	·the ur-type definition·
{final}	The empty set
{variety}	·absent·

Simple type definitions for all the built-in primitive datatypes, namely *string*, *boolean*, *float*, *double*, *number*, *dateTime*, *duration*, *time*, *date*, *gMonth*, *gMonthDay*, *gDay*, *gYear*, *gYearMonth*, *hexBinary*, *base64Binary*, *anyURI* (see the Primitive Datatypes section of [XML Schemas: Datatypes]), as well as for the simple and complex ·ur-type definitions· (as previously described), are present by definition in every schema. All are in the XML Schema {target namespace} (namespace name `http://www.w3.org/2001/XMLSchema`), have an *atomic* {variety} with an empty {facets} and the simple ·ur-type definition· as their ·base type definition· and themselves as {primitive type definition}.

Similarly, simple type definitions for all the built-in derived datatypes (see the Derived Datatypes section of [XML Schemas: Datatypes]) are present by definition in every schema, with properties as specified in [XML Schemas: Datatypes] and as represented in XML in Schema for Schemas (normative) (§A).

3.15 Schemas as a Whole

A schema consists of a set of schema components.

Example

```
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.com/example">
  . . .
</xs:schema>
```

The XML representation of the skeleton of a schema.

3.15.1 The Schema Itself

At the abstract level, the schema itself is just a container for its components.

Schema Component: Schema

{type definitions}
A set of named simple and complex type definitions.
{attribute declarations}
A set of named (top-level) attribute declarations.
{element declarations}
A set of named (top-level) element declarations.
{attribute group definitions}
A set of named attribute group definitions.
{model group definitions}
A set of named model group definitions.
{notation declarations}
A set of notation declarations.
{annotations}
A set of annotations.

3.15.2 XML Representations of Schemas

A schema is represented in XML by one or more ·schema documents·, that is, one or more <schema> element information items. A ·schema document· contains representations for a collection of schema components, e.g. type definitions and element declarations, which have a common {target namespace}. A ·schema document· which has one or more <import> element information items corresponds to a schema with components with more than one {target namespace}, see Import Constraints and Semantics (§4.2.3).

XML Representation Summary: schema Element Information Item

```
<schema
  attributeFormDefault = (qualified | unqualified) : unqualified
  blockDefault = (#all | List of (extension | restriction |
substitution))  : ''
  elementFormDefault = (qualified | unqualified) : unqualified
  finalDefault = (#all | List of (extension | restriction))  : ''
  id = ID
  targetNamespace = anyURI
  version = token
  xml:lang = language
  {any attributes with non-schema namespace . . .}>
  Content: ((include | import | redefine | annotation)*, (((simpleType |
complexType | group | attributeGroup) | element | attribute | notation), annotation*)
*)
```

```
</schema>
```

Schema Schema Component

Property	Representation
{type definitions}	The simple and complex type definitions corresponding to all the <simpleType> and <complexType> element information items in the [children], if any, plus any included or imported definitions, see Assembling a schema for a single target namespace from multiple schema definition documents (§4.2.1) and References to schema components across namespaces (§4.2.3).
{attribute declarations}	The (top-level) attribute declarations corresponding to all the <attribute> element information items in the [children], if any, plus any included or imported declarations, see Assembling a schema for a single target namespace from multiple schema definition documents (§4.2.1) and References to schema components across namespaces (§4.2.3).
{element declarations}	The (top-level) element declarations corresponding to all the <element> element information items in the [children], if any, plus any included or imported declarations, see Assembling a schema for a single target namespace from multiple schema definition documents (§4.2.1) and References to schema components across namespaces (§4.2.3).
{attribute group definitions}	The attribute group definitions corresponding to all the <attributeGroup> element information items in the [children], if any, plus any included or imported definitions, see Assembling a schema for a single target namespace from multiple schema definition documents (§4.2.1) and References to schema components across namespaces (§4.2.3).
{model group definitions}	The model group definitions corresponding to all the <group> element information items in the [children], if any, plus any included or imported definitions, see Assembling a schema for a single target namespace from multiple schema definition documents (§4.2.1) and References to schema components across namespaces (§4.2.3).
{notation declarations}	The notation declarations corresponding to all the <notation> element information items in the [children], if any, plus any included or imported declarations, see Assembling a schema for a single target namespace from multiple schema definition documents (§4.2.1) and References to schema components across namespaces (§4.2.3).
{annotations}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

Note that none of the attribute information items displayed above correspond directly to properties of schemas. The `blockDefault`, `finalDefault`, `attributeFormDefault`, `elementFormDefault`and `targetNamespace` attributes are appealed to in the sub-sections above, as they provide global information applicable to many representation/component correspondences. The other attributes (`id` and `version`) are for user convenience, and this specification defines no semantics for them.

The definition of the schema abstract data model in XML Schema Abstract Data Model (§2.2) makes clear that most components have a {target namespace}. Most components corresponding to representations within a given <schema> element information item will have a {target namespace} which corresponds to the `targetNamespace` attribute.

Since the empty string is not a legal namespace name, supplying an empty string for `targetNamespace` is incoherent, and is *not* the same as not specifying it at all. The appropriate form of schema document corresponding to a ·schema· whose components have no {target namespace} is one which has no `targetNamespace` attribute specified at all.

> **NOTE:** The XML namespaces Recommendation discusses only instance document syntax for elements and attributes; it therefore provides no direct framework for managing the names of type definitions, attribute group definitions, and so on. Nevertheless, the specification applies the target namespace facility uniformly to all schema components, i.e. not only declarations but also definitions have a {target namespace}.

Although the example schema at the beginning of this section might be a complete XML document, <schema> need not be the document element, but can appear within other documents. Indeed there is no requirement that a schema correspond to a (text) document at all: it could correspond to an element information item constructed 'by hand', for instance via a DOM-conformant API.

Aside from <include> and <import>, which do not correspond directly to any schema component at all, each of the element information items which may appear in the content of <schema> corresponds to a schema component, and all except <annotation> are named. The sections below present each such item in turn, setting out the components to which it may correspond.

3.15.2.1 References to Schema Components

Reference to schema components from a schema document is managed in a uniform way, whether the component corresponds to an element information item from the same schema document or is imported (References to schema components across namespaces (§4.2.3)) from an external schema (which may, but need not, correspond to an actual schema document). The form of all such references is a ·QName·.

[Definition:] A **QName** is a name with an optional namespace qualification, as defined in [XML-Namespaces]. When used in connection with the XML representation of schema components or references to them, this refers to the simple type QName as defined in [XML Schemas: Datatypes].

[Definition:] An **NCName** is a name with no colon, as defined in [XML-Namespaces]. When used in connection with the XML representation of schema components in this specification, this refers to the simple type NCName as defined in [XML Schemas: Datatypes].

In each of the XML representation expositions in the following sections, an attribute is shown as having type `QName` if and only if it is interpreted as referencing a schema component.

Example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xhtml="http://www.w3.org/1999/xhtml"
           xmlns="http://www.example.com"
           targetNamespace="http://www.example.com">
  . . .

  <xs:element name="elem1" type="Address"/>

  <xs:element name="elem2" type="xhtml:blockquote"/>

  <xs:attribute name="attr1"
```

```
                type="xsl:quantity"/>
  . . .
</xs:schema>
```

The first of these is most probably a local reference, i.e. a reference to a type definition corresponding to a <complexType> element information item located elsewhere in the schema document, the other two refer to type definitions from schemas for other namespaces and assume that their namespaces have been declared for import. See References to schema components across namespaces (§4.2.3) for a discussion of importing.

3.15.2.2 References to Schema Components from Elsewhere

The names of schema components such as type definitions and element declarations are not of type ID: they are not unique within a schema, just within a symbol space. This means that simple fragment identifiers will not always work to reference schema components from outside the context of schema documents.

There is currently no provision in the definition of the interpretation of fragment identifiers for the `text/xml` MIME type, which is the MIME type for schemas, for referencing schema components as such. However, [XPointer] provides a mechanism which maps well onto the notion of symbol spaces as it is reflected in the XML representation of schema components. A fragment identifier of the form `#xpointer(xs:schema/xs:element[@name="person"])` will uniquely identify the representation of a top-level element declaration with name `person`, and similar fragment identifiers can obviously be constructed for the other global symbol spaces.

Short-form fragment identifiers may also be used in some cases, that is when a DTD or XML Schema is available for the schema in question, and the provision of an `id` attribute for the representations of all primary and secondary schema components, which *is* of type ID, has been exploited.

It is a matter for applications to specify whether they interpret document-level references of either of the above varieties as being to the relevant element information item (i.e. without special recognition of the relation of schema documents to schema components) or as being to the corresponding schema component.

3.15.3 Constraints on XML Representations of Schemas

Schema Representation Constraint: QName Interpretation

Where the type of an attribute information item in a document involved in ·validation· is identified as ·QName·, its ·actual value· is composed of a [Definition:] **local name** and a [Definition:] **namespace name**. Its ·actual value· is determined based on its ·normalized value· and the containing element information item's [in-scope namespaces] following [XML-Namespaces]:

The appropriate **case** among the following must be true:

1 **If** its ·normalized value· is prefixed, **then all** of the following must be true:
  1.1 There must be a namespace in the [in-scope namespaces] whose [prefix] matches the prefix.
  1.2 its ·namespace name· is the [namespace name] of that namespace.
  1.3 Its ·local name· is the portion of its ·normalized value· after the colon (`':'`).
2 **otherwise** (its ·normalized value· is unprefixed) **all** of the following must be true:
  2.1 its ·local name· is its ·normalized value·.
  2.2 The appropriate **case** among the following must be true:
    2.2.1 **If** there is a namespace in the [in-scope namespaces] whose [prefix] has no value, **then** its ·namespace name· is the [namespace name] of that namespace.
    2.2.2 **otherwise** its ·namespace name· is ·absent·.

In the absence of the [in-scope namespaces] property in the infoset for the schema document in

question, processors must reconstruct equivalent information as necessary, using the [namespace attributes] of the containing element information item and its ancestors.

[Definition:] Whenever the word **resolve** in any form is used in this chapter in connection with a ·QName· in a schema document, the following definition QName resolution (Schema Document) (§3.15.3) should be understood:

Schema Representation Constraint: QName resolution (Schema Document)

For a ·QName· to resolve to a schema component of a specified kind **all** of the following must be true:

1 That component is a member of the value of the appropriate property of the schema which corresponds to the schema document within which the ·QName· appears, that is the appropriate **case** among the following must be true:

1.1 **If** the kind specified is simple or complex type definition, **then** the property is the {type definitions}.

1.2 **If** the kind specified is attribute declaration, **then** the property is the {attribute declarations}.

1.3 **If** the kind specified is element declaration, **then** the property is the {element declarations}.

1.4 **If** the kind specified is attribute group, **then** the property is the {attribute group definitions}.

1.5 **If** the kind specified is model group, **then** the property is the {model group definitions}.

1.6 **If** the kind specified is notation declaration, **then** the property is the {notation declarations}.

2 its {local name} matches the ·local name· of the ·QName·;

3 its {target namespace} is identical to the ·namespace name· of the ·QName·;

4 its ·namespace name· is either the target namespace of the schema document containing the ·QName· or that schema document contains an <import> element information item the ·actual value· of whose `namespace` [attribute] is identical to that ·namespace name·.

3.15.4 Validation Rules for Schemas as a Whole

As the discussion above at Schema Component Details (§3) makes clear, at the level of schema components and ·validation·, reference to components by name is normally not involved. In a few cases, however, qualified names appearing in information items being ·validated· must be resolved to schema components by such lookup. The following constraint is appealed to in these cases.

Validation Rule: QName resolution (Instance)

A pair of a local name and a namespace name (or ·absent·) resolve to a schema component of a specified kind in the context of ·validation· by appeal to the appropriate property of the schema being used for the ·assessment·. Each such property indexes components by name. The property to use is determined by the kind of component specified, that is, the appropriate **case** among the following must be true:

1 **If** the kind specified is simple or complex type definition, **then** the property is the {type definitions}.

2 **If** the kind specified is attribute declaration, **then** the property is the {attribute declarations}.

3 **If** the kind specified is element declaration, **then** the property is the {element declarations}.

4 **If** the kind specified is attribute group, **then** the property is the {attribute group definitions}.

5 **If** the kind specified is model group, **then** the property is the {model group definitions}.

6 **If** the kind specified is notation declaration, **then** the property is the {notation declarations}.

The component resolved to is the entry in the table whose {local name} matches the local name of the pair and whose {target namespace} is identical to the namespace name of the pair.

3.15.5 Schema Information Set Contributions

Schema Information Set Contribution: Schema Information

Schema components provide a wealth of information about the basis of ·assessment·, which may well be of relevance to subsequent processing. Reflecting component structure into a form suitable for inclusion in the post-schema-validation infoset is the way this specification provides for making this information available.

Accordingly, [Definition:] by an **item isomorphic** to a component is meant an information item whose type is equivalent to the component's, with one property per property of the component, with the same name, and value either the same atomic value, or an information item corresponding in the same way to its component value, recursively, as necessary.

Processors must add a property in the post-schema-validation infoset to the element information item at which ·assessment· began, as follows:

PSVI Contributions for element information items

`[schema information]`
A set of **namespace schema information** information items, one for each namespace name which appears as the {target namespace} of any schema component in the schema used for that assessment, and one for ·absent· if any schema component in the schema had no {target namespace}. Each **namespace schema information** information item has the following properties and values:

PSVI Contributions for namespace schema information information items

`[schema namespace]`
A namespace name or ·absent·.

`[schema components]`
A (possibly empty) set of schema component information items, each one an ·item isomorphic· to a component whose {target namespace} is the sibling [schema namespace] property above, drawn from the schema used for ·assessment·.

`[schema documents]`
A (possibly empty) set of **schema document** information items, with properties and values as follows, for each schema document which contributed components to the schema, and whose `targetNamespace` matches the sibling [schema namespace] property above (or whose `targetNamespace` was ·absent· but that contributed components to that namespace by being <include>d by a schema document with that `targetNamespace` as per Assembling a schema for a single target namespace from multiple schema definition documents (§4.2.1)):

PSVI Contributions for schema document information items

`[document location]`
Either a URI reference, if available, otherwise ·absent·

`[document]`
A document information item, if available, otherwise ·absent·.

The {schema components} property is provided for processors which wish to provide a single access point to the components of the schema which was used during ·assessment·. Lightweight processors are free to leave it empty, but if it *is* provided, it must contain at a minimum all the top-level (i.e. named) components which actually figured in the ·assessment·, either directly or (because an anonymous component which figured is contained within) indirectly.

Schema Information Set Contribution: ID/IDREF Table

In the post-schema-validation infoset a set of **ID/IDREF binding** information items is associated with the ·validation root· element information item:

PSVI Contributions for element information items

`[ID/IDREF table]`

A (possibly empty) set of **ID/IDREF binding** information items, as specified below.

[Definition:] Let the **eligible item set** be the set of consisting of every attribute or element information item for which **all** of the following are true

1 its [validation context] is the ·validation root·;

2 it was successfully ·validated· with respect to an attribute declaration as per Attribute Locally Valid (§3.2.4) or element declaration as per Element Locally Valid (Element) (§3.3.4) (as appropriate) whose attribute {type definition} or element {type definition} (respectively) is the built-in ID, IDREF or IDREFS simple type definition or a type derived from one of them.

Then there is one **ID/IDREF binding** in the [ID/IDREF table] for every distinct string which is**one** of the following:

1 the ·actual value· of a member of the ·eligible item set· whose type definition is or is derived from ID or IDREF;

2 one of the items in the ·actual value· of a member of the ·eligible item set· whose type definition is or is derived from IDREFS.

Each **ID/IDREF binding** has properties as follows:

PSVI Contributions for ID/IDREF binding information items

`[id]`
The string identified above.

`[binding]`
A set consisting of every element information item for which **all** of the following are true

1 its [validation context] is the ·validation root·;

2 it has an attribute information item in its [attributes] or an element information item in its [children] which was ·validated· by the built-in ID simple type definition or a type derived from it whose [schema normalized value] is the [id] of this **ID/IDREF binding**.

The net effect of the above is to have one entry for every string used as an id, whether by declaration or by reference, associated with those elements, if any, which actually purport to have that id. See Validation Root Valid (ID/IDREF) (§3.3.4) above for the validation rule which actually checks for errors here.

> **NOTE:** The **ID/IDREF binding** information item, unlike most other aspects of this specification, is essentially an internal bookkeeping mechanism. It is introduced to support the definition of Validation Root Valid (ID/IDREF) (§3.3.4) above. Accordingly, conformant processors may, but are *not* required to, expose it in the post-schema-validation infoset. In other words, the above constraint may be read as saying ·assessment· proceeds *as if* such an infoset item existed.

3.15.6 Constraints on Schemas as a Whole

All schemas (see Schemas as a Whole (§3.15)) must satisfy the following constraint.

Schema Component Constraint: Schema Properties Correct

All of the following must be true:

1 The values of the properties of a schema must be as described in the property tableau in The Schema Itself (§3.15.1), modulo the impact of Missing Sub-components (§5.3);

2 Each of the {type definitions}, {element declarations}, {attribute group definitions}, {model group definitions} and {notation declarations} must not contain two or more schema components with the same {name} and {target namespace}.

4 Schemas and Namespaces: Access and Composition

This chapter defines the mechanisms by which this specification establishes the necessary precondition for ·assessment·, namely access to one or more schemas. This chapter also sets out in detail the relationship between schemas and namespaces, as well as mechanisms for modularization of schemas, including provision for incorporating definitions and declarations from one schema in another, possibly with modifications.

Conformance (§2.4) describes three levels of conformance for schema processors, and Schemas and Schema-validity Assessment (§5) provides a formal definition of ·assessment·. This section sets out in detail the 3-layer architecture implied by the three conformance levels. The layers are:

1. The ·assessment· core, relating schema components and instance information items;
2. Schema representation: the connections between XML representations and schema components, including the relationships between namespaces and schema components;
3. XML Schema web-interoperability guidelines: instance->schema and schema->schema connections for the WWW.

Layer 1 specifies the manner in which a schema composed of schema components can be applied to in the ·assessment· of an instance element information item. Layer 2 specifies the use of <schema> elements in XML documents as the standard XML representation for schema information in a broad range of computer systems and execution environments. To support interoperation over the World Wide Web in particular, layer 3 provides a set of conventions for schema reference on the Web. Additional details on each of the three layers is provided in the sections below.

4.1 Layer 1: Summary of the Schema-validity Assessment Core

The fundamental purpose of the ·assessment· core is to define ·assessment· for a single element information item and its descendants with respect to a complex type definition. All processors are required to implement this core predicate in a manner which conforms exactly to this specification.

·assessment· is defined with reference to an ·XML Schema· (note *not* a ·schema document·) which consists of (at a minimum) the set of schema components (definitions and declarations) required for that ·assessment·. This is not a circular definition, but rather a *post facto* observation: no element information item can be fully assessed unless all the components required by any aspect of its (potentially recursive) ·assessment· are present in the schema.

As specified above, each schema component is associated directly or indirectly with a target namespace, or explicitly with no namespace. In the case of multi-namespace documents, components for more than one target namespace will co-exist in a schema.

Processors have the option to assemble (and perhaps to optimize or pre-compile) the entire schema prior to the start of an ·assessment· episode, or to gather the schema lazily as individual components are required. In all cases it is required that:

- The processor succeed in locating the ·schema components· transitively required to complete an ·assessment· (note that components derived from ·schema documents· can be integrated with components obtained through other means);
- no definition or declaration changes once it has been established;
- if the processor chooses to acquire declarations and definitions dynamically, that there be no side effects of such dynamic acquisition that would cause the results of ·assessment· to differ from that which would have been obtained from the same schema components acquired in bulk.

  NOTE: the ·assessment· core is defined in terms of schema components at the abstract level, and no mention is made of the schema definition syntax (i.e. <schema>). Although many processors will acquire schemas in this format, others may operate on compiled representations, on a programmatic representation as exposed in some programming language, etc.

The obligation of a schema-aware processor as far as the ·assessment· core is concerned is to implement one or more of the options for ·assessment· given below in Assessing Schema-Validity (§5.2). Neither the choice of element information item for that ·assessment·, nor which of the means of initiating ·assessment· are used, is within the scope of this specification.

Although ·assessment· is defined recursively, it is also intended to be implementable in streaming processors. Such processors may choose to incrementally assemble the schema during processing in response, for example, to encountering new namespaces. The implication of the invariants expressed above is that such incremental assembly must result in an ·assessment· outcome that is the *same* as would be given if ·assessment· was undertaken again with the final, fully assembled schema.

4.2 Layer 2: Schema Documents, Namespaces and Composition

The sub-sections of Schema Component Details (§3) define an XML representation for type definitions and element declarations and so on, specifying their target namespace and collecting them into schema documents. The two following sections relate to assembling a complete schema for ·assessment· from multiple sources. They should *not* be understood as a form of text substitution, but rather as providing mechanisms for distributed definition of schema components, with appropriate schema-specific semantics.

> **NOTE:** The core ·assessment· architecture requires that a complete schema with all the necessary declarations and definitions be available. This may involve resolving both instance->schema and schema->schema references. As observed earlier in Conformance (§2.4), the precise mechanisms for resolving such references are expected to evolve over time. In support of such evolution, this specification observes the design principle that references from one schema document to a schema use mechanisms that directly parallel those used to reference a schema from an instance document.
>
> **NOTE:** In the sections below, "schemaLocation" really belongs at layer 3. For convenience, it is documented with the layer 2 mechanisms of import and include, with which it is closely associated.

4.2.1 Assembling a schema for a single target namespace from multiple schema definition documents

Schema components for a single target namespace can be assembled from several ·schema documents·, that is several <schema> element information items:

XML Representation Summary: `include` Element Information Item

```
<include
  id = ID
  schemaLocation = anyURI
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</include>
```

A <schema> information item may contain any number of <include> elements. Their `schemaLocation` attributes, consisting of a URI reference, identify other ·schema documents·, that is <schema> information items.

The ·XML Schema· corresponding to <schema> contains not only the components corresponding to its definition and declaration [children], but also all the components of all the ·XML Schemas· corresponding

to any <include>d schema documents. Such included schema documents must either (a) have the same `targetNamespace` as the <include>ing schema document, or (b) no `targetNamespace` at all, in which case the <include>d schema document is converted to the <include>ing schema document's `targetNamespace`.

Schema Representation Constraint: Inclusion Constraints and Semantics

In addition to the conditions imposed on <include> element information items by the schema for schemas, **all** of the following must be true:

1 If the ·actual value· of the `schemaLocation` [attribute] successfully resolves **one** of the following must be true:

1.1 It resolves to (a fragment of) a resource which is an XML document (of type `application/xml` or `text/xml` with an XML declaration for preference, but this is not required), which in turn corresponds to a <schema> element information item in a well-formed information set, which in turn corresponds to a valid schema.

1.2 It resolves to a <schema> element information item in a well-formed information set, which in turn corresponds to a valid schema.

In either case call the <include>d <schema> item **SII**, the valid schema **I** and the <include>ing item's parent <schema> item **SII'**.

2 **One** of the following must be true:

2.1 **SII** has a `targetNamespace` [attribute], and its ·actual value· is identical to the ·actual value· of the `targetNamespace` [attribute] of **SII'** (which must have such an [attribute]).

2.2 Neither **SII** nor **SII'** have a `targetNamespace` [attribute].

2.3 **SII** has no `targetNamespace` [attribute] (but **SII'** does).

3 The appropriate **case** among the following must be true:

3.1 **If** clause 2.1 or clause 2.2 above is satisfied, **then** the schema corresponding to **SII'** must include not only definitions or declarations corresponding to the appropriate members of its own [children], but also components identical to all the ·schema components· of **I**.

3.2 **If** clause 2.3 above is satisfied, **then** the schema corresponding to the <include>d item's parent <schema> must include not only definitions or declarations corresponding to the appropriate members of its own [children], but also components identical to all the ·schema components· of **I**, except that anywhere the ·absent· target namespace name would have appeared, the ·actual value· of the `targetNamespace` [attribute] of **SII'** is used. In particular, it replaces ·absent· in the following places:

3.2.1 The {target namespace} of named schema components, both at the top level and (in the case of nested type definitions and nested attribute and element declarations whose `code` was *qualified*) nested within definitions;

3.2.2 The {namespace constraint} of a wildcard, whether negated or not;

It is *not* an error for the ·actual value· of the `schemaLocation` [attribute] to fail to resolve it all, in which case no corresponding inclusion is performed. It *is* an error for it to resolve but the rest of clause 1 above to fail to be satisfied. Failure to resolve may well cause less than complete ·assessment· outcomes, of course.

> **NOTE:** As discussed in Missing Sub-components (§5.3), ·QName·s in XML representations may fail to ·resolve·, rendering components incomplete and unusable because of missing subcomponents. During schema construction, implementations are likely to retain ·QName· values for such references, in case subsequent processing provides a referent. ·Absent· target ·namespace name·s of such as-yet unresolved reference ·QName·s in <include>d components should also be converted if clause 3.2 is satisfied.

NOTE: The above is carefully worded so that multiple <include>ing of the same schema document will not constitute a violation of clause 2 of Schema Properties Correct (§3.15.6), but applications are allowed, indeed encouraged, to avoid <include>ing the same schema document more than once to forestall the necessity of establishing identity component by component.

4.2.2 Including modified component definitions

In order to provide some support for evolution and versioning, it is possible to incorporate components corresponding to a schema document *with modifications*. The modifications have a pervasive impact, that is, only the redefined components are used, even when referenced from other incorporated components, whether redefined themselves or not.

XML Representation Summary: `redefine` Element Information Item

```
<redefine
  id = ID
  schemaLocation = anyURI
  {any attributes with non-schema namespace . . .}>
  Content: (annotation | (simpleType | complexType | group | attributeGroup))*
</redefine>
```

A <schema> information item may contain any number of <redefine> elements. Their `schemaLocation` attributes, consisting of a URI reference, identify other ·schema documents·, that is <schema> information items.

The ·XML Schema· corresponding to <schema> contains not only the components corresponding to its definition and declaration [children], but also all the components of all the ·XML Schemas· corresponding to any <redefine>d schema documents. Such schema documents must either (a) have the same `targetNamespace` as the <redefine>ing schema document, or (b) no `targetNamespace` at all, in which case the <redefine>d schema document is converted to the <redefine>ing schema document's `targetNamespace`.

The definitions within the <redefine> element itself are restricted to be redefinitions of components from the <redefine>d schema document, *in terms of themselves*. That is,

- Type definitions must use themselves as their base type definition;
- Attribute group definitions and model group definitions must be supersets or subsets of their original definitions, either by including exactly one reference to themselves or by containing only (possibly restricted) components which appear in a corresponding way in their <redefine>d selves.

Not all the components of the <redefine>d schema document need be redefined.

This mechanism is intended to provide a declarative and modular approach to schema modification, with functionality no different except in scope from what would be achieved by wholesale text copying and redefinition by editing. In particular redefining a type is not guaranteed to be side-effect free: it may have unexpected impacts on other type definitions which are based on the redefined one, even to the extent that some such definitions become ill-formed.

> **NOTE:** The pervasive impact of redefinition reinforces the need for implementations to adopt some form of lazy or 'just-in-time' approach to component construction, which is also called for in order to avoid inappropriate dependencies on the order in which definitions and references appear in (collections of) schema documents.

Example

```
v1.xsd:
 <xs:complexType name="personName">
  <xs:sequence>
   <xs:element name="title" minOccurs="0"/>
   <xs:element name="forename" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
```

```
  </xs:complexType>

  <xs:element name="addressee" type="personName"/>

 v2.xsd:
  <xs:redefine schemaLocation="v1.xsd">
   <xs:complexType name="personName">
    <xs:complexContent>
     <xs:extension base="personName">
      <xs:sequence>
       <xs:element name="generation" minOccurs="0"/>
      </xs:sequence>
     </xs:extension>
    </xs:complexContent>
   </xs:complexType>
  </xs:redefine>

  <xs:element name="author" type="personName"/>
```

The schema corresponding to `v2.xsd` has everything specified by `v1.xsd`, with the `personName` type redefined, as well as everything it specifies itself. According to this schema, elements constrained by the `personName` type may end with a `generation` element. This includes not only the `author` element, but also the `addressee` element.

Schema Representation Constraint: Redefinition Constraints and Semantics

In addition to the conditions imposed on <redefine> element information items by the schema for schemas **all** of the following must be true:

1 If there are any element information items among the [children] other than <annotation> then the ·actual value· of the `schemaLocation` [attribute] must successfully resolve.

2 If the ·actual value· of the `schemaLocation` [attribute] successfully resolves **one** of the following must be true:

2.1 it resolves to (a fragment of) a resource which is an XML document (see clause 1.1), which in turn corresponds to a <schema> element information item in a well-formed information set, which in turn corresponds to a valid schema.

2.2 It resolves to a <schema> element information item in a well-formed information set, which in turn corresponds to a valid schema.

In either case call the <redefine>d <schema> item **SII**, the valid schema **I** and the <redefine>ing item's parent <schema> item **SII'**.

3 **One** of the following must be true:

3.1 **SII** has a `targetNamespace` [attribute], and its ·actual value· is identical to the ·actual value· of the `targetNamespace` [attribute] of **SII'** (which must have such an [attribute]).

3.2 Neither **SII** nor **SII'** have a `targetNamespace` [attribute].

3.3 **SII** has no `targetNamespace` [attribute] (but **SII'** does).

4 The appropriate **case** among the following must be true:

4.1 **If** clause 3.1 or clause 3.2 above is satisfied, **then** the schema corresponding to **SII'** must include not only definitions or declarations corresponding to the appropriate members of its own [children], but also components identical to all the ·schema components· of **I**, with the exception of those explicitly redefined (see Individual Component Redefinition (§4.2.2) below).

4.2 **If** clause 3.3 above is satisfied, **then** the schema corresponding to **SII'** must include not only definitions or declarations corresponding to the appropriate members of its own [children], but also components identical to all the ·schema components· of **I**, with the exception of those explicitly redefined (see Individual Component Redefinition (§4.2.2) below), except that anywhere the ·absent· target namespace name would have appeared, the ·actual value· of the `targetNamespace` [attribute] of **SII'** is used (see clause 3.2 in Inclusion Constraints and Semantics (§4.2.1) for details).

5 Within the [children], each <simpleType> must have a <restriction> among its [children] and each

<complexType> must have a `restriction` or `extension` among its grand-[children] the ·actual value· of whose `base` [attribute] must be the same as the ·actual value· of its own `name` attribute plus target namespace;

6 Within the [children], for each <group> the appropriate **case** among the following must be true:

6.1 **If** it has a <group> among its contents at some level the ·actual value· of whose `ref` [attribute] is the same as the ·actual value· of its own `name` attribute plus target namespace, **then all** of the following must be true:

6.1.1 It must have exactly one such group.

6.1.2 The ·actual value· of both that group's `minOccurs` and `maxOccurs` [attribute] must be `1` (or ·absent·).

6.2 **If** it has no such self-reference, **then all** of the following must be true:

6.2.1 The ·actual value· of its own `name` attribute plus target namespace must successfully ·resolve· to a model group definition in **I**.

6.2.2 The {model group} of the model group definition which corresponds to it per XML Representation of Model Group Definition Schema Components (§3.7.2) must be a ·valid restriction· of the {model group} of that model group definition in **I**, as defined in Particle Valid (Restriction) (§3.9.6).

7 Within the [children], for each <attributeGroup> the appropriate **case** among the following must be true:

7.1 **If** it has an <attributeGroup> among its contents the ·actual value· of whose `ref` [attribute] is the same as the ·actual value· of its own `name` attribute plus target namespace, **then** it must have exactly one such group.

7.2 **If** it has no such self-reference, **then all** of the following must be true:

7.2.1 The ·actual value· of its own `name` attribute plus target namespace must successfully ·resolve· to an attribute group definition in **I**.

7.2.2 The {attribute uses} and {attribute wildcard} of the attribute group definition which corresponds to it per XML Representation of Attribute Group Definition Schema Components (§3.6.2) must be ·valid restrictions· of the {attribute uses} and {attribute wildcard} of that attribute group definition in **I**, as defined in clause 2, clause 3 and clause 4 of Derivation Valid (Restriction, Complex) (§3.4.6) (where references to the base type definition are understood as references to the attribute group definition in **I**).

> **NOTE:** An attribute group restrictively redefined per clause 7.2 corresponds to an attribute group whose {attribute uses} consist all and only of those attribute uses corresponding to <attribute>s explicitly present among the [children] of the <redefine>ing <attributeGroup>. No inheritance from the <redefine>d attribute group occurs. Its {attribute wildcard} is similarly based purely on an explicit <anyAttribute>, if present.

Schema Representation Constraint: Individual Component Redefinition

Corresponding to each non-<annotation> member of the [children] of a <redefine> there are one or two schema components in the <redefine>ing schema:

1 The <simpleType> and <complexType> [children] information items each correspond to two components:

1.1 One component which corresponds to the top-level definition item with the same `name` in the <redefine>d schema document, as defined in Schema Component Details (§3), except that its {name} is ·absent·;

1.2 One component which corresponds to the information item itself, as defined in Schema Component Details (§3), except that its {base type definition} is the component defined in 1.1 above.

This pairing ensures the coherence constraints on type definitions are respected, while at the same time achieving the desired effect, namely that references to names of redefined components in both the <redefine>ing and <redefine>d schema documents resolve to the redefined component as specified in 1.2 above.

2 The <group> and <attributeGroup> [children] each correspond to a single component, as defined

in Schema Component Details (§3), except that if and when a self-reference based on a `ref` [attribute] whose ·actual value· is the same as the item's `name` plus target namespace is resolved, a component which corresponds to the top-level definition item of that name and the appropriate kind in **I** is used.

In all cases there must be a top-level definition item of the appropriate name and kind in the <redefine>d schema document.

NOTE: The above is carefully worded so that multiple equivalent <redefine>ing of the same schema document will not constitute a violation of clause 2 of Schema Properties Correct (§3.15.6), but applications are allowed, indeed encouraged, to avoid <redefine>ing the same schema document in the same way more than once to forestall the necessity of establishing identity component by component (although this will have to be done for the individual redefinitions themselves).

4.2.3 References to schema components across namespaces

As described in XML Schema Abstract Data Model (§2.2), every top-level schema component is associated with a target namespace (or, explicitly, with none). This section sets out the exact mechanism and syntax in the XML form of schema definition by which a reference to a foreign component is made, that is, a component with a different target namespace from that of the referring component.

Two things are required: not only a means of addressing such foreign components but also a signal to schema-aware processors that a schema document contains such references:

XML Representation Summary: `import` Element Information Item

```
<import
  id = ID
  namespace = anyURI
  schemaLocation = anyURI
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</import>
```

The <import> element information item identifies namespaces used in external references, i.e. those whose ·QName· identifies them as coming from a different namespace (or none) than the enclosing schema document's `targetNamespace`. The ·actual value· of its `namespace` [attribute] indicates that the containing schema document may contain qualified references to schema components in that namespace (via one or more prefixes declared with namespace declarations in the normal way). If that attribute is absent, then the import allows unqualified reference to components with no target namespace. Note that components to be imported need not be in the form of a ·schema document·; the processor is free to access or construct components using means of its own choosing.

The ·actual value· of the `schemaLocation`, if present, gives a hint as to where a serialization of a ·schema document· with declarations and definitions for that namespace (or none) may be found. When no `schemaLocation` [attribute] is present, the schema author is leaving the identification of that schema to the instance, application or user, via the mechanisms described below in Layer 3: Schema Document Access and Web-interoperability (§4.3). When a `schemaLocation` is present, it must contain a single URI reference which the schema author warrants will resolve to a serialization of a ·schema document· containing the component(s) in the <import>ed namespace referred to elsewhere in the containing schema document.

NOTE: Since both the `namespace` and `schemaLocation` [attribute] are optional, a bare `<import/>` information item is allowed. This simply allows unqualified reference to foreign components with no target namespace without giving any hints as to where to find them.

Example

The same namespace may be used both for real work, and in the course of defining schema components in terms of foreign components:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:html="http://www.w3.org/1999/xhtml"
        targetNamespace="uri:mywork" xmlns:my="uri:mywork">

 <import namespace="http://www.w3.org/1999/xhtml"/>

 <annotation>
  <documentation>
   <html:p>[Some documentation for my schema]</html:p>
  </documentation>
 </annotation>

 . . .

 <complexType name="myType">
  <sequence>
   <element ref="html:p" minOccurs="0"/>
  </sequence>
  . . .
 </complexType>

 <element name="myElt" type="my:myType"/>
</schema>
```

The treatment of references as ·QNames· implies that since (with the exception of the schema for schemas) the target namespace and the XML Schema namespace differ, without massive redeclaration of the default namespace *either* internal references to the names being defined in a schema document *or* the schema declaration and definition elements themselves must be explicitly qualified. This example takes the first option -- most other examples in this specification have taken the second.

Schema Representation Constraint: Import Constraints and Semantics

In addition to the conditions imposed on <import> element information items by the schema for schemas **all** of the following must be true:

1 The appropriate **case** among the following must be true:

1.1 **If** the `namespace` [attribute] is present, **then** its ·actual value· must not match the ·actual value· of the enclosing <schema>'s `targetNamespace` [attribute].

1.2 **If** the `namespace` [attribute] is not present, **then** the enclosing <schema> must have a `targetNamespace` [attribute]

2 If the application schema reference strategy using the ·actual value·s of the `schemaLocation` and `namespace` [attributes], provides a referent, as defined by Schema Document Location Strategy (§4.3.2), **one** of the following must be true:

2.1 The referent is (a fragment of) a resource which is an XML document (see clause 1.1), which in turn corresponds to a <schema> element information item in a well-formed information set, which in turn corresponds to a valid schema.

2.2 The referent is a <schema> element information item in a well-formed information set, which in turn corresponds to a valid schema.

In either case call the <schema> item **SII** and the valid schema **I**.

3 The appropriate **case** among the following must be true:

3.1 **If** there is a `namespace` [attribute], **then** its ·actual value· must be identical to the ·actual value· of the `targetNamespace` [attribute] of **SII**.

3.2 **If** there is no `namespace` [attribute], **then SII** must have no `targetNamespace` [attribute]

It is *not* an error for the application schema reference strategy to fail. It *is* an error for it to resolve but

the rest of clause 2 above to fail to be satisfied. Failure to find a referent may well cause less than complete ·assessment· outcomes, of course.

The ·schema components· (that is {type definitions}, {attribute declarations}, {element declarations}, {attribute group definitions}, {model group definitions}, {notation declarations}) of a schema corresponding to a <schema> element information item with one or more <import> element information items must include not only definitions or declarations corresponding to the appropriate members of its [children], but also, for each of those <import> element information items for which clause 2 above is satisfied, a set of ·schema components· identical to all the ·schema components· of **I**.

> **NOTE:** The above is carefully worded so that multiple <import>ing of the same schema document will not constitute a violation of clause 2 of Schema Properties Correct (§3.15.6), but applications are allowed, indeed encouraged, to avoid <import>ing the same schema document more than once to forestall the necessity of establishing identity component by component. Given that the `schemaLocation` [attribute] is only a hint, it is open to applications to ignore all but the first <import> for a given namespace, regardless of the ·actual value· of `schemaLocation`, but such a strategy risks missing useful information when new `schemaLocations` are offered.

4.3 Layer 3: Schema Document Access and Web-interoperability

Layers 1 and 2 provide a framework for ·assessment· and XML definition of schemas in a broad variety of environments. Over time, a range of standards and conventions may well evolve to support interoperability of XML Schema implementations on the World Wide Web. Layer 3 defines the minimum level of function required of all conformant processors operating on the Web: it is intended that, over time, future standards (e.g. XML Packages) for interoperability on the Web and in other environments can be introduced without the need to republish this specification.

4.3.1 Standards for representation of schemas and retrieval of schema documents on the Web

For interoperability, serialized ·schema documents·, like all other Web resources, may be identified by URI and retrieved using the standard mechanisms of the Web (e.g. http, https, etc.) Such documents on the Web must be part of XML documents (see clause 1.1), and are represented in the standard XML schema definition form described by layer 2 (that is as <schema> element information items).

> **NOTE:** there will often be times when a schema document will be a complete XML 1.0 document whose document element is <schema>. There will be other occasions in which <schema> items will be contained in other documents, perhaps referenced using fragment and/or XPointer notation.

> **NOTE:** The variations among server software and web site administration policies make it difficult to recommend any particular approach to retrieval requests intended to retrieve serialized ·schema documents·. An `Accept` header of `application/xml, text/xml; q=0.9, */*` is perhaps a reasonable starting point.

4.3.2 How schema definitions are located on the Web

As described in Layer 1: Summary of the Schema-validity Assessment Core (§4.1), processors are responsible for providing the schema components (definitions and declarations) needed for ·assessment·. This section introduces a set of normative conventions to facilitate interoperability for instance and schema documents retrieved and processed from the Web.

> **NOTE:** As discussed above in Layer 2: Schema Documents, Namespaces and Composition

(§4.2), other non-Web mechanisms for delivering schemas for ·assessment· may exist, but are outside the scope of this specification.

Processors on the Web are free to undertake ·assessment· against arbitrary schemas in any of the ways set out in Assessing Schema-Validity (§5.2). However, it is useful to have a common convention for determining the schema to use. Accordingly, general-purpose schema-aware processors (i.e. those not specialized to one or a fixed set of pre-determined schemas) undertaking ·assessment· of a document on the web must behave as follows:

- unless directed otherwise by the user, ·assessment· is undertaken on the document element information item of the specified document;
- unless directed otherwise by the user, the processor is required to construct a schema corresponding to a schema document whose `targetNamespace` is identical to the namespace name, if any, of the element information item on which ·assessment· is undertaken.

The composition of the complete schema for use in ·assessment· is discussed in Layer 2: Schema Documents, Namespaces and Composition (§4.2) above. The means used to locate appropriate schema document(s) are processor and application dependent, subject to the following requirements:

1. Schemas are represented on the Web in the form specified above in Standards for representation of schemas and retrieval of schema documents on the Web (§4.3.1);
2. The author of a document uses namespace declarations to indicate the intended interpretation of names appearing therein; there may or may not be a schema retrievable via the namespace name. Accordingly whether a processor's default behavior is or is not to attempt such dereferencing, it must always provide for user-directed overriding of that default.

   NOTE: Experience suggests that it is not in all cases safe or desirable from a performance point of view to dereference namespace names as a matter of course. User community and/or consumer/provider agreements may establish circumstances in which such dereference is a sensible default strategy: this specification allows but does not require particular communities to establish and implement such conventions. Users are always free to supply namespace names as schema location information when dereferencing *is* desired: see below.

3. On the other hand, in case a document author (human or not) created a document with a particular schema in view, and warrants that some or all of the document is conforms to that schema, the `schemaLocation` and `noNamespaceSchemaLocation` [attributes] (in the XML Schema instance namespace, that is, `http://www.w3.org/2001/XMLSchema-instance`) (hereafter `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation`) are provided. The first records the author's warrant with pairs of URI references (one for the namespace name, and one for a hint as to the location of a schema document defining names for that namespace name). The second similarly provides a URI reference as a hint as to the location of a schema document with no `targetNamespace` [attribute].

   Unless directed otherwise, for example by the invoking application or by command line option, processors should attempt to dereference each schema document location URI in the ·actual value· of such `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` [attributes], see details below.
4. `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` [attributes] can occur on any element. However, it is an error if such an attribute occurs *after* the first appearance of an element or attribute information item within an element information item initially ·validated· whose [namespace name] it addresses. According to the rules of Layer 1: Summary of the Schema-validity Assessment Core (§4.1), the corresponding schema may be lazily assembled, but is otherwise stable throughout ·assessment·. Although schema location attributes can occur on any element, and can be processed incrementally as discovered, their effect is essentially global to the ·assessment·. Definitions and declarations remain in effect beyond the scope of the element on which the binding is declared.

Example

Multiple schema bindings can be declared using a single attribute. For example consider a stylesheet:

```
<stylesheet xmlns="http://www.w3.org/1999/XSL/Transform"
            xmlns:html="http://www.w3.org/1999/xhtml"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.w3.org/1999/XSL/Transform
                                http://www.w3.org/1999/XSL/Transform.xsd
                                http://www.w3.org/1999/xhtml
                                http://www.w3.org/1999/xhtml.xsd">
```

The namespace names used in `schemaLocation` can, but need not be identical to those actually qualifying the element within whose start tag it is found or its other attributes. For example, as above, all schema location information can be declared on the document element of a document, if desired, regardless of where the namespaces are actually used.

Schema Representation Constraint: Schema Document Location Strategy

Given a namespace name (or none) and (optionally) a URI reference from `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation`, schema-aware processors may implement any combination of the following strategies, in any order:

1 Do nothing, for instance because a schema containing components for the given namespace name is already known to be available, or because it is known in advance that no efforts to locate schema documents will be successful (for example in embedded systems);

2 Based on the location URI, identify an existing schema document, either as a resource which is an XML document or a <schema> element information item, in some local schema repository;

3 Based on the namespace name, identify an existing schema document, either as a resource which is an XML document or a <schema> element information item, in some local schema repository;

4 Attempt to resolve the location URI, to locate a resource on the web which is or contains or references a <schema> element;

5 Attempt to resolve the namespace name to locate such a resource.

Whenever possible configuration and/or invocation options for selecting and/or ordering the implemented strategies should be provided.

Improved or alternative conventions for Web interoperability can be standardized in the future without reopening this specification. For example, the W3C is currently considering initiatives to standardize the packaging of resources relating to particular documents and/or namespaces: this would be an addition to the mechanisms described here for layer 3. This architecture also facilitates innovation at layer 2: for example, it would be possible in the future to define an additional standard for the representation of schema components which allowed e.g. type definitions to be specified piece by piece, rather than all at once.

5 Schemas and Schema-validity Assessment

The architecture of schema-aware processing allows for a rich characterization of XML documents: schema validity is not a binary predicate.

This specification distinguishes between errors in schema construction and structure, on the one hand, and schema validation outcomes, on the other.

5.1 Errors in Schema Construction and Structure

Before ·assessment· can be attempted, a schema is required. Special-purpose applications are free to determine a schema for use in ·assessment· by whatever means are appropriate, but general purpose processors should implement the strategy set out in Schema Document Location Strategy (§4.3.2), starting with the namespaces declared in the document whose ·assessment· is being undertaken, and the

·actual value·s of the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` [attributes] thereof, if any, along with an other information about schema identity or schema document location provided by users in application-specific ways, if any.

It is an error if a schema and all the components which are the value of any of its properties, recursively, fail to satisfy all the relevant Constraints on Schemas set out in the last section of each of the subsections of Schema Component Details (§3).

If a schema is derived from one or more schema documents (that is, one or more <schema> element information items) based on the correspondence rules set out in Schema Component Details (§3) and Schemas and Namespaces: Access and Composition (§4), two additional conditions hold:

- It is an error if any such schema document would not be fully valid with respect to a schema corresponding to the Schema for Schemas (normative) (§A), that is, following schema-validation with such a schema, the <schema> element information items would have a [validation attempted] property with value *full* or *partial* and a [validity] property with value *valid*.
- It is an error if any such schema document is or contains any element information items which violate any of the relevant Schema Representation Constraints set out in Schema Representation Constraints (§C.3).

The three cases described above are the only types of error which this specification defines. With respect to the processes of the checking of schema structure and the construction of schemas corresponding to schema documents, this specification imposes no restrictions on processors after an error is detected. However ·assessment· with respect to schema-like entities which do *not* satisfy all the above conditions is incoherent. Accordingly, conformant processors must not attempt to undertake ·assessment· using such non-schemas.

5.2 Assessing Schema-Validity

With a schema which satisfies the conditions expressed in Errors in Schema Construction and Structure (§5.1) above, the schema-validity of an element information item can be assessed. Three primary approaches to this are possible:

1 The user or application identifies a complex type definition from among the {type definitions} of the schema, and appeals to Schema-Validity Assessment (Element) (§3.3.4) (clause 1.2);
2 The user or application identifies a element declaration from among the {element declarations} of the schema, checks that its {name} and {target namespace} match the [local name] and [namespace name] of the item, and appeals to Schema-Validity Assessment (Element) (§3.3.4) (clause 1.1);
3 The processor starts from Schema-Validity Assessment (Element) (§3.3.4) with no stipulated declaration or definition, and either ·strict· or ·lax· assessment ensues, depending on whether or not the element information and the schema determine either an element declaration (by name) or a type definition (via `xsi:type`) or not.

The outcome of this effort, in any case, will be manifest in the [validation attempted] and [validity] properties on the element information item and its [attributes] and [children], recursively, as defined by Assessment Outcome (Element) (§3.3.5) and Assessment Outcome (Attribute) (§3.2.5). It is up to applications to decide what constitutes a successful outcome.

Note that every element and attribute information item participating in the ·assessment· will also have a [validation context] property which refers back to the element information item at which ·assessment· began. [Definition:] This item, that is the element information item at which ·assessment· began, is called the **validation root**.

> **NOTE:** This specification does not reconstruct the XML 1.0 notion of *root* in either schemas or instances. Equivalent functionality is provided for at ·assessment· invocation, via clause 2

above.

NOTE: This specification has nothing normative to say about multiple ·assessment· episodes. It should however be clear from the above that if a processor restarts ·assessment· with respect to a post-schema-validation infoset some post-schema-validation infoset contributions from the previous ·assessment· may be overwritten. Restarting nonetheless may be useful, particularly at a node whose [validation attempted] property is *none*, in which case there are three obvious cases in which additional useful information may result:

- ·assessment· was not attempted because of a ·validation· failure, but declarations and/or definitions are available for at least some of the [children] or [attributes];
- ·assessment· was not attempted because a named definition or declaration was missing, but after further effort the processor has retrieved it.
- ·assessment· was not attempted because it was *skip*ped, but the processor has at least some declarations and/or definitions available for at least some of the [children] or [attributes].

5.3 Missing Sub-components

At the beginning of Schema Component Details (§3), attention is drawn to the fact that most kinds of schema components have properties which are described therein as having other components, or sets of other components, as values, but that when components are constructed on the basis of their correspondence with element information items in schema documents, such properties usually correspond to QNames, and the ·resolution· of such QNames may fail, resulting in one or more values of or containing ·absent· where a component is mandated.

If at any time during ·assessment·, an element or attribute information item is being ·validated· with respect to a component of any kind any of whose properties has or contains such an ·absent· value, the ·validation· is modified, as following:

- In the case of attribute information items, the effect is as if clause 1 of Attribute Locally Valid (§3.2.4) had failed;
- In the case of element information items, the effect is as if clause 1 of Element Locally Valid (Element) (§3.3.4) had failed;
- In the case of element information items, processors may choose to continue ·assessment·: see ·lax assessment·.

Because of the value specification for [validation attempted] in Assessment Outcome (Element) (§3.3.5), if this situation ever arises, the document as a whole cannot show a [validation attempted] of *full*.

5.4 Responsibilities of Schema-aware Processors

Schema-aware processors are responsible for processing XML documents, schemas and schema documents, as appropriate given the level of conformance (as defined in Conformance (§2.4)) they support, consistently with the conditions set out above.

A Schema for Schemas (normative)

The XML Schema definition for *XML Schema: Structures* itself is presented here as normative part of the specification, and as an illustrative example of the XML Schema in defining itself with the very constructs that it defines. The names of XML Schema language types, elements, attributes and groups defined here are evocative of their purpose, but are occasionally verbose.

There is some annotation in comments, but a fuller annotation will require the use of embedded documentation facilities or a hyperlinked external annotation for which tools are not yet readily available.

Since an *XML Schema: Structures* is an XML document, it has optional XML and doctype declarations that are provided here for completeness. The root `schema` element defines a new schema. Since this is a schema for *XML Schema: Structures*, the `targetNamespace` references the XML Schema namespace itself.

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- XML Schema schema for XML Schemas: Part 1: Structures -->
<!DOCTYPE xs:schema PUBLIC "-//W3C//DTD XMLSCHEMA 200102//EN" "XMLSchema.
dtd" [

<!-- provide ID type information even for parsers which only read the
     internal subset -->
<!ATTLIST xs:schema          id  ID  #IMPLIED>
<!ATTLIST xs:complexType     id  ID  #IMPLIED>
<!ATTLIST xs:complexContent  id  ID  #IMPLIED>
<!ATTLIST xs:simpleContent   id  ID  #IMPLIED>
<!ATTLIST xs:extension       id  ID  #IMPLIED>
<!ATTLIST xs:element         id  ID  #IMPLIED>
<!ATTLIST xs:group           id  ID  #IMPLIED>
<!ATTLIST xs:all             id  ID  #IMPLIED>
<!ATTLIST xs:choice          id  ID  #IMPLIED>
<!ATTLIST xs:sequence        id  ID  #IMPLIED>
<!ATTLIST xs:any             id  ID  #IMPLIED>
<!ATTLIST xs:anyAttribute    id  ID  #IMPLIED>
<!ATTLIST xs:attribute       id  ID  #IMPLIED>
<!ATTLIST xs:attributeGroup  id  ID  #IMPLIED>
<!ATTLIST xs:unique          id  ID  #IMPLIED>
<!ATTLIST xs:key             id  ID  #IMPLIED>
<!ATTLIST xs:keyref          id  ID  #IMPLIED>
<!ATTLIST xs:selector        id  ID  #IMPLIED>
<!ATTLIST xs:field           id  ID  #IMPLIED>
<!ATTLIST xs:include         id  ID  #IMPLIED>
<!ATTLIST xs:import          id  ID  #IMPLIED>
<!ATTLIST xs:redefine        id  ID  #IMPLIED>
<!ATTLIST xs:notation        id  ID  #IMPLIED>
]>
<xs:schema targetNamespace="http://www.w3.org/2001/XMLSchema"
blockDefault="#all" elementFormDefault="qualified" version="Id: XMLSchema.
xsd,v 1.48 2001/04/24 18:56:39 ht Exp " xmlns:xs="http://www.w3.org/2001/
XMLSchema" xml:lang="EN">

 <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/2001/REC-xmlschema-1-
20010502/structures.html">
   The schema corresponding to this document is normative,
   with respect to the syntactic constraints it expresses in the
   XML Schema language.  The documentation (within &lt;documentation>
elements)
   below, is not normative, but rather highlights important aspects of
   the W3C Recommendation of which this is a part</xs:documentation>
 </xs:annotation>

 <xs:annotation>
   <xs:documentation>
   The simpleType element and all of its members are defined
   in datatypes.xsd</xs:documentation>
 </xs:annotation>
```

```
 <xs:include schemaLocation="datatypes.xsd"/>

 <xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd">
   <xs:annotation>
     <xs:documentation>
       Get access to the xml: attribute groups for xml:lang
       as declared on 'schema' and 'documentation' below
     </xs:documentation>
   </xs:annotation>
 </xs:import>

 <xs:complexType name="openAttrs">
   <xs:annotation>
     <xs:documentation>
       This type is extended by almost all schema types
       to allow attributes from other namespaces to be
       added to user schemas.
     </xs:documentation>
   </xs:annotation>
   <xs:complexContent>
     <xs:restriction base="xs:anyType">
       <xs:anyAttribute namespace="##other" processContents="lax"/>
     </xs:restriction>
   </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="annotated">
   <xs:annotation>
     <xs:documentation>
       This type is extended by all types which allow annotation
       other than &lt;schema&gt; itself
     </xs:documentation>
   </xs:annotation>
   <xs:complexContent>
     <xs:extension base="xs:openAttrs">
       <xs:sequence>
         <xs:element ref="xs:annotation" minOccurs="0"/>
       </xs:sequence>
       <xs:attribute name="id" type="xs:ID"/>
     </xs:extension>
   </xs:complexContent>
 </xs:complexType>

 <xs:group name="schemaTop">
  <xs:annotation>
   <xs:documentation>
   This group is for the
   elements which occur freely at the top level of schemas.
   All of their types are based on the "annotated" type by extension.</xs:
documentation>
  </xs:annotation>
  <xs:choice>
   <xs:group ref="xs:redefinable"/>
   <xs:element ref="xs:element"/>
   <xs:element ref="xs:attribute"/>
   <xs:element ref="xs:notation"/>
  </xs:choice>
 </xs:group>
```

```
 <xs:group name="redefinable">
  <xs:annotation>
   <xs:documentation>
   This group is for the
   elements which can self-redefine (see &lt;redefine> below).</xs:
documentation>
  </xs:annotation>
  <xs:choice>
   <xs:element ref="xs:simpleType"/>
   <xs:element ref="xs:complexType"/>
   <xs:element ref="xs:group"/>
   <xs:element ref="xs:attributeGroup"/>
  </xs:choice>
 </xs:group>

 <xs:simpleType name="formChoice">
  <xs:annotation>
   <xs:documentation>
   A utility type, not for public use</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:NMTOKEN">
   <xs:enumeration value="qualified"/>
   <xs:enumeration value="unqualified"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="reducedDerivationControl">
  <xs:annotation>
   <xs:documentation>
   A utility type, not for public use</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:derivationControl">
   <xs:enumeration value="extension"/>
   <xs:enumeration value="restriction"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="derivationSet">
  <xs:annotation>
   <xs:documentation>
   A utility type, not for public use</xs:documentation>
   <xs:documentation>
   #all or (possibly empty) subset of {extension, restriction}</xs:
documentation>
  </xs:annotation>
  <xs:union>
   <xs:simpleType>
    <xs:restriction base="xs:token">
     <xs:enumeration value="#all"/>
    </xs:restriction>
   </xs:simpleType>
   <xs:simpleType>
    <xs:list itemType="xs:reducedDerivationControl"/>
   </xs:simpleType>
  </xs:union>
 </xs:simpleType>

 <xs:element name="schema" id="schema">
  <xs:annotation>
```

```
    <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
schema"/>
  </xs:annotation>
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="xs:openAttrs">
     <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element ref="xs:include"/>
       <xs:element ref="xs:import"/>
       <xs:element ref="xs:redefine"/>
       <xs:element ref="xs:annotation"/>
      </xs:choice>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
       <xs:group ref="xs:schemaTop"/>
       <xs:element ref="xs:annotation" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
     </xs:sequence>
     <xs:attribute name="targetNamespace" type="xs:anyURI"/>
     <xs:attribute name="version" type="xs:token"/>
     <xs:attribute name="finalDefault" type="xs:derivationSet"
use="optional" default=""/>
     <xs:attribute name="blockDefault" type="xs:blockSet" use="optional"
default=""/>
     <xs:attribute name="attributeFormDefault" type="xs:formChoice"
use="optional" default="unqualified"/>
     <xs:attribute name="elementFormDefault" type="xs:formChoice"
use="optional" default="unqualified"/>
     <xs:attribute name="id" type="xs:ID"/>
     <xs:attribute ref="xml:lang"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:key name="element">
   <xs:selector xpath="xs:element"/>
   <xs:field xpath="@name"/>
  </xs:key>

  <xs:key name="attribute">
   <xs:selector xpath="xs:attribute"/>
   <xs:field xpath="@name"/>
  </xs:key>

  <xs:key name="type">
   <xs:selector xpath="xs:complexType|xs:simpleType"/>
   <xs:field xpath="@name"/>
  </xs:key>

  <xs:key name="group">
   <xs:selector xpath="xs:group"/>
   <xs:field xpath="@name"/>
  </xs:key>

  <xs:key name="attributeGroup">
   <xs:selector xpath="xs:attributeGroup"/>
   <xs:field xpath="@name"/>
  </xs:key>
```

```
  <xs:key name="notation">
   <xs:selector xpath="xs:notation"/>
   <xs:field xpath="@name"/>
  </xs:key>

  <xs:key name="identityConstraint">
   <xs:selector xpath=".//xs:key|.//xs:unique|.//xs:keyref"/>
   <xs:field xpath="@name"/>
  </xs:key>

 </xs:element>

 <xs:simpleType name="allNNI">
  <xs:annotation><xs:documentation>
   for maxOccurs</xs:documentation></xs:annotation>
  <xs:union memberTypes="xs:nonNegativeInteger">
   <xs:simpleType>
    <xs:restriction base="xs:NMTOKEN">
     <xs:enumeration value="unbounded"/>
    </xs:restriction>
   </xs:simpleType>
  </xs:union>
 </xs:simpleType>

 <xs:attributeGroup name="occurs">
  <xs:annotation><xs:documentation>
   for all particles</xs:documentation></xs:annotation>
  <xs:attribute name="minOccurs" type="xs:nonNegativeInteger"
use="optional" default="1"/>
  <xs:attribute name="maxOccurs" type="xs:allNNI" use="optional"
default="1"/>
 </xs:attributeGroup>

 <xs:attributeGroup name="defRef">
  <xs:annotation><xs:documentation>
   for element, group and attributeGroup,
   which both define and reference</xs:documentation></xs:annotation>
  <xs:attribute name="name" type="xs:NCName"/>
  <xs:attribute name="ref" type="xs:QName"/>
 </xs:attributeGroup>

 <xs:group name="typeDefParticle">
  <xs:annotation>
    <xs:documentation>
   'complexType' uses this</xs:documentation></xs:annotation>
  <xs:choice>
   <xs:element name="group" type="xs:groupRef"/>
   <xs:element ref="xs:all"/>
   <xs:element ref="xs:choice"/>
   <xs:element ref="xs:sequence"/>
  </xs:choice>
 </xs:group>



 <xs:group name="nestedParticle">
  <xs:choice>
   <xs:element name="element" type="xs:localElement"/>
   <xs:element name="group" type="xs:groupRef"/>
   <xs:element ref="xs:choice"/>
```

```
    <xs:element ref="xs:sequence"/>
    <xs:element ref="xs:any"/>
   </xs:choice>
  </xs:group>

  <xs:group name="particle">
   <xs:choice>
    <xs:element name="element" type="xs:localElement"/>
    <xs:element name="group" type="xs:groupRef"/>
    <xs:element ref="xs:all"/>
    <xs:element ref="xs:choice"/>
    <xs:element ref="xs:sequence"/>
    <xs:element ref="xs:any"/>
   </xs:choice>
  </xs:group>

  <xs:complexType name="attribute">
   <xs:complexContent>
    <xs:extension base="xs:annotated">
     <xs:sequence>
      <xs:element name="simpleType" minOccurs="0" type="xs:
localSimpleType"/>
     </xs:sequence>
     <xs:attributeGroup ref="xs:defRef"/>
     <xs:attribute name="type" type="xs:QName"/>
     <xs:attribute name="use" use="optional" default="optional">
      <xs:simpleType>
       <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="prohibited"/>
        <xs:enumeration value="optional"/>
        <xs:enumeration value="required"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="default" type="xs:string"/>
     <xs:attribute name="fixed" type="xs:string"/>
     <xs:attribute name="form" type="xs:formChoice"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="topLevelAttribute">
   <xs:complexContent>
    <xs:restriction base="xs:attribute">
     <xs:sequence>
      <xs:element ref="xs:annotation" minOccurs="0"/>
      <xs:element name="simpleType" minOccurs="0" type="xs:
localSimpleType"/>
     </xs:sequence>
     <xs:attribute name="ref" use="prohibited"/>
     <xs:attribute name="form" use="prohibited"/>
     <xs:attribute name="use" use="prohibited"/>
     <xs:attribute name="name" use="required" type="xs:NCName"/>
    </xs:restriction>
   </xs:complexContent>
  </xs:complexType>

  <xs:group name="attrDecls">
   <xs:sequence>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
```

```
    <xs:element name="attribute" type="xs:attribute"/>
    <xs:element name="attributeGroup" type="xs:attributeGroupRef"/>
   </xs:choice>
   <xs:element ref="xs:anyAttribute" minOccurs="0"/>
  </xs:sequence>
 </xs:group>

 <xs:element name="anyAttribute" type="xs:wildcard" id="anyAttribute">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
anyAttribute"/>
  </xs:annotation>
 </xs:element>

 <xs:group name="complexTypeModel">
  <xs:choice>
      <xs:element ref="xs:simpleContent"/>
      <xs:element ref="xs:complexContent"/>
      <xs:sequence>
       <xs:annotation>
        <xs:documentation>
   This branch is short for
   &lt;complexContent>
   &lt;restriction base="xs:anyType">
   ...
   &lt;/restriction>
   &lt;/complexContent></xs:documentation>
       </xs:annotation>
       <xs:group ref="xs:typeDefParticle" minOccurs="0"/>
       <xs:group ref="xs:attrDecls"/>
      </xs:sequence>
  </xs:choice>
 </xs:group>

 <xs:complexType name="complexType" abstract="true">
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:group ref="xs:complexTypeModel"/>
    <xs:attribute name="name" type="xs:NCName">
     <xs:annotation>
      <xs:documentation>
      Will be restricted to required or forbidden</xs:documentation>
     </xs:annotation>
    </xs:attribute>
    <xs:attribute name="mixed" type="xs:boolean" use="optional"
default="false">
     <xs:annotation>
      <xs:documentation>
      Not allowed if simpleContent child is chosen.
      May be overriden by setting on complexContent child.</xs:
documentation>
    </xs:annotation>
    </xs:attribute>
    <xs:attribute name="abstract" type="xs:boolean" use="optional"
default="false"/>
    <xs:attribute name="final" type="xs:derivationSet"/>
    <xs:attribute name="block" type="xs:derivationSet"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
```

```
<xs:complexType name="topLevelComplexType">
 <xs:complexContent>
  <xs:restriction base="xs:complexType">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:group ref="xs:complexTypeModel"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:NCName" use="required"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="localComplexType">
 <xs:complexContent>
  <xs:restriction base="xs:complexType">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:group ref="xs:complexTypeModel"/>
   </xs:sequence>
   <xs:attribute name="name" use="prohibited"/>
   <xs:attribute name="abstract" use="prohibited"/>
   <xs:attribute name="final" use="prohibited"/>
   <xs:attribute name="block" use="prohibited"/>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="restrictionType">
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:sequence>
    <xs:choice>
     <xs:group ref="xs:typeDefParticle" minOccurs="0"/>
     <xs:group ref="xs:simpleRestrictionModel" minOccurs="0"/>
    </xs:choice>
    <xs:group ref="xs:attrDecls"/>
   </xs:sequence>
   <xs:attribute name="base" type="xs:QName" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="complexRestrictionType">
 <xs:complexContent>
  <xs:restriction base="xs:restrictionType">
   <xs:sequence>
    <xs:element ref="xs:annotation" minOccurs="0"/>
    <xs:group ref="xs:typeDefParticle" minOccurs="0"/>
    <xs:group ref="xs:attrDecls"/>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="extensionType">
 <xs:complexContent>
  <xs:extension base="xs:annotated">
   <xs:sequence>
    <xs:group ref="xs:typeDefParticle" minOccurs="0"/>
```

```
      <xs:group ref="xs:attrDecls"/>
     </xs:sequence>
     <xs:attribute name="base" type="xs:QName" use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:element name="complexContent" id="complexContent">
   <xs:annotation>
    <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
complexContent"/>
   </xs:annotation>
   <xs:complexType>
    <xs:complexContent>
     <xs:extension base="xs:annotated">
      <xs:choice>
       <xs:element name="restriction" type="xs:complexRestrictionType"/>
       <xs:element name="extension" type="xs:extensionType"/>
      </xs:choice>
      <xs:attribute name="mixed" type="xs:boolean">
       <xs:annotation>
        <xs:documentation>
        Overrides any setting on complexType parent.</xs:documentation>
       </xs:annotation>
     </xs:attribute>
     </xs:extension>
    </xs:complexContent>
   </xs:complexType>
  </xs:element>

  <xs:complexType name="simpleRestrictionType">
   <xs:complexContent>
    <xs:restriction base="xs:restrictionType">
     <xs:sequence>
      <xs:element ref="xs:annotation" minOccurs="0"/>
      <xs:group ref="xs:simpleRestrictionModel" minOccurs="0"/>
      <xs:group ref="xs:attrDecls"/>
     </xs:sequence>
    </xs:restriction>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="simpleExtensionType">
   <xs:complexContent>
    <xs:restriction base="xs:extensionType">
     <xs:sequence>
      <xs:annotation>
       <xs:documentation>
       No typeDefParticle group reference</xs:documentation>
      </xs:annotation>
      <xs:element ref="xs:annotation" minOccurs="0"/>
      <xs:group ref="xs:attrDecls"/>
     </xs:sequence>
    </xs:restriction>
   </xs:complexContent>
  </xs:complexType>

  <xs:element name="simpleContent" id="simpleContent">
   <xs:annotation>
    <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
```

```
simpleContent"/>
  </xs:annotation>
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="xs:annotated">
     <xs:choice>
      <xs:element name="restriction" type="xs:simpleRestrictionType"/>
      <xs:element name="extension" type="xs:simpleExtensionType"/>
     </xs:choice>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>

 <xs:element name="complexType" type="xs:topLevelComplexType"
id="complexType">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
complexType"/>
  </xs:annotation>
 </xs:element>


  <xs:simpleType name="blockSet">
   <xs:annotation>
    <xs:documentation>
    A utility type, not for public use</xs:documentation>
    <xs:documentation>
    #all or (possibly empty) subset of {substitution, extension,
    restriction}</xs:documentation>
   </xs:annotation>
   <xs:union>
    <xs:simpleType>
     <xs:restriction base="xs:token">
      <xs:enumeration value="#all"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
     <xs:list>
      <xs:simpleType>
       <xs:restriction base="xs:derivationControl">
        <xs:enumeration value="extension"/>
        <xs:enumeration value="restriction"/>
        <xs:enumeration value="substitution"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:list>
    </xs:simpleType>
   </xs:union>
  </xs:simpleType>

 <xs:complexType name="element" abstract="true">
  <xs:annotation>
   <xs:documentation>
   The element element can be used either
   at the top level to define an element-type binding globally,
   or within a content model to either reference a globally-defined
   element or type or declare an element-type binding locally.
   The ref form is not allowed at the top level.</xs:documentation>
  </xs:annotation>
```

```
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:sequence>
     <xs:choice minOccurs="0">
      <xs:element name="simpleType" type="xs:localSimpleType"/>
      <xs:element name="complexType" type="xs:localComplexType"/>
     </xs:choice>
     <xs:group ref="xs:identityConstraint" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attributeGroup ref="xs:defRef"/>
    <xs:attribute name="type" type="xs:QName"/>
    <xs:attribute name="substitutionGroup" type="xs:QName"/>
    <xs:attributeGroup ref="xs:occurs"/>
    <xs:attribute name="default" type="xs:string"/>
    <xs:attribute name="fixed" type="xs:string"/>
    <xs:attribute name="nillable" type="xs:boolean" use="optional"
default="false"/>
    <xs:attribute name="abstract" type="xs:boolean" use="optional"
default="false"/>
    <xs:attribute name="final" type="xs:derivationSet"/>
    <xs:attribute name="block" type="xs:blockSet"/>
    <xs:attribute name="form" type="xs:formChoice"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="topLevelElement">
  <xs:complexContent>
   <xs:restriction base="xs:element">
    <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
     <xs:choice minOccurs="0">
      <xs:element name="simpleType" type="xs:localSimpleType"/>
      <xs:element name="complexType" type="xs:localComplexType"/>
     </xs:choice>
     <xs:group ref="xs:identityConstraint" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="ref" use="prohibited"/>
    <xs:attribute name="form" use="prohibited"/>
    <xs:attribute name="minOccurs" use="prohibited"/>
    <xs:attribute name="maxOccurs" use="prohibited"/>
    <xs:attribute name="name" use="required" type="xs:NCName"/>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="localElement">
  <xs:complexContent>
   <xs:restriction base="xs:element">
    <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
     <xs:choice minOccurs="0">
      <xs:element name="simpleType" type="xs:localSimpleType"/>
      <xs:element name="complexType" type="xs:localComplexType"/>
     </xs:choice>
     <xs:group ref="xs:identityConstraint" minOccurs="0"
maxOccurs="unbounded"/>
```

```
    </xs:sequence>
    <xs:attribute name="substitutionGroup" use="prohibited"/>
    <xs:attribute name="final" use="prohibited"/>
    <xs:attribute name="abstract" use="prohibited"/>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>

 <xs:element name="element" type="xs:topLevelElement" id="element">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
element"/>
  </xs:annotation>
 </xs:element>

 <xs:complexType name="group" abstract="true">
  <xs:annotation>
   <xs:documentation>
   group type for explicit groups, named top-level groups and
   group references</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:group ref="xs:particle" minOccurs="0" maxOccurs="unbounded"/>
    <xs:attributeGroup ref="xs:defRef"/>
    <xs:attributeGroup ref="xs:occurs"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="realGroup">
  <xs:complexContent>
   <xs:restriction base="xs:group">
    <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
     <xs:choice minOccurs="0" maxOccurs="1">
      <xs:element ref="xs:all"/>
      <xs:element ref="xs:choice"/>
      <xs:element ref="xs:sequence"/>
     </xs:choice>
    </xs:sequence>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="namedGroup">
  <xs:annotation>
   <xs:documentation>Should derive this from realGroup, but too
complicated
                     for now</xs:documentation>
  </xs:annotation>
  <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
     <xs:choice minOccurs="1" maxOccurs="1">
      <xs:element name="all">
       <xs:complexType>
        <xs:complexContent>
         <xs:restriction base="xs:all">
          <xs:group ref="xs:allModel"/>
          <xs:attribute name="minOccurs" use="prohibited"/>
```

```
            <xs:attribute name="maxOccurs" use="prohibited"/>
           </xs:restriction>
          </xs:complexContent>
         </xs:complexType>
        </xs:element>
        <xs:element name="choice" type="xs:simpleExplicitGroup"/>
        <xs:element name="sequence" type="xs:simpleExplicitGroup"/>
       </xs:choice>
     </xs:sequence>
     <xs:attribute name="name" use="required" type="xs:NCName"/>
     <xs:attribute name="ref" use="prohibited"/>
     <xs:attribute name="minOccurs" use="prohibited"/>
     <xs:attribute name="maxOccurs" use="prohibited"/>
 </xs:complexType>

 <xs:complexType name="groupRef">
  <xs:complexContent>
   <xs:restriction base="xs:realGroup">
    <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="ref" use="required" type="xs:QName"/>
    <xs:attribute name="name" use="prohibited"/>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="explicitGroup">
  <xs:annotation>
   <xs:documentation>
   group type for the three kinds of group</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
   <xs:restriction base="xs:group">
    <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
     <xs:group ref="xs:nestedParticle" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:NCName" use="prohibited"/>
    <xs:attribute name="ref" type="xs:QName" use="prohibited"/>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="simpleExplicitGroup">
  <xs:complexContent>
   <xs:restriction base="xs:explicitGroup">
    <xs:sequence>
     <xs:element ref="xs:annotation" minOccurs="0"/>
     <xs:group ref="xs:nestedParticle" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="minOccurs" use="prohibited"/>
    <xs:attribute name="maxOccurs" use="prohibited"/>
   </xs:restriction>
  </xs:complexContent>
 </xs:complexType>

 <xs:group name="allModel">
```

```
    <xs:sequence>
        <xs:element ref="xs:annotation" minOccurs="0"/>
        <xs:element name="element" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
          <xs:annotation>
           <xs:documentation>restricted max/min</xs:documentation>
          </xs:annotation>
          <xs:complexContent>
           <xs:restriction base="xs:localElement">
            <xs:sequence>
             <xs:element ref="xs:annotation" minOccurs="0"/>
             <xs:choice minOccurs="0">
              <xs:element name="simpleType" type="xs:localSimpleType"/>
              <xs:element name="complexType" type="xs:localComplexType"/>
             </xs:choice>
             <xs:group ref="xs:identityConstraint" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="minOccurs" use="optional" default="1">
             <xs:simpleType>
              <xs:restriction base="xs:nonNegativeInteger">
               <xs:enumeration value="0"/>
               <xs:enumeration value="1"/>
              </xs:restriction>
             </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="maxOccurs" use="optional" default="1">
             <xs:simpleType>
              <xs:restriction base="xs:allNNI">
               <xs:enumeration value="0"/>
               <xs:enumeration value="1"/>
              </xs:restriction>
             </xs:simpleType>
            </xs:attribute>
           </xs:restriction>
          </xs:complexContent>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
  </xs:group>

   <xs:complexType name="all">
    <xs:annotation>
     <xs:documentation>
    Only elements allowed inside</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
     <xs:restriction base="xs:explicitGroup">
      <xs:group ref="xs:allModel"/>
      <xs:attribute name="minOccurs" use="optional" default="1">
       <xs:simpleType>
        <xs:restriction base="xs:nonNegativeInteger">
         <xs:enumeration value="0"/>
         <xs:enumeration value="1"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="maxOccurs" use="optional" default="1">
       <xs:simpleType>
        <xs:restriction base="xs:allNNI">
```

```
        <xs:enumeration value="1"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
    </xs:restriction>
   </xs:complexContent>
  </xs:complexType>

 <xs:element name="all" id="all" type="xs:all">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
all"/>
  </xs:annotation>
 </xs:element>

 <xs:element name="choice" type="xs:explicitGroup" id="choice">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
choice"/>
  </xs:annotation>
 </xs:element>

 <xs:element name="sequence" type="xs:explicitGroup" id="sequence">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
sequence"/>
  </xs:annotation>
 </xs:element>

 <xs:element name="group" type="xs:namedGroup" id="group">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
group"/>
  </xs:annotation>
 </xs:element>

 <xs:complexType name="wildcard">
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:attribute name="namespace" type="xs:namespaceList" use="optional"
default="##any"/>
    <xs:attribute name="processContents" use="optional" default="strict">
     <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
       <xs:enumeration value="skip"/>
       <xs:enumeration value="lax"/>
       <xs:enumeration value="strict"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:attribute>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:element name="any" id="any">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
any"/>
  </xs:annotation>
  <xs:complexType>
```

```
    <xs:complexContent>
     <xs:extension base="xs:wildcard">
      <xs:attributeGroup ref="xs:occurs"/>
     </xs:extension>
    </xs:complexContent>
   </xs:complexType>
  </xs:element>

   <xs:annotation>
    <xs:documentation>
    simple type for the value of the 'namespace' attr of
    'any' and 'anyAttribute'</xs:documentation>
   </xs:annotation>
   <xs:annotation>
    <xs:documentation>
    Value is
                ##any      - - any non-conflicting WFXML/attribute at all

                ##other    - - any non-conflicting WFXML/attribute from
                                namespace other than targetNS

                ##local    - - any unqualified non-conflicting WFXML/
attribute

                one or     - - any non-conflicting WFXML/attribute from
                more URI        the listed namespaces
                references
                (space separated)

     ##targetNamespace or ##local may appear in the above list, to
         refer to the targetNamespace of the enclosing
         schema or an absent targetNamespace respectively</xs:
documentation>
   </xs:annotation>

  <xs:simpleType name="namespaceList">
   <xs:annotation>
    <xs:documentation>
    A utility type, not for public use</xs:documentation>
   </xs:annotation>
   <xs:union>
    <xs:simpleType>
     <xs:restriction base="xs:token">
      <xs:enumeration value="##any"/>
      <xs:enumeration value="##other"/>
     </xs:restriction>
    </xs:simpleType>
    <xs:simpleType>
     <xs:list>
      <xs:simpleType>
       <xs:union memberTypes="xs:anyURI">
        <xs:simpleType>
         <xs:restriction base="xs:token">
          <xs:enumeration value="##targetNamespace"/>
          <xs:enumeration value="##local"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:union>
      </xs:simpleType>
     </xs:list>
```

```
    </xs:simpleType>
   </xs:union>
  </xs:simpleType>

  <xs:element name="attribute" type="xs:topLevelAttribute" id="attribute">
   <xs:annotation>
    <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
attribute"/>
   </xs:annotation>
  </xs:element>

  <xs:complexType name="attributeGroup" abstract="true">
   <xs:complexContent>
    <xs:extension base="xs:annotated">
     <xs:group ref="xs:attrDecls"/>
     <xs:attributeGroup ref="xs:defRef"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="namedAttributeGroup">
   <xs:complexContent>
    <xs:restriction base="xs:attributeGroup">
     <xs:sequence>
      <xs:element ref="xs:annotation" minOccurs="0"/>
      <xs:group ref="xs:attrDecls"/>
     </xs:sequence>
     <xs:attribute name="name" use="required" type="xs:NCName"/>
     <xs:attribute name="ref" use="prohibited"/>
    </xs:restriction>
   </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="attributeGroupRef">
   <xs:complexContent>
    <xs:restriction base="xs:attributeGroup">
     <xs:sequence>
      <xs:element ref="xs:annotation" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="ref" use="required" type="xs:QName"/>
     <xs:attribute name="name" use="prohibited"/>
    </xs:restriction>
   </xs:complexContent>
  </xs:complexType>

  <xs:element name="attributeGroup" type="xs:namedAttributeGroup"
id="attributeGroup">
   <xs:annotation>
    <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
attributeGroup"/>
   </xs:annotation>
  </xs:element>

  <xs:element name="include" id="include">
   <xs:annotation>
    <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
include"/>
   </xs:annotation>
   <xs:complexType>
    <xs:complexContent>
```

```
    <xs:extension base="xs:annotated">
     <xs:attribute name="schemaLocation" type="xs:anyURI" use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>

 <xs:element name="redefine" id="redefine">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
redefine"/>
  </xs:annotation>
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="xs:openAttrs">
     <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="xs:annotation"/>
      <xs:group ref="xs:redefinable"/>
     </xs:choice>
     <xs:attribute name="schemaLocation" type="xs:anyURI" use="required"/>
     <xs:attribute name="id" type="xs:ID"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>

 <xs:element name="import" id="import">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
import"/>
  </xs:annotation>
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="xs:annotated">
     <xs:attribute name="namespace" type="xs:anyURI"/>
     <xs:attribute name="schemaLocation" type="xs:anyURI"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>

 <xs:element name="selector" id="selector">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
selector"/>
  </xs:annotation>
  <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:annotated">
     <xs:attribute name="xpath" use="required">
      <xs:simpleType>
       <xs:annotation>
        <xs:documentation>A subset of XPath expressions for use
in selectors</xs:documentation>
        <xs:documentation>A utility type, not for public
use</xs:documentation>
       </xs:annotation>
       <xs:restriction base="xs:token">
        <xs:annotation>
         <xs:documentation>The following pattern is intended to allow
```

```
XPath
                            expressions per the following EBNF:
          Selector    ::=    Path ( '|' Path )*
          Path    ::=    ('.//')? Step ( '/' Step )*
          Step    ::=    '.' | NameTest
          NameTest    ::=    QName | '*' | NCName ':' '*'
                            child:: is also allowed
         </xs:documentation>
        </xs:annotation>
        <xs:pattern value="(\.//)?(((child::)?((\i\c*:)?(\i\c*|\*)))|\.)(/
(((child::)?((\i\c*:)?(\i\c*|\*)))|\.))*(\|(\.//)?(((child::)?((\i\c*:)?
(\i\c*|\*)))|\.)(/(((child::)?((\i\c*:)?(\i\c*|\*)))|\.))*)*">
        </xs:pattern>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
 </xs:element>

 <xs:element name="field" id="field">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
field"/>
  </xs:annotation>
  <xs:complexType>
  <xs:complexContent>
   <xs:extension base="xs:annotated">
     <xs:attribute name="xpath" use="required">
      <xs:simpleType>
       <xs:annotation>
        <xs:documentation>A subset of XPath expressions for use
in fields</xs:documentation>
        <xs:documentation>A utility type, not for public
use</xs:documentation>
       </xs:annotation>
       <xs:restriction base="xs:token">
        <xs:annotation>
         <xs:documentation>The following pattern is intended to allow
XPath
                            expressions per the same EBNF as for selector,
                            with the following change:
          Path    ::=    ('.//')? ( Step '/' )* ( Step | '@' NameTest )
         </xs:documentation>
        </xs:annotation>
        <xs:pattern value="(\.//)?((((child::)?((\i\c*:)?(\i\c*|\*)))|
\.)/)*((((child::)?((\i\c*:)?(\i\c*|\*)))|\.)|((attribute::|@)((\i\c*:)?
(\i\c*|\*))))(\|(\.//)?((((child::)?((\i\c*:)?(\i\c*|\*)))|\.)/)*
((((child::)?((\i\c*:)?(\i\c*|\*)))|\.)|((attribute::|@)((\i\c*:)?(\i\c*|
\*)))))*">
        </xs:pattern>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
 </xs:element>
```

```
 <xs:complexType name="keybase">
  <xs:complexContent>
   <xs:extension base="xs:annotated">
    <xs:sequence>
     <xs:element ref="xs:selector"/>
     <xs:element ref="xs:field" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:NCName" use="required"/>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:group name="identityConstraint">
  <xs:annotation>
   <xs:documentation>The three kinds of identity constraints, all with
                     type of or derived from 'keybase'.
   </xs:documentation>
  </xs:annotation>
  <xs:choice>
   <xs:element ref="xs:unique"/>
   <xs:element ref="xs:key"/>
   <xs:element ref="xs:keyref"/>
  </xs:choice>
 </xs:group>

 <xs:element name="unique" type="xs:keybase" id="unique">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
unique"/>
  </xs:annotation>
 </xs:element>
 <xs:element name="key" type="xs:keybase" id="key">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
key"/>
  </xs:annotation>
 </xs:element>
 <xs:element name="keyref" id="keyref">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
keyref"/>
  </xs:annotation>
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="xs:keybase">
     <xs:attribute name="refer" type="xs:QName" use="required"/>
    </xs:extension>
   </xs:complexContent>
  </xs:complexType>
 </xs:element>

 <xs:element name="notation" id="notation">
  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
notation"/>
  </xs:annotation>
  <xs:complexType>
   <xs:complexContent>
    <xs:extension base="xs:annotated">
     <xs:attribute name="name" type="xs:NCName" use="required"/>
```

```
      <xs:attribute name="public" type="xs:public" use="required"/>
      <xs:attribute name="system" type="xs:anyURI"/>
     </xs:extension>
    </xs:complexContent>
   </xs:complexType>
  </xs:element>

  <xs:simpleType name="public">
   <xs:annotation>
    <xs:documentation>
    A utility type, not for public use</xs:documentation>
    <xs:documentation>
    A public identifier, per ISO 8879</xs:documentation>
   </xs:annotation>
   <xs:restriction base="xs:token"/>
  </xs:simpleType>

  <xs:element name="appinfo" id="appinfo">
    <xs:annotation>
      <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
appinfo"/>
    </xs:annotation>
    <xs:complexType mixed="true">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:any processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="source" type="xs:anyURI"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="documentation" id="documentation">
    <xs:annotation>
      <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
documentation"/>
    </xs:annotation>
    <xs:complexType mixed="true">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:any processContents="lax"/>
      </xs:sequence>
      <xs:attribute name="source" type="xs:anyURI"/>
    <xs:attribute ref="xml:lang"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="annotation" id="annotation">
    <xs:annotation>
      <xs:documentation source="http://www.w3.org/TR/xmlschema-1/#element-
annotation"/>
    </xs:annotation>
    <xs:complexType>
     <xs:complexContent>
      <xs:extension base="xs:openAttrs">
       <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="xs:appinfo"/>
        <xs:element ref="xs:documentation"/>
       </xs:choice>
       <xs:attribute name="id" type="xs:ID"/>
      </xs:extension>
     </xs:complexContent>
    </xs:complexType>
```

```
 </xs:element>

 <xs:annotation>
  <xs:documentation>
   notations for use within XML Schema schemas</xs:documentation>
 </xs:annotation>

 <xs:notation name="XMLSchemaStructures" public="structures"
system="http://www.w3.org/2000/08/XMLSchema.xsd"/>
 <xs:notation name="XML" public="REC-xml-19980210" system="http://www.w3.
org/TR/1998/REC-xml-19980210"/>

 <xs:complexType name="anyType" mixed="true">
  <xs:annotation>
   <xs:documentation>
   Not the real urType, but as close an approximation as we can
   get in the XML representation</xs:documentation>
  </xs:annotation>
  <xs:sequence>
   <xs:any minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:anyAttribute/>
 </xs:complexType>
</xs:schema>
```

NOTE: And that is the end of the schema for *XML Schema: Structures*.

B References (normative)

XML 1.0 (Second Edition)
Extensible Markup Language (XML) 1.0, Second Edition, Tim Bray et al., eds., W3C, 6 October 2000. See http://www.w3.org/TR/2000/REC-xml-20001006

XML-Infoset
XML Information Set, John Cowan and Richard Tobin, eds., W3C, 16 March 2001. See http://www.w3.org/TR/2001/WD-xml-infoset-20010316/

XML-Namespaces
Namespaces in XML, Tim Bray et al., eds., W3C, 14 January 1999. See http://www.w3.org/TR/1999/REC-xml-names-19990114/

XML Schema Requirements
XML Schema Requirements , Ashok Malhotra and Murray Maloney, eds., W3C, 15 February 1999. See http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215

XML Schemas: Datatypes
XML Schema Part 2: Datatypes, Paul V. Biron and Ashok Malhotra, eds., W3C, 2 May 2001. See http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/datatypes.html

XPath
XML Path Language, James Clark and Steve DeRose, eds., W3C, 16 November 1999. See http://www.w3.org/TR/1999/REC-xpath-19991116

XPointer
XML Pointer Language (XPointer), Eve Maler and Steve DeRose, eds., W3C, 8 January 2001. See http://www.w3.org/TR/2001/WD-xptr-20010108/

C Outcome Tabulations (normative)

To facilitate consistent reporting of schema errors and ·validation· failures, this section tabulates and provides unique names for all the constraints listed in this document. Wherever such constraints have numbered parts, reports should use the name given below plus the part number, separated by a period ('.'). Thus for example `cos-ct-extends.1.2` should be used to report a violation of the clause 1.2 of

Derivation Valid (Extension) (§3.4.6).

C.1 Validation Rules

cvc-assess-attr
Schema-Validity Assessment (Attribute)
cvc-assess-elt
Schema-Validity Assessment (Element)
cvc-attribute
Attribute Locally Valid
cvc-au
Attribute Locally Valid (Use)
cvc-complex-type
Element Locally Valid (Complex Type)
cvc-datatype-valid
Datatype Valid
cvc-elt
Element Locally Valid (Element)
cvc-enumeration-valid
enumeration valid
cvc-facet-valid
Facet Valid
cvc-fractionDigits-valid
fractionDigits Valid
cvc-id
Validation Root Valid (ID/IDREF)
cvc-identity-constraint
Identity-constraint Satisfied
cvc-length-valid
Length Valid
cvc-maxExclusive-valid
maxExclusive Valid
cvc-maxInclusive-valid
maxInclusive Valid
cvc-maxLength-valid
maxLength Valid
cvc-minExclusive-valid
minExclusive Valid
cvc-minInclusive-valid
minInclusive Valid
cvc-minLength-valid
minLength Valid
cvc-model-group
Element Sequence Valid
cvc-particle
Element Sequence Locally Valid (Particle)
cvc-pattern-valid
pattern valid
cvc-resolve-instance
QName resolution (Instance)
cvc-simple-type
String Valid
cvc-totalDigits-valid
totalDigits Valid
cvc-type
Element Locally Valid (Type)

cvc-wildcard
Item Valid (Wildcard)
cvc-wildcard-namespace
Wildcard allows Namespace Name

C.2 Contributions to the post-schema-validation infoset

attribute information item properties
[attribute declaration] (Attribute Declaration)
[member type definition] (Attribute Validated by Type)
[member type definition anonymous] (Attribute Validated by Type)
[member type definition name] (Attribute Validated by Type)
[member type definition namespace] (Attribute Validated by Type)
[schema default] (Attribute Validated by Type)
[schema error code] (Validation Failure (Attribute))
[schema normalized value] (Attribute Validated by Type)
[schema specified] (Assessment Outcome (Attribute))
[type definition] (Attribute Validated by Type)
[type definition anonymous] (Attribute Validated by Type)
[type definition name] (Attribute Validated by Type)
[type definition namespace] (Attribute Validated by Type)
[type definition type] (Attribute Validated by Type)
[validation attempted] (Assessment Outcome (Attribute))
[validation context] (Assessment Outcome (Attribute))
[validity] (Assessment Outcome (Attribute))
element information item properties
[element declaration] (Element Declaration)
[ID/IDREF table] (ID/IDREF Table)
[identity-constraint table] (Identity-constraint Table)
[member type definition] (Element Validated by Type)
[member type definition anonymous] (Element Validated by Type)
[member type definition name] (Element Validated by Type)
[member type definition namespace] (Element Validated by Type)
[nil] (Element Declaration)
[notation] (Validated with Notation)
[notation public] (Validated with Notation)
[notation system] (Validated with Notation)
[schema default] (Element Validated by Type)
[schema error code] (Validation Failure (Element))
[schema information] (Schema Information)
[schema normalized value] (Element Validated by Type)
[schema specified] (Element Default Value)
[type definition] (Element Validated by Type)
[type definition anonymous] (Element Validated by Type)
[type definition name] (Element Validated by Type)
[type definition namespace] (Element Validated by Type)
[type definition type] (Element Validated by Type)
[validation attempted] (Assessment Outcome (Element))
[validation context] (Assessment Outcome (Element))
[validity] (Assessment Outcome (Element))
ID/IDREF binding information item properties
[binding] (ID/IDREF Table)
[id] (ID/IDREF Table)
Identity-constraint Binding information item properties

[definition] (Identity-constraint Table)
[node table] (Identity-constraint Table)

namespace schema information information item properties

[schema components] (Schema Information)
[schema documents] (Schema Information)
[schema namespace] (Schema Information)

schema document information item properties

[document location] (Schema Information)
[document] (Schema Information)

C.3 Schema Representation Constraints

schema_reference
Schema Document Location Strategy

src-annotation
Annotation Definition Representation OK

src-attribute
Attribute Declaration Representation OK

src-attribute_group
Attribute Group Definition Representation OK

src-ct
Complex Type Definition Representation OK

src-element
Element Declaration Representation OK

src-expredef
Individual Component Redefinition

src-identity-constraint
Identity-constraint Definition Representation OK

src-import
Import Constraints and Semantics

src-include
Inclusion Constraints and Semantics

src-list-itemType-or-simpleType
itemType attribute or simpleType child

src-model_group
Model Group Representation OK

src-model_group_defn
Model Group Definition Representation OK

src-multiple-enumerations
Multiple enumerations

src-multiple-patterns
Multiple patterns

src-notation
Notation Definition Representation OK

src-qname
QName Interpretation

src-redefine
Redefinition Constraints and Semantics

src-resolve
QName resolution (Schema Document)

src-restriction-base-or-simpleType
base attribute or simpleType child

src-simple-type
Simple Type Definition Representation OK

src-single-facet-value
Single Facet Value

src-union-memberTypes-or-simpleTypes
memberTypes attribute or simpleType children
src-wildcard
Wildcard Representation OK
st-restrict-facets
Simple Type Restriction (Facets)

C.4 Schema Component Constraints

ag-props-correct
Attribute Group Definition Properties Correct
an-props-correct
Annotation Correct
a-props-correct
Attribute Declaration Properties Correct
au-props-correct
Attribute Use Correct
c-fields-xpaths
Fields Value OK
cos-all-limited
All Group Limited
cos-applicable-facets
applicable facets
cos-aw-intersect
Attribute Wildcard Intersection
cos-aw-union
Attribute Wildcard Union
cos-choice-range
Effective Total Range (choice)
cos-ct-derived-ok
Type Derivation OK (Complex)
cos-ct-extends
Derivation Valid (Extension)
cos-element-consistent
Element Declarations Consistent
cos-equiv-class
Substitution Group
cos-equiv-derived-ok-rec
Substitution Group OK (Transitive)
cos-group-emptiable
Particle Emptiable
cos-list-of-atomic
list of atomic
cos-no-circular-unions
no circular unions
cos-nonambig
Unique Particle Attribution
cos-ns-subset
Wildcard Subset
cos-particle-extend
Particle Valid (Extension)
cos-particle-restrict
Particle Valid (Restriction)
cos-seq-range
Effective Total Range (all and sequence)
cos-st-derived-ok

Type Derivation OK (Simple)

cos-st-restricts
Derivation Valid (Restriction, Simple)

cos-valid-default
Element Default Valid (Immediate)

c-props-correct
Identity-constraint Definition Properties Correct

c-selector-xpath
Selector Value OK

ct-props-correct
Complex Type Definition Properties Correct

derivation-ok-restriction
Derivation Valid (Restriction, Complex)

enumeration-required-notation
enumeration facet value required for NOTATION

enumeration-valid-restriction
enumeration valid restriction

e-props-correct
Element Declaration Properties Correct

fractionDigits-totalDigits
fractionDigits less than or equal to totalDigits

length-minLength-maxLength
length and minLength or maxLength

length-valid-restriction
length valid restriction

maxExclusive-valid-restriction
maxExclusive valid restriction

maxInclusive-maxExclusive
maxInclusive and maxExclusive

maxInclusive-valid-restriction
maxInclusive valid restriction

maxLength-valid-restriction
maxLength valid restriction

mgd-props-correct
Model Group Definition Properties Correct

mg-props-correct
Model Group Correct

minExclusive-less-than-equal-to-maxExclusive
minExclusive <= maxExclusive

minExclusive-less-than-maxInclusive
minExclusive < maxInclusive

minExclusive-valid-restriction
minExclusive valid restriction

minInclusive-less-than-equal-to-maxInclusive
minInclusive <= maxInclusive

minInclusive-less-than-maxExclusive
minInclusive < maxExclusive

minInclusive-minExclusive
minInclusive and minExclusive

minInclusive-valid-restriction
minInclusive valid restriction

minLength-less-than-equal-to-maxLength
minLength <= maxLength

minLength-valid-restriction
minLength valid restriction

no-xmlns
xmlns Not Allowed

no-xsi
xsi: Not Allowed
n-props-correct
Notation Declaration Correct
p-props-correct
Particle Correct
range-ok
Occurrence Range OK
rcase-MapAndSum
Particle Derivation OK (Sequence:Choice -- MapAndSum)
rcase-NameAndTypeOK
Particle Restriction OK (Elt:Elt -- NameAndTypeOK)
rcase-NSCompat
Particle Derivation OK (Elt:Any -- NSCompat)
rcase-NSRecurseCheckCardinality
Particle Derivation OK (All/Choice/Sequence:Any -- NSRecurseCheckCardinality)
rcase-NSSubset
Particle Derivation OK (Any:Any -- NSSubset)
rcase-Recurse
Particle Derivation OK (All:All,Sequence:Sequence -- Recurse)
rcase-RecurseAsIfGroup
Particle Derivation OK (Elt:All/Choice/Sequence -- RecurseAsIfGroup)
rcase-RecurseLax
Particle Derivation OK (Choice:Choice -- RecurseLax)
rcase-RecurseUnordered
Particle Derivation OK (Sequence:All -- RecurseUnordered)
sch-props-correct
Schema Properties Correct
st-props-correct
Simple Type Definition Properties Correct
totalDigits-valid-restriction
totalDigits valid restriction
whiteSpace-valid-restriction
whiteSpace valid restriction
w-props-correct
Wildcard Properties Correct

D Required Information Set Items and Properties (normative)

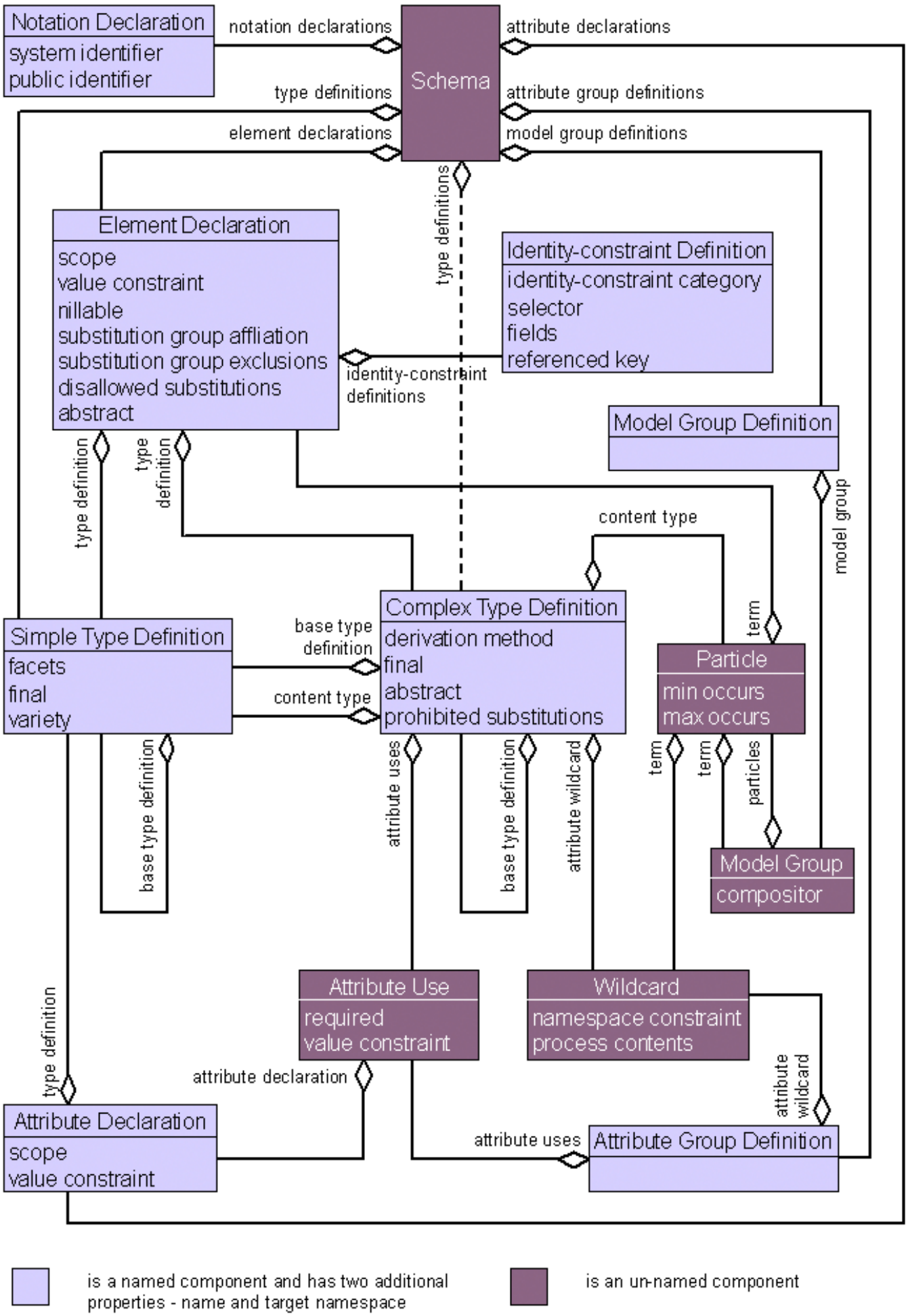
This specification requires as a precondition for ·assessment· an information set as defined in [XML-Infoset] which supports at least the following information items and properties:

Attribute Information Item
[local name], [namespace name], [normalized value]
Character Information Item
[character code]
Element Information Item
[local name], [namespace name], [children], [attributes], [in-scope namespaces] or [namespace attributes]
Namespace Information Item
[prefix], [namespace name]

This specification does not require any destructive alterations to the input information set: all the information set contributions specified herein are additive.

This appendix is intended to satisfy the requirements for Conformance to the [XML-Infoset] specification.

E Schema Components Diagram (non-normative)



XML Schema Component Data Model

F Glossary (non-normative)

The listing below is for the benefit of readers of a printed version of this document: it collects together all the definitions which appear in the document above.

absent
Throughout this specification, the term **absent** is used as a distinguished property value denoting absence

actual value
The phrase **actual value** is used to refer to the member of the value space of the simple type definition associated with an attribute information item which corresponds to its ·normalized value·

assessment
the word **assessment** is used to refer to the overall process of local validation, schema-validity assessment and infoset augmentation

base type definition
A type definition used as the basis for an ·extension· or ·restriction· is known as the **base type definition** of that definition

component name
Declarations and definitions may have and be identified by **name**s, which are NCNames as defined by [XML-Namespaces]

conformance to the XML Representation of Schemas
·Minimally conforming· processors which accept schemas represented in the form of XML documents as described in Layer 2: Schema Documents, Namespaces and Composition (§4.2) are additionally said to provide **conformance to the XML Representation of Schemas**.

content model
A particle can be used in a complex type definition to constrain the ·validation· of the [children] of an element information item; such a particle is called a **content model**

context-determined declaration
During ·validation·, associations between element and attribute information items among the [children] and [attributes] on the one hand, and element and attribute declarations on the other, are established as a side-effect. Such declarations are called the **context-determined declarations**

declaration
declaration components are associated by (qualified) name to information items being ·validated·

definition
definition components define internal schema components that can be used in other schema components

element substitution group
Through the new mechanism of **element substitution groups**, XML Schemas provides a more powerful model supporting substitution of one named element for another

extension
A complex type definition which allows element or attribute content in addition to that allowed by another specified type definition is said to be an **extension**

final
the complex type is said to be **final**, because no further derivations are possible

fully conforming
Fully conforming processors are network-enabled processors which are not only both ·minimally conforming· and ·in conformance to the XML Representation of Schemas·, but which additionally must be capable of accessing schema documents from the World Wide Web according to Representation of Schemas on the World Wide Web (§2.7) and How schema definitions are located on the Web (§4.3.2).

implicitly contains

A list of particles **implicitly contains** an element declaration if a member of the list contains that element declaration in its ·substitution group·

initial value

the **initial value** of some attribute information item is the value of the [normalized value] property of that item. Similarly, the **initial value** of an element information item is the string composed of, in order, the [character code] of each character information item in the [children] of that element information item

item isomorphic to a component

by an **item isomorphic** to a component is meant an information item whose type is equivalent to the component's, with one property per property of the component, with the same name, and value either the same atomic value, or an information item corresponding in the same way to its component value, recursively, as necessary

laxly assessed

an element information item's schema validity may be **laxly assessed** if its ·context-determined declaration· is not *skip* by ·validating· with respect to the ·ur-type definition· as perElement Locally Valid (Type) (§3.3.4)

minimally conforming

Minimally conforming processors must completely and correctly implement the ·Schema Component Constraints·, ·Validation Rules·, and ·Schema Information Set Contributions· contained in this specification

NCName

An **NCName** is a name with no colon, as defined in [XML-Namespaces]. When used in connection with the XML representation of schema components in this specification, this refers to the simple type NCName as defined in [XML Schemas: Datatypes]

normalized value

The **normalized value** of an element or attribute information item is an ·initial value· whose white space, if any, has been normalized according to the value of the whiteSpace facet of the simple type definition used in its ·validation·:

partition

Define a **partition** of a sequence as a sequence of sub-sequences, some or all of which may be empty, such that concatenating all the sub-sequences yields the original sequence

QName

A **QName** is a name with an optional namespace qualification, as defined in [XML-Namespaces]. When used in connection with the XML representation of schema components or references to them, this refers to the simple type QName as defined in [XML Schemas: Datatypes]

resolve

Whenever the word **resolve** in any form is used in this chapter in connection with a ·QName· in a schema document, the following definition QName resolution (Schema Document) (§3.15.3) should be understood

restriction

A type definition whose declarations or facets are in a one-to-one relation with those of another specified type definition, with each in turn restricting the possibilities of the one it corresponds to, is said to be a **restriction**

schema component

Schema component is the generic term for the building blocks that comprise the abstract data model of the schema.

Schema Component Constraint

Constraints on the schema components themselves, i.e. conditions components must satisfy to be components at all. Located in the sixth sub-section of the per-component sections of Schema Component Details (§3) and tabulated in Schema Component Constraints (§C.4)

schema document

A document in this form (i.e. a <schema> element information item) is a **schema document**

Schema Information Set Contribution

Augmentations to post-schema-validation infosets expressed by schema components, which follow as a consequence of ·validation· and/or ·assessment·. Located in the fifth sub-section of the per-

component sections of Schema Component Details (§3) and tabulated in Contributions to the post-schema-validation infoset (§C.2)

Schema Representation Constraint
Constraints on the representation of schema components in XML beyond those which are expressed in Schema for Schemas (normative) (§A). Located in the third sub-section of the per-component sections of Schema Component Details (§3) and tabulated in Schema Representation Constraints (§C.3)

symbol space
this specification introduces the term **symbol space** to denote a collection of names, each of which is unique with respect to the others

target namespace
Several kinds of component have a **target namespace**, which is either ·absent· or a namespace name, also as defined by [XML-Namespaces]

type definition
This specification uses the phrase **type definition** in cases where no distinction need be made between simple and complex types

Type Definition Hierarchy
Except for a distinguished ·ur-type definition·, every ·type definition· is, by construction, either a ·restriction· or an ·extension· of some other type definition. The graph of these relationships forms a tree known as the **Type Definition Hierarchy**

ur-type definition
A distinguished **ur-type definition** is present in each ·XML Schema·, serving as the root of the type definition hierarchy for that schema

valid
the word **valid** and its derivatives are used to refer to clause 1 above, the determination of local schema-validity

validation root
This item, that is the element information item at which ·assessment· began, is called the **validation root**

Validation Rules
Contributions to ·validation· associated with schema components. Located in the fourth sub-section of the per-component sections of Schema Component Details (§3) and tabulated in Validation Rules (§C.1)

valid extension
If this constraint Derivation Valid (Extension) (§3.4.6) holds of a complex type definition, it is a **valid extension** of its {base type definition}

valid restriction
If this constraint Derivation Valid (Restriction, Complex) (§3.4.6) holds of a complex type definition, it is a **valid restriction** of its {base type definition}

valid restriction
If this constraint Derivation Valid (Restriction, Simple) (§3.14.6) holds of a simple type definition, it is a **valid restriction** of its ·base type definition·

XML Schema
An **XML Schema** is a set of ·schema components·

G DTD for Schemas (non-normative)

The DTD for *XML Schema: Structures* is given below. Note there is *no* implication here the `schema` must be the root element of a document.

Although this DTD is non-normative, any XML document which is not valid per this DTD, given redefinitions in its internal subset of the 'p' and 's' parameter entities below appropriate to its namespace declaration of the XML Schema namespace, is almost certainly not a valid schema document, with the exception of documents with multiple namespace prefixes for the XML Schema namespace itself. Accordingly authoring XML Schema documents using this DTD and DTD-based authoring tools, and

specifying it as the DOCTYPE of documents intended to be XML Schema documents and validating them with a validating XML parser, are sensible development strategies which users are encouraged to adopt until XML Schema-based authoring tools and validators are more widely available.

```
<!-- DTD for XML Schemas: Part 1: Structures
     Public Identifier: "-//W3C//DTD XMLSCHEMA 200102//EN"
     Official Location: http://www.w3.org/2001/XMLSchema.dtd -->
<!-- Id: XMLSchema.dtd,v 1.30 2001/03/16 15:23:02 ht Exp  -->
<!-- With the exception of cases with multiple namespace
     prefixes for the XML Schema namespace, any XML document which is
     not valid per this DTD given redefinitions in its internal subset of
the
     'p' and 's' parameter entities below appropriate to its namespace
     declaration of the XML Schema namespace is almost certainly not
     a valid schema. -->

<!-- The simpleType element and its constituent parts
     are defined in XML Schema: Part 2: Datatypes -->
<!ENTITY % xs-datatypes PUBLIC 'datatypes' 'datatypes.dtd' >

<!ENTITY % p 'xs:'> <!-- can be overriden in the internal subset of a
                         schema document to establish a different
                         namespace prefix -->
<!ENTITY % s ':xs'> <!-- if %p is defined (e.g. as foo:) then you must
                         also define %s as the suffix for the appropriate
                         namespace declaration (e.g. :foo) -->
<!ENTITY % nds 'xmlns%s;'>

<!-- Define all the element names, with optional prefix -->
<!ENTITY % schema "%p;schema">
<!ENTITY % complexType "%p;complexType">
<!ENTITY % complexContent "%p;complexContent">
<!ENTITY % simpleContent "%p;simpleContent">
<!ENTITY % extension "%p;extension">
<!ENTITY % element "%p;element">
<!ENTITY % unique "%p;unique">
<!ENTITY % key "%p;key">
<!ENTITY % keyref "%p;keyref">
<!ENTITY % selector "%p;selector">
<!ENTITY % field "%p;field">
<!ENTITY % group "%p;group">
<!ENTITY % all "%p;all">
<!ENTITY % choice "%p;choice">
<!ENTITY % sequence "%p;sequence">
<!ENTITY % any "%p;any">
<!ENTITY % anyAttribute "%p;anyAttribute">
<!ENTITY % attribute "%p;attribute">
<!ENTITY % attributeGroup "%p;attributeGroup">
<!ENTITY % include "%p;include">
<!ENTITY % import "%p;import">
<!ENTITY % redefine "%p;redefine">
<!ENTITY % notation "%p;notation">

<!-- annotation elements -->
<!ENTITY % annotation "%p;annotation">
<!ENTITY % appinfo "%p;appinfo">
<!ENTITY % documentation "%p;documentation">

<!-- Customisation entities for the ATTLIST of each element type.
     Define one of these if your schema takes advantage of the
```

```
      anyAttribute='##other' in the schema for schemas -->

<!ENTITY % schemaAttrs ''>
<!ENTITY % complexTypeAttrs ''>
<!ENTITY % complexContentAttrs ''>
<!ENTITY % simpleContentAttrs ''>
<!ENTITY % extensionAttrs ''>
<!ENTITY % elementAttrs ''>
<!ENTITY % groupAttrs ''>
<!ENTITY % allAttrs ''>
<!ENTITY % choiceAttrs ''>
<!ENTITY % sequenceAttrs ''>
<!ENTITY % anyAttrs ''>
<!ENTITY % anyAttributeAttrs ''>
<!ENTITY % attributeAttrs ''>
<!ENTITY % attributeGroupAttrs ''>
<!ENTITY % uniqueAttrs ''>
<!ENTITY % keyAttrs ''>
<!ENTITY % keyrefAttrs ''>
<!ENTITY % selectorAttrs ''>
<!ENTITY % fieldAttrs ''>
<!ENTITY % includeAttrs ''>
<!ENTITY % importAttrs ''>
<!ENTITY % redefineAttrs ''>
<!ENTITY % notationAttrs ''>
<!ENTITY % annotationAttrs ''>
<!ENTITY % appinfoAttrs ''>
<!ENTITY % documentationAttrs ''>

<!ENTITY % complexDerivationSet "CDATA">
      <!-- #all or space-separated list drawn from derivationChoice -->
<!ENTITY % blockSet "CDATA">
      <!-- #all or space-separated list drawn from
                      derivationChoice + 'substitution' -->

<!ENTITY % mgs '%all; | %choice; | %sequence;'>
<!ENTITY % cs '%choice; | %sequence;'>
<!ENTITY % formValues '(qualified|unqualified)'>


<!ENTITY % attrDecls    '((%attribute;| %attributeGroup;)*,(%
anyAttribute;)?)'>

<!ENTITY % particleAndAttrs '((%mgs; | %group;)?, %attrDecls;)'>

<!-- This is used in part2 -->
<!ENTITY % restriction1 '((%mgs; | %group;)?)'>

%xs-datatypes;

<!-- the duplication below is to produce an unambiguous content model
     which allows annotation everywhere -->
<!ELEMENT %schema; ((%include; | %import; | %redefine; | %annotation;)*,
                    ((%simpleType; | %complexType;
                      | %element; | %attribute;
                      | %attributeGroup; | %group;
                      | %notation; ),
                     (%annotation;)*)* )>
<!ATTLIST %schema;
   targetNamespace      %URIref;              #IMPLIED
```

```
   version              CDATA                  #IMPLIED
   %nds;                %URIref;               #FIXED 'http://www.w3.
org/2001/XMLSchema'
   xmlns                CDATA                  #IMPLIED
   finalDefault         %complexDerivationSet; ''
   blockDefault         %blockSet;             ''
   id                   ID                     #IMPLIED
   elementFormDefault   %formValues;           'unqualified'
   attributeFormDefault %formValues;           'unqualified'
   xml:lang             CDATA                  #IMPLIED
   %schemaAttrs;>
<!-- Note the xmlns declaration is NOT in the Schema for Schemas,
     because at the Infoset level where schemas operate,
     xmlns(:prefix) is NOT an attribute! -->
<!-- The declaration of xmlns is a convenience for schema authors -->

<!-- The id attribute here and below is for use in external references
     from non-schemas using simple fragment identifiers.
     It is NOT used for schema-to-schema reference, internal or
     external. -->

<!-- a type is a named content type specification which allows attribute
     declarations-->
<!-- -->

<!ELEMENT %complexType; ((%annotation;)?,
                         (%simpleContent;|%complexContent;|
                          %particleAndAttrs;))>

<!ATTLIST %complexType;
          name      %NCName;                        #IMPLIED
          id        ID                              #IMPLIED
          abstract  %boolean;                       #IMPLIED
          final     %complexDerivationSet;          #IMPLIED
          block     %complexDerivationSet;          #IMPLIED
          mixed (true|false) 'false'
          %complexTypeAttrs;>

<!-- particleAndAttrs is shorthand for a root type -->
<!-- mixed is disallowed if simpleContent, overriden if complexContent
     has one too. -->

<!-- If anyAttribute appears in one or more referenced attributeGroups
     and/or explicitly, the intersection of the permissions is used -->

<!ELEMENT %complexContent; (%restriction;|%extension;)>
<!ATTLIST %complexContent;
          mixed (true|false) #IMPLIED
          id    ID           #IMPLIED
          %complexContentAttrs;>

<!-- restriction should use the branch defined above, not the simple
     one from part2; extension should use the full model  -->

<!ELEMENT %simpleContent; (%restriction;|%extension;)>
<!ATTLIST %simpleContent;
          id    ID           #IMPLIED
          %simpleContentAttrs;>

<!-- restriction should use the simple branch from part2, not the
```

```
     one defined above; extension should have no particle  -->

<!ELEMENT %extension; (%particleAndAttrs;)>
<!ATTLIST %extension;
          base  %QName;      #REQUIRED
          id    ID           #IMPLIED
          %extensionAttrs;>

<!-- an element is declared by either:
 a name and a type (either nested or referenced via the type attribute)
 or a ref to an existing element declaration -->

<!ELEMENT %element; ((%annotation;)?, (%complexType;| %simpleType;)?,
                     (%unique; | %key; | %keyref;)*)>
<!-- simpleType or complexType only if no type|ref attribute -->
<!-- ref not allowed at top level -->
<!ATTLIST %element;
            name               %NCName;               #IMPLIED
            id                 ID                     #IMPLIED
            ref                %QName;                #IMPLIED
            type               %QName;                #IMPLIED
            minOccurs          %nonNegativeInteger;   #IMPLIED
            maxOccurs          CDATA                  #IMPLIED
            nillable           %boolean;              #IMPLIED
            substitutionGroup  %QName;                #IMPLIED
            abstract           %boolean;              #IMPLIED
            final              %complexDerivationSet; #IMPLIED
            block              %blockSet;             #IMPLIED
            default            CDATA                  #IMPLIED
            fixed              CDATA                  #IMPLIED
            form               %formValues;           #IMPLIED
            %elementAttrs;>
<!-- type and ref are mutually exclusive.
     name and ref are mutually exclusive, one is required -->
<!-- In the absence of type AND ref, type defaults to type of
     substitutionGroup, if any, else the ur-type, i.e. unconstrained -->
<!-- default and fixed are mutually exclusive -->

<!ELEMENT %group; ((%annotation;)?,(%mgs;)?)>
<!ATTLIST %group;
          name        %NCName;               #IMPLIED
          ref         %QName;                #IMPLIED
          minOccurs   %nonNegativeInteger;   #IMPLIED
          maxOccurs   CDATA                  #IMPLIED
          id          ID                     #IMPLIED
          %groupAttrs;>

<!ELEMENT %all; ((%annotation;)?, (%element;)*)>
<!ATTLIST %all;
          minOccurs   (1)                    #IMPLIED
          maxOccurs   (1)                    #IMPLIED
          id          ID                     #IMPLIED
          %allAttrs;>

<!ELEMENT %choice; ((%annotation;)?, (%element;| %group;| %cs; | %any;)*)>
<!ATTLIST %choice;
          minOccurs   %nonNegativeInteger;   #IMPLIED
          maxOccurs   CDATA                  #IMPLIED
          id          ID                     #IMPLIED
          %choiceAttrs;>
```

```
<!ELEMENT %sequence; ((%annotation;)?, (%element;| %group;| %cs; | %any;)
*)>
<!ATTLIST %sequence;
          minOccurs   %nonNegativeInteger;   #IMPLIED
          maxOccurs   CDATA                  #IMPLIED
          id          ID                     #IMPLIED
          %sequenceAttrs;>

<!-- an anonymous grouping in a model, or
     a top-level named group definition, or a reference to same -->

<!-- Note that if order is 'all', group is not allowed inside.
     If order is 'all' THIS group must be alone (or referenced alone) at
     the top level of a content model -->
<!-- If order is 'all', minOccurs==maxOccurs==1 on element/any inside -->
<!-- Should allow minOccurs=0 inside order='all' . . . -->

<!ELEMENT %any; (%annotation;)?>
<!ATTLIST %any;
            namespace       CDATA                  '##any'
            processContents (skip|lax|strict)      'strict'
            minOccurs       %nonNegativeInteger;   '1'
            maxOccurs       CDATA                  '1'
            id              ID                     #IMPLIED
            %anyAttrs;>

<!-- namespace is interpreted as follows:
                  ##any      - - any non-conflicting WFXML at all

                  ##other    - - any non-conflicting WFXML from namespace
other
                                   than targetNamespace

                  ##local    - - any unqualified non-conflicting WFXML/
attribute
                  one or     - - any non-conflicting WFXML from
                  more URI        the listed namespaces
                  references

                  ##targetNamespace ##local may appear in the above list,
                    with the obvious meaning -->

<!ELEMENT %anyAttribute; (%annotation;)?>
<!ATTLIST %anyAttribute;
            namespace       CDATA              '##any'
            processContents (skip|lax|strict)  'strict'
            id              ID                 #IMPLIED
            %anyAttributeAttrs;>
<!-- namespace is interpreted as for 'any' above -->

<!-- simpleType only if no type|ref attribute -->
<!-- ref not allowed at top level, name iff at top level -->
<!ELEMENT %attribute; ((%annotation;)?, (%simpleType;)?)>
<!ATTLIST %attribute;
          name      %NCName;      #IMPLIED
          id        ID            #IMPLIED
          ref       %QName;       #IMPLIED
          type      %QName;       #IMPLIED
          use       (prohibited|optional|required) #IMPLIED
```

```
          default   CDATA         #IMPLIED
          fixed     CDATA         #IMPLIED
          form      %formValues;  #IMPLIED
          %attributeAttrs;>
<!-- type and ref are mutually exclusive.
     name and ref are mutually exclusive, one is required -->
<!-- default for use is optional when nested, none otherwise -->
<!-- default and fixed are mutually exclusive -->
<!-- type attr and simpleType content are mutually exclusive -->

<!-- an attributeGroup is a named collection of attribute decls, or a
     reference thereto -->
<!ELEMENT %attributeGroup; ((%annotation;)?,
                      (%attribute; | %attributeGroup;)*,
                      (%anyAttribute;)?) >
<!ATTLIST %attributeGroup;
                 name       %NCName;       #IMPLIED
                 id         ID             #IMPLIED
                 ref        %QName;        #IMPLIED
                 %attributeGroupAttrs;>

<!-- ref iff no content, no name.  ref iff not top level -->

<!-- better reference mechanisms -->
<!ELEMENT %unique; ((%annotation;)?, %selector;, (%field;)+)>
<!ATTLIST %unique;
          name     %NCName;       #REQUIRED
          id       ID             #IMPLIED
          %uniqueAttrs;>

<!ELEMENT %key;    ((%annotation;)?, %selector;, (%field;)+)>
<!ATTLIST %key;
          name     %NCName;       #REQUIRED
          id       ID             #IMPLIED
          %keyAttrs;>

<!ELEMENT %keyref; ((%annotation;)?, %selector;, (%field;)+)>
<!ATTLIST %keyref;
          name     %NCName;       #REQUIRED
          refer    %QName;        #REQUIRED
          id       ID             #IMPLIED
          %keyrefAttrs;>

<!ELEMENT %selector; ((%annotation;)?)>
<!ATTLIST %selector;
          xpath %XPathExpr; #REQUIRED
          id    ID          #IMPLIED
          %selectorAttrs;>
<!ELEMENT %field; ((%annotation;)?)>
<!ATTLIST %field;
          xpath %XPathExpr; #REQUIRED
          id    ID          #IMPLIED
          %fieldAttrs;>

<!-- Schema combination mechanisms -->
<!ELEMENT %include; (%annotation;)?>
<!ATTLIST %include;
          schemaLocation %URIref; #REQUIRED
          id             ID       #IMPLIED
          %includeAttrs;>
```

```
<!ELEMENT %import; (%annotation;)?>
<!ATTLIST %import;
          namespace      %URIref; #IMPLIED
          schemaLocation %URIref; #IMPLIED
          id             ID       #IMPLIED
          %importAttrs;>

<!ELEMENT %redefine; (%annotation; | %simpleType; | %complexType; |
                      %attributeGroup; | %group;)*>
<!ATTLIST %redefine;
          schemaLocation %URIref; #REQUIRED
          id             ID       #IMPLIED
          %redefineAttrs;>

<!ELEMENT %notation; (%annotation;)?>
<!ATTLIST %notation;
          name        %NCName;    #REQUIRED
          id          ID          #IMPLIED
          public      CDATA       #REQUIRED
          system      %URIref;    #IMPLIED
          %notationAttrs;>

<!-- Annotation is either application information or documentation -->
<!-- By having these here they are available for datatypes as well
     as all the structures elements -->

<!ELEMENT %annotation; (%appinfo; | %documentation;)*>
<!ATTLIST %annotation; %annotationAttrs;>

<!-- User must define annotation elements in internal subset for this
     to work -->
<!ELEMENT %appinfo; ANY>   <!-- too restrictive -->
<!ATTLIST %appinfo;
          source     %URIref;      #IMPLIED
          id         ID         #IMPLIED
          %appinfoAttrs;>
<!ELEMENT %documentation; ANY>   <!-- too restrictive -->
<!ATTLIST %documentation;
          source     %URIref;   #IMPLIED
          id         ID         #IMPLIED
          xml:lang   CDATA      #IMPLIED
          %documentationAttrs;>

<!NOTATION XMLSchemaStructures PUBLIC
           'structures' 'http://www.w3.org/2001/XMLSchema.xsd' >
<!NOTATION XML PUBLIC
           'REC-xml-1998-0210' 'http://www.w3.org/TR/1998/REC-xml-
19980210' >
```

H Analysis of the Unique Particle Attribution Constraint (non-normative)

A specification of the import of Unique Particle Attribution (§3.8.6) which does not appeal to a processing model is difficult. What follows is intended as guidance, without claiming to be complete.

[Definition:] Two non-group particles **overlap** if

- They are both element declaration particles whose declarations have the same {name} and {target namespace}.

or

- They are both element declaration particles one of which is in the other's ·substitution group·.

or

- They are both wildcards, and the intensional intersection of their {namespace constraint}s as defined in Attribute Wildcard Intersection (§3.10.6) is not the empty set.

or

- One is a wildcard and the other an element declaration, and the {target namespace} of the element declaration, or of any member of its ·substitution group·, is ·valid· with respect to the {namespace constraint} of the wildcard.

A content model will violate the unique attribution constraint if it contains two particles which ·overlap· and which either

- are both in the {particles} of a *choice* or *all* group

or

- may ·validate· adjacent information items and the first has {min occurs} less than {max occurs}.

Two particles may ·validate· adjacent information items if they are separated by at most epsilon transitions in the most obvious transcription of a content model into a finite-state automaton.

A precise formulation of this constraint can also be offered in terms of operations on finite-state automaton: transcribe the content model into an automaton in the usual way using epsilon transitions for optionality and unbounded maxOccurs, unfolding other numeric occurrence ranges and treating the heads of substitution groups as if they were choices over all elements in the group, *but* using not element QNames as transition labels, but rather pairs of element QNames and positions in the model. Determinize this automaton, treating wildcard transitions as opaque. Now replace all QName+position transition labels with the element QNames alone. If the result has any states with two or more identical-QName-labeled transitions from it, or a QName-labeled transition and a wildcard transition which subsumes it, or two wildcard transitions whose intentional intersection is non-empty, the model does not satisfy the Unique Attribution constraint.

I References (non-normative)

DCD
Document Content Description for XML (DCD), Tim Bray et al., eds., W3C, 10 August 1998. See http://www.w3.org/TR/1998/NOTE-dcd-19980731

DDML
Document Definition Markup Language, Ronald Bourret, John Cowan, Ingo Macherius, Simon St. Laurent, eds., W3C, 19 January 1999. See http://www.w3.org/TR/1999/NOTE-ddml-19990119

SOX
Schema for Object-oriented XML, Andrew Davidson et al., eds., W3C, 1998. See http://www.w3.org/1999/07/NOTE-SOX-19990730/

SOX-2
Schema for Object-oriented XML, Version 2.0, Andrew Davidson, et al., W3C, 30 July 1999. See http://www.w3.org/TR/NOTE-SOX/

XDR

XML-Data Reduced, Charles Frankston and Henry S. Thompson, 3 July 1998. See http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm

XML-Data

XML-Data, Andrew Layman et al., W3C, 05 January 1998. See http://www.w3.org/TR/1998/NOTE-XML-data-0105/

XML Schema: Primer

XML Schema Part 0: Primer, David C. Fallside, ed., W3C, 2 May 2001. See http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/primer.html

J Acknowledgements (non-normative)

The following have contributed material to this draft:

- David Fallside, IBM
- Scott Lawrence, Agranat Systems
- Andrew Layman, Microsoft
- Eve L. Maler, Sun Microsystems
- Asir S. Vedamuthu, webMethods, Inc

The editors acknowledge the members of the XML Schema Working Group, the members of other W3C Working Groups, and industry experts in other forums who have contributed directly or indirectly to the process or content of creating this document. The Working Group is particularly grateful to Lotus Development Corp. and IBM for providing teleconferencing facilities.

The current members of the XML Schema Working Group are:

Jim Barnette, Defense Information Systems Agency (DISA); Paul V. Biron, Health Level Seven; Don Box, DevelopMentor; Allen Brown, Microsoft; Lee Buck, TIBCO Extensibility; Charles E. Campbell, Informix; Wayne Carr, Intel; Peter Chen, Bootstrap Alliance and LSU; David Cleary, Progress Software; Dan Connolly, W3C (staff contact); Ugo Corda, Xerox; Roger L. Costello, MITRE; Haavard Danielson, Progress Software; Josef Dietl, Mozquito Technologies; David Ezell, Hewlett-Packard Company; Alexander Falk, Altova GmbH; David Fallside, IBM; Dan Fox, Defense Logistics Information Service (DLIS); Matthew Fuchs, Commerce One; Andrew Goodchild, Distributed Systems Technology Centre (DSTC Pty Ltd); Paul Grosso, Arbortext, Inc; Martin Gudgin, DevelopMentor; Dave Hollander, Contivo, Inc (co-chair); Mary Holstege, Invited Expert; Jane Hunter, Distributed Systems Technology Centre (DSTC Pty Ltd); Rick Jelliffe, Academia Sinica; Simon Johnston, Rational Software; Bob Lojek, Mozquito Technologies; Ashok Malhotra, Microsoft; Lisa Martin, IBM; Noah Mendelsohn, Lotus Development Corporation; Adrian Michel, Commerce One; Alex Milowski, Invited Expert; Don Mullen, TIBCO Extensibility; Dave Peterson, Graphic Communications Association; Jonathan Robie, Software AG; Eric Sedlar, Oracle Corp.; C. M. Sperberg-McQueen, W3C (co-chair); Bob Streich, Calico Commerce; William K. Stumbo, Xerox; Henry S. Thompson, University of Edinburgh; Mark Tucker, Health Level Seven; Asir S. Vedamuthu, webMethods, Inc; Priscilla Walmsley, XMLSolutions; Norm Walsh, Sun Microsystems; Aki Yoshida, SAP AG; Kongyi Zhou, Oracle Corp.

The XML Schema Working Group has benefited in its work from the participation and contributions of a number of people not currently members of the Working Group, including in particular those named below. Affiliations given are those current at the time of their work with the WG.

Paula Angerstein, Vignette Corporation; David Beech, Oracle Corp.; Gabe Beged-Dov, Rogue Wave Software; Greg Bumgardner, Rogue Wave Software; Dean Burson, Lotus Development Corporation; Mike Cokus, MITRE; Andrew Eisenberg, Progress Software; Rob Ellman, Calico Commerce; George Feinberg, Object Design; Charles Frankston, Microsoft; Ernesto Guerrieri, Inso; Michael Hyman, Microsoft; Renato Iannella, Distributed Systems Technology Centre (DSTC Pty Ltd); Dianne Kennedy, Graphic Communications Association; Janet Koenig, Sun Microsystems; Setrag Khoshafian, Technology Deployment International (TDI); Ara Kullukian, Technology Deployment International (TDI); Andrew Layman, Microsoft; Dmitry Lenkov, Hewlett-Packard Company; John McCarthy, Lawrence Berkeley National Laboratory; Murata Makoto, Xerox; Eve Maler, Sun Microsystems; Murray Maloney, Muzmo Communication, acting for Commerce One; Chris Olds, Wall Data; Frank Olken, Lawrence Berkeley

National Laboratory; Shriram Revankar, Xerox; Mark Reinhold, Sun Microsystems; John C. Schneider, MITRE; Lew Shannon, NCR; William Shea, Merrill Lynch; Ralph Swick, W3C; Tony Stewart, Rivcom; Matt Timmermans, Microstar; Jim Trezzo, Oracle Corp.; Steph Tryphonas, Microstar

XML Schema Part 2: Datatypes

W3C Recommendation 02 May 2001

This version:
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
(in XML and HTML, with a schema and DTD including datatype definitions, as well as a schema for built-in datatypes only, in a separate namespace.)
Latest version:
http://www.w3.org/TR/xmlschema-2/
Previous version:
http://www.w3.org/TR/2001/PR-xmlschema-2-20010330/
Editors:
Paul V. Biron (Kaiser Permanente, for Health Level Seven) <Paul.V.Biron@kp.org>
Ashok Malhotra (Microsoft, formerly of IBM) <ashokma@microsoft.com>

Abstract

XML Schema: Datatypes is part 2 of the specification of the XML Schema language. It defines facilities for defining datatypes to be used in XML Schemas as well as other XML specifications. The datatype language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs) for specifying datatypes on elements and attributes.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document has been produced by the W3C XML Schema Working Group as part of the W3C XML Activity. The goals of the XML Schema language are discussed in the XML Schema Requirements document. The authors of this document are the XML Schema WG members. Different parts of this specification have different editors.

This version of this document incorporates some editorial changes from earlier versions.

Please report errors in this document to www-xml-schema-comments@w3.org (archive). The list of known errors in this specification is available at http://www.w3.org/2001/05/xmlschema-errata.

The English version of this specification is the only normative version. Information about translations of this document is available at http://www.w3.org/2001/05/xmlschema-translations.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/.

Table of contents

Appendices

---

1 Introduction

1.1 Purpose

The [XML 1.0 (Second Edition)] specification defines limited facilities for applying datatypes to document content in that documents may contain or refer to DTDs that assign types to elements and attributes. However, document authors, including authors of traditional *documents* and those transporting *data* in XML, often require a higher degree of type checking to ensure robustness in document understanding and data interchange.

The table below offers two typical examples of XML instances in which datatypes are implicit: the instance on the left represents a billing invoice, the instance on the right a memo or perhaps an email message in XML.

Data oriented	Document oriented

```
<invoice>
  <orderDate>1999-01-21</orderDate>
  <shipDate>1999-01-25</shipDate>
  <billingAddress>
   <name>Ashok Malhotra</name>
   <street>123 Microsoft Ave.</street>
   <city>Hawthorne</city>
   <state>NY</state>
   <zip>10532-0000</zip>
  </billingAddress>
  <voice>555-1234</voice>
  <fax>555-4321</fax>
</invoice>
```

```
<memo importance='high'
      date='1999-03-23'>
  <from>Paul V. Biron</from>
  <to>Ashok Malhotra</to>
  <subject>Latest draft</subject>
  <body>
    We need to discuss the latest
    draft <emph>immediately</emph>.
    Either email me at <email>
    mailto:paul.v.biron@kp.org</email>
    or call <phone>555-9876</phone>
  </body>
</memo>
```

The invoice contains several dates and telephone numbers, the postal abbreviation for a state (which comes from an enumerated list of sanctioned values), and a ZIP code (which takes a definable regular form). The memo contains many of the same types of information: a date, telephone number, email address and an "importance" value (from an enumerated list, such as "low", "medium" or "high"). Applications which process invoices and memos need to raise exceptions if something that was supposed to be a date or telephone number does not conform to the rules for valid dates or telephone numbers.

In both cases, validity constraints exist on the content of the instances that are not expressible in XML DTDs. The limited datatyping facilities in XML have prevented validating XML processors from supplying the rigorous type checking required in these situations. The result has been that individual applications writers have had to implement type checking in an ad hoc manner. This specification addresses the need of both document authors and applications writers for a robust, extensible datatype system for XML which could be incorporated into XML processors. As discussed below, these datatypes could be used in other XML-related standards as well.

1.2 Requirements

The [XML Schema Requirements] document spells out concrete requirements to be fulfilled by this specification, which state that the XML Schema Language must:

1. provide for primitive data typing, including byte, date, integer, sequence, SQL and Java primitive datatypes, etc.;
2. define a type system that is adequate for import/export from database systems (e.g., relational, object, OLAP);
3. distinguish requirements relating to lexical data representation vs. those governing an underlying information set;
4. allow creation of user-defined datatypes, such as datatypes that are derived from existing datatypes and which may constrain certain of its properties (e.g., range, precision, length, format).

1.3 Scope

This portion of the XML Schema Language discusses datatypes that can be used in an XML Schema. These datatypes can be specified for element content that would be specified as #PCDATA and attribute values of various types in a DTD. It is the intention of this specification that it be usable outside of the context of XML Schemas for a wide range of other XML-related activities such as [XSL] and [RDF Schema].

1.4 Terminology

The terminology used to describe XML Schema Datatypes is defined in the body of this specification. The terms defined in the following list are used in building those definitions and in describing the actions of a datatype processor:

[Definition:] for compatibility
A feature of this specification included solely to ensure that schemas which use this feature remain compatible with [XML 1.0 (Second Edition)]

[Definition:] may
Conforming documents and processors are permitted to but need not behave as described.

[Definition:] match
(Of strings or names:) Two strings or names being compared must be identical. Characters with multiple possible representations in ISO/IEC 10646 (e.g. characters with both precomposed and base+diacritic forms) match only if they have the same representation in both strings. No case folding is performed. (Of strings and rules in the grammar:) A string matches a grammatical production if it belongs to the language generated by that production.

[Definition:] must
Conforming documents and processors are required to behave as described; otherwise they are in ·error·.

[Definition:] error
A violation of the rules of this specification; results are undefined. Conforming software ·may· detect and report an **error** and ·may· recover from it.

1.5 Constraints and Contributions

This specification provides three different kinds of normative statements about schema components, their representations in XML and their contribution to the schema-validation of information items:

[Definition:] Constraint on Schemas
Constraints on the schema components themselves, i.e. conditions components ·must· satisfy to be components at all. Largely to be found in Datatype components (§4).

[Definition:] Schema Representation Constraint
Constraints on the representation of schema components in XML. Some but not all of these are expressed in Schema for Datatype Definitions (normative) (§A) and DTD for Datatype Definitions (non-normative) (§B).

[Definition:] Validation Rule
Constraints expressed by schema components which information items ·must· satisfy to be schema-valid. Largely to be found in Datatype components (§4).

2 Type System

This section describes the conceptual framework behind the type system defined in this specification. The framework has been influenced by the [ISO 11404] standard on language-independent datatypes as well as the datatypes for [SQL] and for programming languages such as Java.

The datatypes discussed in this specification are computer representations of well known abstract concepts such as *integer* and *date*. It is not the place of this specification to define these abstract concepts; many other publications provide excellent definitions.

2.1 Datatype

[Definition:] In this specification, a **datatype** is a 3-tuple, consisting of a) a set of distinct values, called its ·value space·, b) a set of lexical representations, called its ·lexical space·, and c) a set of ·facet·s that characterize properties of the ·value space·, individual values or lexical items.

2.2 Value space

[Definition:] A **value space** is the set of values for a given datatype. Each value in the **value space** of a datatype is denoted by one or more literals in its ·lexical space·.

The ·value space· of a given datatype can be defined in one of the following ways:

- defined axiomatically from fundamental notions (intensional definition) [see ·primitive·]
- enumerated outright (extensional definition) [see ·enumeration·]
- defined by restricting the ·value space· of an already defined datatype to a particular subset with a given set of properties [see ·derived·]
- defined as a combination of values from one or more already defined ·value space·(s) by a specific construction procedure [see ·list· and ·union·]

·value space·s have certain properties. For example, they always have the property of ·cardinality·, some definition of *equality* and might be ·ordered·, by which individual values within the ·value space· can be compared to one another. The properties of ·value space·s that are recognized by this specification are defined in Fundamental facets (§2.4.1).

2.3 Lexical space

In addition to its ·value space·, each datatype also has a lexical space.

[Definition:] A **lexical space** is the set of valid *literals* for a datatype.

For example, "100" and "1.0E2" are two different literals from the ·lexical space· of float which both denote the same value. The type system defined in this specification provides a mechanism for schema designers to control the set of values and the corresponding set of acceptable literals of those values for a datatype.

> **NOTE:** The literals in the ·lexical space·s defined in this specification have the following characteristics:
>
> **Interoperability:**
> The number of literals for each value has been kept small; for many datatypes there is a one-to-one mapping between literals and values. This makes it easy to exchange the values between different systems. In many cases, conversion from locale-dependent representations will be required on both the originator and the recipient side, both for computer processing and for interaction with humans.
> **Basic readability:**
> Textual, rather than binary, literals are used. This makes hand editing, debugging, and similar activities possible.
> **Ease of parsing and serializing:**
> Where possible, literals correspond to those found in common programming languages and libraries.

2.3.1 Canonical Lexical Representation

While the datatypes defined in this specification have, for the most part, a single lexical representation i.e. each value in the datatype's ·value space· is denoted by a single literal in its ·lexical space·, this is not always the case. The example in the previous section showed two literals for the datatype float which denote the same value. Similarly, there ·may· be several literals for one of the date or time datatypes that denote the same value using different timezone indicators.

[Definition:] A **canonical lexical representation** is a set of literals from among the valid set of literals for a datatype such that there is a one-to-one mapping between literals in the **canonical lexical representation** and values in the ·value space·.

2.4 Facets

[Definition:] A **facet** is a single defining aspect of a ·value space·. Generally speaking, each facet characterizes a ·value space· along independent axes or dimensions.

The facets of a datatype serve to distinguish those aspects of one datatype which *differ* from other datatypes. Rather than being defined solely in terms of a prose description the datatypes in this specification are defined in terms of the *synthesis* of facet values which together determine the ·value space· and properties of the datatype.

Facets are of two types: *fundamental* facets that define the datatype and *non-fundamental* or *constraining* facets that constrain the permitted values of a datatype.

2.4.1 Fundamental facets

[Definition:] A **fundamental facet** is an abstract property which serves to semantically characterize the values in a ·value space·.

All **fundamental facets** are fully described in Fundamental Facets (§4.2).

2.4.2 Constraining or Non-fundamental facets

[Definition:] A **constraining facet** is an optional property that can be applied to a datatype to constrain its ·value space·.

Constraining the ·value space· consequently constrains the ·lexical space·. Adding ·constraining facet·s to a ·base type· is described in Derivation by restriction (§4.1.2.1).

All **constraining facets** are fully described in Constraining Facets (§4.3).

2.5 Datatype dichotomies

It is useful to categorize the datatypes defined in this specification along various dimensions, forming a set of characterization dichotomies.

2.5.1 Atomic vs. list vs. union datatypes

The first distinction to be made is that between ·atomic·, ·list· and ·union· datatypes.

- [Definition:] **Atomic** datatypes are those having values which are regarded by this specification as being indivisible.
- [Definition:] **List** datatypes are those having values each of which consists of a finite-length (possibly empty) sequence of values of an ·atomic· datatype.
- [Definition:] **Union** datatypes are those whose ·value space·s and ·lexical space·s are the union of the ·value space·s and ·lexical space·s of one or more other datatypes.

For example, a single token which ·match·es Nmtoken from [XML 1.0 (Second Edition)] could be the value of an ·atomic· datatype (NMTOKEN); while a sequence of such tokens could be the value of a ·list· datatype (NMTOKENS).

2.5.1.1 Atomic datatypes

·atomic· datatypes can be either ·primitive· or ·derived·. The ·value space· of an ·atomic· datatype is a set of "atomic" values, which for the purposes of this specification, are not further decomposable. The ·lexical space· of an ·atomic· datatype is a set of *literals* whose internal structure is specific to the datatype in question.

2.5.1.2 List datatypes

Several type systems (such as the one described in [ISO 11404]) treat ·list· datatypes as special cases of the more general notions of aggregate or collection datatypes.

·list· datatypes are always ·derived·. The ·value space· of a ·list· datatype is a set of finite-length sequences of ·atomic· values. The ·lexical space· of a ·list· datatype is a set of literals whose internal structure is a white space separated sequence of literals of the ·atomic· datatype of the items in the ·list· (where whitespace ·match·esS in [XML 1.0 (Second Edition)]).

[Definition:] The ·atomic· datatype that participates in the definition of a ·list· datatype is known as the **itemType** of that ·list· datatype.

Example

```
<simpleType name='sizes'>
  <list itemType='decimal'/>
</simpleType>
<cerealSizes xsi:type='sizes'> 8 10.5 12 </cerealSizes>
```

A ·list· datatype can be ·derived· from an ·atomic· datatype whose ·lexical space· allows whitespace (such as string or anyURI). In such a case, regardless of the input, list items will be separated at whitespace boundaries.

Example

```
<simpleType name='listOfString'>
  <list itemType='string'/>
</simpleType>
<someElement xsi:type='listOfString'>
this is not list item 1
this is not list item 2
this is not list item 3
</someElement>
```

In the above example, the value of the *someElement* element is not a ·list· of ·length· 3; rather, it is a ·list· of ·length· 18.

When a datatype is ·derived· from a ·list· datatype, the following ·constraining facet·s apply:

- ·length·
- ·maxLength·
- ·minLength·
- ·enumeration·
- ·pattern·
- ·whiteSpace·

For each of ·length·, ·maxLength· and ·minLength·, the *unit of length* is measured in number of list items. The value of ·whiteSpace· is fixed to the value *collapse*.

The canonical-lexical-representation for the ·list· datatype is defined as the lexical form in which each item in the ·list· has the canonical lexical representation of its ·itemType·.

2.5.1.3 Union datatypes

The ·value space· and ·lexical space· of a ·union· datatype are the union of the ·value space·s and ·lexical space·s of its ·memberTypes·. ·union· datatypes are always ·derived·. Currently, there are no ·built-in· ·union· datatypes.

Example

A prototypical example of a ·union· type is the maxOccurs attribute on the element element in XML Schema itself: it is a union of nonNegativeInteger and an enumeration with the single member, the string "unbounded", as shown below.

```
<attributeGroup name="occurs">
  <attribute name="minOccurs" type="nonNegativeInteger"
      default="1"/>
  <attribute name="maxOccurs">
    <simpleType>
      <union>
        <simpleType>
          <restriction base='nonNegativeInteger'/>
        </simpleType>
```

```
          <simpleType>
            <restriction base='string'>
              <enumeration value='unbounded'/>
            </restriction>
          </simpleType>
        </union>
      </simpleType>
    </attribute>
  </attributeGroup>
```

Any number (greater than 1) of ·atomic· or ·list· ·datatype·s can participate in a ·union· type.

[Definition:] The datatypes that participate in the definition of a ·union· datatype are known as the **memberTypes** of that ·union· datatype.

The order in which the ·memberTypes· are specified in the definition (that is, the order of the <simpleType> children of the <union> element, or the order of the QNames in the *memberTypes* attribute) is significant. During validation, an element or attribute's value is validated against the ·memberTypes· in the order in which they appear in the definition until a match is found. The evaluation order can be overridden with the use of xsi:type.

Example

For example, given the definition below, the first instance of the <size> element validates correctly as an integer (§3.3.13), the second and third as string (§3.2.1).

```
<xsd:element name='size'>
  <xsd:simpleType>
    <xsd:union>
      <xsd:simpleType>
        <xsd:restriction base='integer'/>
      </xsd:simpleType>
      <xsd:simpleType>
        <xsd:restriction base='string'/>
      </xsd:simpleType>
    </xsd:union>
  </xsd:simpleType>
</xsd:element>
<size>1</size>
<size>large</size>
<size xsi:type='xsd:string'>1</size>
```

The canonical-lexical-representation for a ·union· datatype is defined as the lexical form in which the values have the canonical lexical representation of the appropriate ·memberTypes·.

> **NOTE:** A datatype which is ·atomic· in this specification need not be an "atomic" datatype in any programming language used to implement this specification. Likewise, a datatype which is a ·list· in this specification need not be a "list" datatype in any programming language used to implement this specification. Furthermore, a datatype which is a ·union· in this specification need not be a "union" datatype in any programming language used to implement this specification.

2.5.2 Primitive vs. derived datatypes

Next, we distinguish between ·primitive· and ·derived· datatypes.

- [Definition:] **Primitive** datatypes are those that are not defined in terms of other datatypes; they exist *ab initio*.
- [Definition:] **Derived** datatypes are those that are defined in terms of other datatypes.

For example, in this specification, float is a well-defined mathematical concept that cannot be defined in terms of

other datatypes, while a integer is a special case of the more general datatype decimal.

[Definition:] There exists a conceptual datatype, whose name is **anySimpleType**, that is the simple version of the ur-type definition from [XML Schema Part 1: Structures]. **anySimpleType** can be considered as the ·base type· of all ·primitive· types. The ·value space· of **anySimpleType** can be considered to be the ·union· of the ·value space·s of all ·primitive· datatypes.

The datatypes defined by this specification fall into both the ·primitive· and ·derived· categories. It is felt that a judiciously chosen set of ·primitive· datatypes will serve the widest possible audience by providing a set of convenient datatypes that can be used as is, as well as providing a rich enough base from which the variety of datatypes needed by schema designers can be ·derived·.

In the example above, integer is ·derived· from decimal.

> **NOTE:** A datatype which is ·primitive· in this specification need not be a "primitive" datatype in any programming language used to implement this specification. Likewise, a datatype which is ·derived· in this specification need not be a "derived" datatype in any programming language used to implement this specification.

As described in more detail in XML Representation of Simple Type Definition Schema Components (§4.1.2), each ·user-derived· datatype ·must· be defined in terms of another datatype in one of three ways: 1) by assigning ·constraining facet·s which serve to *restrict* the ·value space· of the ·user-derived· datatype to a subset of that of the ·base type·; 2) by creating a ·list· datatype whose ·value space· consists of finite-length sequences of values of its ·itemType·; or 3) by creating a ·union· datatype whose ·value space· consists of the union of the ·value space· its ·memberTypes·.

2.5.2.1 Derived by restriction

[Definition:] A datatype is said to be ·derived· by **restriction** from another datatype when values for zero or more ·constraining facet·s are specified that serve to constrain its ·value space· and/or its ·lexical space· to a subset of those of its ·base type·.

[Definition:] Every datatype that is ·derived· by **restriction** is defined in terms of an existing datatype, referred to as its **base type**. **base type**s can be either ·primitive· or ·derived·.

2.5.2.2 Derived by list

A ·list· datatype can be ·derived· from another datatype (its ·itemType·) by creating a ·value space· that consists of a finite-length sequence of values of its ·itemType·.

2.5.2.3 Derived by union

One datatype can be ·derived· from one or more datatypes by ·union·ing their ·value space·s and, consequently, their ·lexical space·s.

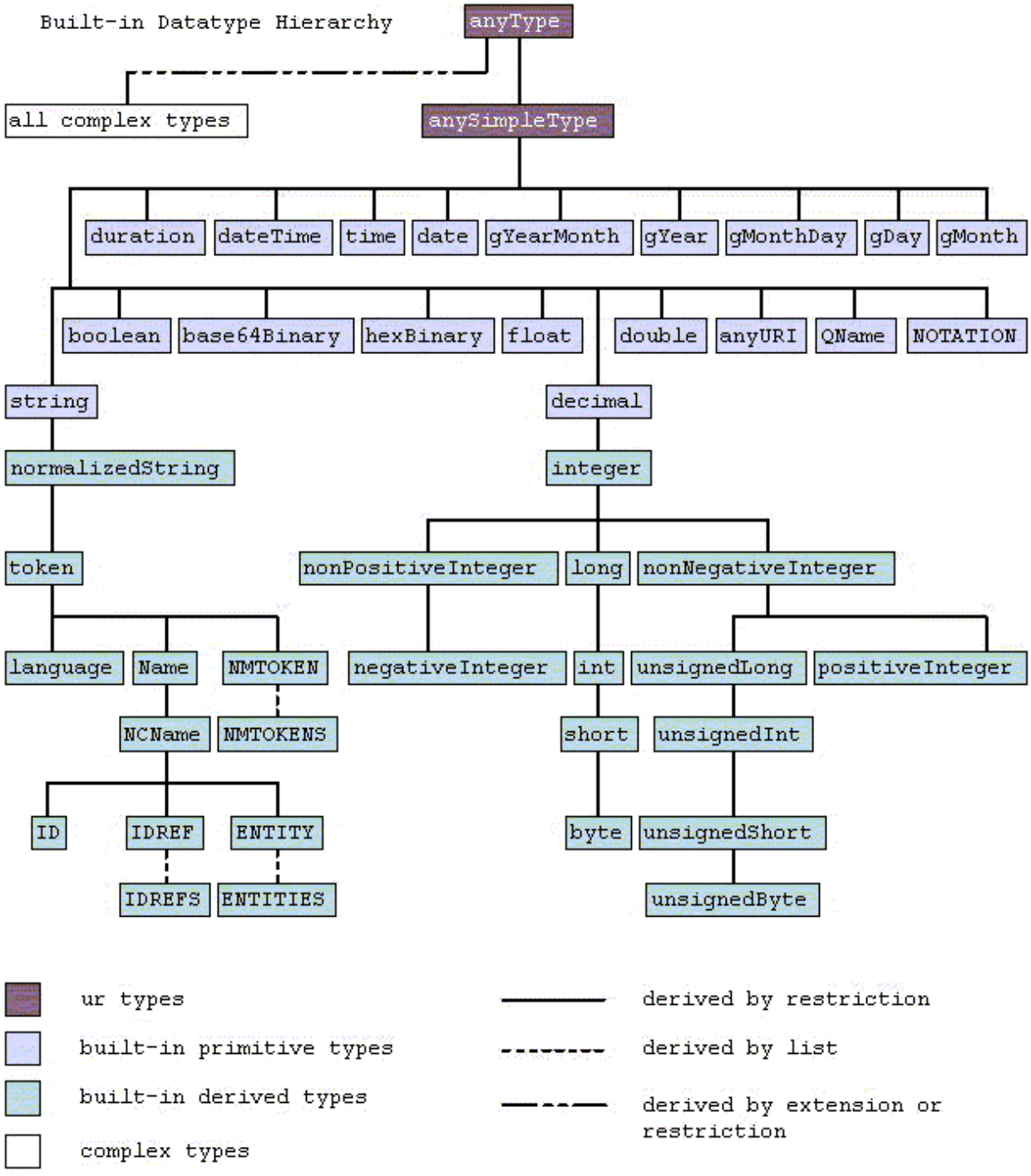
2.5.3 Built-in vs. user-derived datatypes

- [Definition:] **Built-in** datatypes are those which are defined in this specification, and can be either ·primitive· or ·derived·;
- [Definition:] **User-derived** datatypes are those ·derived· datatypes that are defined by individual schema designers.

Conceptually there is no difference between the ·built-in· ·derived· datatypes included in this specification and the ·user-derived· datatypes which will be created by individual schema designers. The ·built-in· ·derived· datatypes are those which are believed to be so common that if they were not defined in this specification many schema designers would end up "reinventing" them. Furthermore, including these ·derived· datatypes in this specification serves to demonstrate the mechanics and utility of the datatype generation facilities of this specification.

NOTE: A datatype which is ·built-in· in this specification need not be a "built-in" datatype in any programming language used to implement this specification. Likewise, a datatype which is ·user-derived· in this specification need not be a "user-derived" datatype in any programming language used to implement this specification.

3 Built-in datatypes



Each built-in datatype in this specification (both ·primitive· and ·derived·) can be uniquely addressed via a URI Reference constructed as follows:

1. the base URI is the URI of the XML Schema namespace
2. the fragment identifier is the name of the datatype

For example, to address the int datatype, the URI is:

- `http://www.w3.org/2001/XMLSchema#int`

Additionally, each facet definition element can be uniquely addressed via a URI constructed as follows:

1. the base URI is the URI of the XML Schema namespace
2. the fragment identifier is the name of the facet

For example, to address the maxInclusive facet, the URI is:

- `http://www.w3.org/2001/XMLSchema#maxInclusive`

Additionally, each facet usage in a built-in datatype definition can be uniquely addressed via a URI constructed as follows:

1. the base URI is the URI of the XML Schema namespace
2. the fragment identifier is the name of the datatype, followed by a period (".") followed by the name of the facet

For example, to address the usage of the maxInclusive facet in the definition of int, the URI is:

- `http://www.w3.org/2001/XMLSchema#int.maxInclusive`

3.1 Namespace considerations

The ·built-in· datatypes defined by this specification are designed to be used with the XML Schema definition language as well as other XML specifications. To facilitate usage within the XML Schema definition language, the ·built-in· datatypes in this specification have the namespace name:

- http://www.w3.org/2001/XMLSchema

To facilitate usage in specifications other than the XML Schema definition language, such as those that do not want to know anything about aspects of the XML Schema definition language other than the datatypes, each ·built-in· datatype is also defined in the namespace whose URI is:

- http://www.w3.org/2001/XMLSchema-datatypes

This applies to both ·built-in· ·primitive· and ·built-in· ·derived· datatypes.

Each ·user-derived· datatype is also associated with a unique namespace. However, ·user-derived· datatypes do not come from the namespace defined by this specification; rather, they come from the namespace of the schema in which they are defined (see XML Representation of Schemas in [XML Schema Part 1: Structures]).

3.2 Primitive datatypes

The ·primitive· datatypes defined by this specification are described below. For each datatype, the ·value space· and ·lexical space· are defined, ·constraining facet·s which apply to the datatype are listed and any datatypes ·derived· from this datatype are specified.

·primitive· datatypes can only be added by revisions to this specification.

3.2.1 string

[Definition:] The **string** datatype represents character strings in XML. The ·value space· of **string** is the set of finite-length sequences of characters (as defined in [XML 1.0 (Second Edition)]) that ·match· the Char production from [XML 1.0 (Second Edition)]. A character is an atomic unit of communication; it is not further specified except to note that every character has a corresponding Universal Character Set code point, which is an integer.

> **NOTE:** Many human languages have writing systems that require child elements for control of aspects such as bidirectional formating or ruby annotation (see [Ruby] and Section 8.2.4 Overriding the bidirectional algorithm: the BDO element of [HTML 4.01]). Thus, **string**, as a simple type that can contain only characters but not child elements, is often not suitable for representing text. In such situations, a complex type that allows mixed content should be considered. For more information, see Section 5.5 Any Element, Any Attribute of [XML Schema Language: Part 2 Primer].
>
> **NOTE:** As noted in ordered, the fact that this specification does not specify an ·order-relation· for ·string· does not preclude other applications from treating strings as being ordered.

3.2.1.1 Constraining facets

string has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.2.1.2 Derived datatypes

The following ·built-in· datatypes are ·derived· from **string**:

- normalizedString

3.2.2 boolean

[Definition:] **boolean** has the ·value space· required to support the mathematical concept of binary-valued logic: {true, false}.

3.2.2.1 Lexical representation

An instance of a datatype that is defined as ·boolean· can have the following legal literals {true, false, 1, 0}.

3.2.2.2 Canonical representation

The canonical representation for **boolean** is the set of literals {true, false}.

3.2.2.3 Constraining facets

boolean has the following ·constraining facets·:

- pattern
- whiteSpace

3.2.3 decimal

[Definition:] **decimal** represents arbitrary precision decimal numbers. The ·value space· of **decimal** is the set of the values *i* × *10*^-*n*, where *i* and *n* are integers such that *n* >= *0*. The ·order-relation· on **decimal** is: *x* < *y iff y* - *x* is positive.

[Definition:] The ·value space· of types derived from **decimal** with a value for ·totalDigits· of *p* is the set of values *i* × *10*^-*n*, where *n* and *i* are integers such that *p* >= *n* >= *0* and the number of significant decimal digits in *i* is less than or equal to *p*.

[Definition:] The ·value space· of types derived from **decimal** with a value for ·fractionDigits· of *s* is the set of values *i* × *10*^-*n*, where *i* and *n* are integers such that *0* <= *n* <= *s*.

> **NOTE:** All ·minimally conforming· processors ·must· support decimal numbers with a minimum of 18 decimal digits (i.e., with a ·totalDigits· of 18). However, ·minimally conforming· processors ·may· set an application-defined limit on the maximum number of decimal digits they are prepared to support, in which case that application-defined maximum number ·must· be clearly documented.

3.2.3.1 Lexical representation

decimal has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39) separated by a period as a decimal indicator. If ·totalDigits· is specified, the number of digits must be less than or equal to ·totalDigits·. If ·fractionDigits· is specified, the number of digits following the decimal point must be less than or equal to the ·fractionDigits·. An optional leading sign is allowed. If the sign is omitted, "+" is assumed. Leading and trailing zeroes are optional. If the fractional part is zero, the period and following zero(es) can be omitted. For example: `-1.23, 12678967.543233, +100000.00, 210`.

3.2.3.2 Canonical representation

The canonical representation for **decimal** is defined by prohibiting certain options from the Lexical representation (§3.2.3.1). Specifically, the preceding optional "+" sign is prohibited. The decimal point is required. Leading and trailing zeroes are prohibited subject to the following: there must be at least one digit to the right and to the left of the decimal point which may be a zero.

3.2.3.3 Constraining facets

decimal has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.3.4 Derived datatypes

The following ·built-in· datatypes are ·derived· from **decimal**:

- integer

3.2.4 float

[Definition:] **float** corresponds to the IEEE single-precision 32-bit floating point type [IEEE 754-1985]. The basic ·value space· of **float** consists of the values *m* × *2^e*, where *m* is an integer whose absolute value is less than *2^24*, and *e* is an integer between -149 and 104, inclusive. In addition to the basic ·value space· described above, the ·value space· of **float** also contains the following *special values*: positive and negative zero, positive and negative infinity and not-a-number. The ·order-relation· on **float** is: *x* < *y iff y - x* is positive. Positive zero is greater than negative zero. Not-a-number equals itself and is greater than all float values including positive infinity.

A literal in the ·lexical space· representing a decimal number *d* maps to the normalized value in the ·value space· of **float** that is closest to *d* in the sense defined by [Clinger, WD (1990)]; if *d* is exactly halfway between two such values then the even value is chosen.

3.2.4.1 Lexical representation

float values have a lexical representation consisting of a mantissa followed, optionally, by the character "E" or "e", followed by an exponent. The exponent ·must· be an integer. The mantissa must be a decimal number. The representations for exponent and mantissa must follow the lexical rules for integer and decimal. If the "E" or "e" and the following exponent are omitted, an exponent value of 0 is assumed.

The *special values* positive and negative zero, positive and negative infinity and not-a-number have lexical representations `0`, `-0`, `INF`, `-INF` and `NaN`, respectively.

For example, `-1E4, 1267.43233E12, 12.78e-2, 12 and INF` are all legal literals for **float**.

3.2.4.2 Canonical representation

The canonical representation for **float** is defined by prohibiting certain options from the Lexical representation (§3.2.4.1). Specifically, the exponent must be indicated by "E". Leading zeroes and the preceding optional "+" sign are prohibited in the exponent. For the mantissa, the preceding optional "+" sign is prohibited and the decimal point is required. For the exponent, the preceding optional "+" sign is prohibited. Leading and trailing zeroes are prohibited subject to the following: number representations must be normalized such that there is a single digit to the left of the decimal point and at least a single digit to the right of the decimal point.

3.2.4.3 Constraining facets

float has the following ·constraining facets·:

- pattern
- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.5 double

[Definition:] The **double** datatype corresponds to IEEE double-precision 64-bit floating point type [IEEE 754-1985]. The basic ·value space· of **double** consists of the values *m* × *2^e*, where *m* is an integer whose absolute value is less than *2^53*, and *e* is an integer between -1075 and 970, inclusive. In addition to the basic ·value space· described above, the ·value space· of **double** also contains the following *special values*: positive and negative zero, positive and negative infinity and not-a-number. The ·order-relation· on **double** is: *x* < *y iff y - x* is positive. Positive zero is greater than negative zero. Not-a-number equals itself and is greater than all double values including positive infinity.

A literal in the ·lexical space· representing a decimal number *d* maps to the normalized value in the ·value space·

of **double** that is closest to *d*; if *d* is exactly halfway between two such values then the even value is chosen. This is the *best approximation* of *d* ([Clinger, WD (1990)], [Gay, DM (1990)]), which is more accurate than the mapping required by [IEEE 754-1985].

3.2.5.1 Lexical representation

double values have a lexical representation consisting of a mantissa followed, optionally, by the character "E" or "e", followed by an exponent. The exponent ·must· be an integer. The mantissa must be a decimal number. The representations for exponent and mantissa must follow the lexical rules for integer and decimal. If the "E" or "e" and the following exponent are omitted, an exponent value of 0 is assumed.

The *special values* positive and negative zero, positive and negative infinity and not-a-number have lexical representations `0`, `-0`, `INF`, `-INF` and `NaN`, respectively.

For example, `-1E4, 1267.43233E12, 12.78e-2, 12 and INF` are all legal literals for **double**.

3.2.5.2 Canonical representation

The canonical representation for **double** is defined by prohibiting certain options from the Lexical representation (§3.2.5.1). Specifically, the exponent must be indicated by "E". Leading zeroes and the preceding optional "+" sign are prohibited in the exponent. For the mantissa, the preceding optional "+" sign is prohibited and the decimal point is required. For the exponent, the preceding optional "+" sign is prohibited. Leading and trailing zeroes are prohibited subject to the following: number representations must be normalized such that there is a single digit to the left of the decimal point and at least a single digit to the right of the decimal point.

3.2.5.3 Constraining facets

double has the following ·constraining facets·:

- pattern
- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.6 duration

[Definition:] **duration** represents a duration of time. The ·value space· of **duration** is a six-dimensional space where the coordinates designate the Gregorian year, month, day, hour, minute, and second components defined in § 5.5.3.2 of [ISO 8601], respectively. These components are ordered in their significance by their order of appearance i.e. as year, month, day, hour, minute, and second.

3.2.6.1 Lexical representation

The lexical representation for **duration** is the [ISO 8601] extended format P*n*Y*n* M*n*DT*n*H *n*M*n*S, where *n*Y represents the number of years, *n*M the number of months, *n*D the number of days, 'T' is the date/time separator, *n*H the number of hours, *n*M the number of minutes and *n*S the number of seconds. The number of seconds can include decimal digits to arbitrary precision.

The values of the Year, Month, Day, Hour and Minutes components are not restricted but allow an arbitrary integer. Similarly, the value of the Seconds component allows an arbitrary decimal. Thus, the lexical representation of **duration** does not follow the alternative format of § 5.5.3.2.1 of [ISO 8601].

An optional preceding minus sign ('-') is allowed, to indicate a negative duration. If the sign is omitted a positive duration is indicated. See also ISO 8601 Date and Time Formats (§D).

For example, to indicate a duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes, one would write:

`P1Y2M3DT10H30M`. One could also indicate a duration of minus 120 days as: `-P120D`.

Reduced precision and truncated representations of this format are allowed provided they conform to the following:

- If the number of years, months, days, hours, minutes, or seconds in any expression equals zero, the number and its corresponding designator ·may· be omitted. However, at least one number and its designator ·must· be present.
- The seconds part ·may· have a decimal fraction.
- The designator 'T' shall be absent if all of the time items are absent. The designator 'P' must always be present.

For example, P1347Y, P1347M and P1Y2MT2H are all allowed; P0Y1347M and P0Y1347M0D are allowed. P-1347M is not allowed although -P1347M is allowed. P1Y2MT is not allowed.

3.2.6.2 Order relation on duration

In general, the ·order-relation· on **duration** is a partial order since there is no determinate relationship between certain durations such as one month (P1M) and 30 days (P30D). The ·order-relation· of two **duration** values *x* and *y* is *x* < *y iff s+x* < *s+y* for each qualified dateTime *s* in the list below. These values for *s* cause the greatest deviations in the addition of dateTimes and durations. Addition of durations to time instants is defined in Adding durations to dateTimes (§E).

- 1696-09-01T00:00:00Z
- 1697-02-01T00:00:00Z
- 1903-03-01T00:00:00Z
- 1903-07-01T00:00:00Z

The following table shows the strongest relationship that can be determined between example durations. The symbol <> means that the order relation is indeterminate. Note that because of leap-seconds, a seconds field can vary from 59 to 60. However, because of the way that addition is defined in Adding durations to dateTimes (§E), they are still totally ordered.

	Relation					
P**1Y**	> P**364D**	<> P**365D**			<> P**366D**	< P**367D**
P**1M**	> P**27D**	<> P**28D**	<> P**29D**	<> P**30D**	<> P**31D**	< P**32D**
P**5M**	> P**149D**	<> P**150D**	<> P**151D**	<> P**152D**	<> P**153D**	< P**154D**

Implementations are free to optimize the computation of the ordering relationship. For example, the following table can be used to compare durations of a small number of months against days.

	Months	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**...**
Days	**Minimum**	28	59	89	120	150	181	212	242	273	303	334	365	393	...
	Maximum	31	62	92	123	153	184	215	245	276	306	337	366	397	...

3.2.6.3 Facet Comparison for durations

In comparing **duration** values with minInclusive, minExclusive, maxInclusive and maxExclusive facet values indeterminate comparisons should be considered as "false".

3.2.6.4 Totally ordered durations

Certain derived datatypes of durations can be guaranteed have a total order. For this, they must have fields from only one row in the list below and the time zone must either be required or prohibited.

- year, month

- day, hour, minute, second

For example, a datatype could be defined to correspond to the [SQL] datatype Year-Month interval that required a four digit year field and a two digit month field but required all other fields to be unspecified. This datatype could be defined as below and would have a total order.

```
<simpleType name='SQL-Year-Month-Interval'>
    <restriction base='duration'>
      <pattern value='P\p{Nd}{4}Y\p{Nd}{2}M'/>
    </restriction>
</simpleType>
```

3.2.6.5 Constraining facets

duration has the following ·constraining facets·:

- pattern
- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.7 dateTime

[Definition:] **dateTime** represents a specific instant of time. The ·value space· of **dateTime** is the space of *Combinations of date and time of day* values as defined in § 5.4 of [ISO 8601].

3.2.7.1 Lexical representation

A single lexical representation, which is a subset of the lexical representations allowed by [ISO 8601], is allowed for **dateTime**. This lexical representation is the [ISO 8601] extended format CCYY-MM-DDThh:mm:ss where "CC" represents the century, "YY" the year, "MM" the month and "DD" the day, preceded by an optional leading "-" sign to indicate a negative number. If the sign is omitted, "+" is assumed. The letter "T" is the date/time separator and "hh", "mm", "ss" represent hour, minute and second respectively. Additional digits can be used to increase the precision of fractional seconds if desired i.e the format ss.ss... with any number of digits after the decimal point is supported. The fractional seconds part is optional; other parts of the lexical form are not optional. To accommodate year values greater than 9999 additional digits can be added to the left of this representation. Leading zeros are required if the year value would otherwise have fewer than four digits; otherwise they are forbidden. The year 0000 is prohibited.

The CCYY field must have at least four digits, the MM, DD, SS, hh, mm and ss fields exactly two digits each (not counting fractional seconds); leading zeroes must be used if the field would otherwise have too few digits.

This representation may be immediately followed by a "Z" to indicate Coordinated Universal Time (UTC) or, to indicate the time zone, i.e. the difference between the local time and Coordinated Universal Time, immediately followed by a sign, + or -, followed by the difference from UTC represented as hh:mm (note: the minutes part is required). See ISO 8601 Date and Time Formats (§D) for details about legal values in the various fields. If the time zone is included, both hours and minutes must be present.

For example, to indicate 1:20 pm on May the 31st, 1999 for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), one would write: `1999-05-31T13:20:00-05:00`.

3.2.7.2 Canonical representation

The canonical representation for **dateTime** is defined by prohibiting certain options from the Lexical representation (§3.2.7.1). Specifically, either the time zone must be omitted or, if present, the time zone must be Coordinated Universal Time (UTC) indicated by a "Z".

3.2.7.3 Order relation on dateTime

In general, the ·order-relation· on **dateTime** is a partial order since there is no determinate relationship between certain instants. For example, there is no determinate ordering between (a) 2000-01-20T12:00:00 and (b) 2000-01-20T12:00:00**Z**. Based on timezones currently in use, (c) could vary from 2000-01-20T12:00:00+12:00 to 2000-01-20T12:00:00-13:00. It is, however, possible for this range to expand or contract in the future, based on local laws. Because of this, the following definition uses a somewhat broader range of indeterminate values: +14:00..-14:00.

The following definition uses the notation S[year] to represent the year field of S, S[month] to represent the month field, and so on. The notation (Q & "-14:00") means adding the timezone -14:00 to Q, where Q did not already have a timezone. *This is a logical explanation of the process. Actual implementations are free to optimize as long as they produce the same results.*

The ordering between two **dateTime**s P and Q is defined by the following algorithm:

A.Normalize P and Q. That is, if there is a timezone present, but it is not Z, convert it to Z using the addition operation defined in Adding durations to dateTimes (§E)

- Thus 2000-03-04T23:00:00+03:00 normalizes to 2000-03-04T20:00:00Z

B. If P and Q either both have a time zone or both do not have a time zone, compare P and Q field by field from the year field down to the second field, and return a result as soon as it can be determined. That is:

1. For each i in {year, month, day, hour, minute, second}
    1. If P[i] and Q[i] are both not specified, continue to the next i
    2. If P[i] is not specified and Q[i] is, or vice versa, stop and return P <> Q
    3. If P[i] < Q[i], stop and return P < Q
    4. If P[i] > Q[i], stop and return P > Q
2. Stop and return P = Q

C.Otherwise, if P contains a time zone and Q does not, compare as follows:

1. P < Q if P < (Q with time zone +14:00)
2. P > Q if P > (Q with time zone -14:00)
3. P <> Q otherwise, that is, if (Q with time zone +14:00) < P < (Q with time zone -14:00)

D. Otherwise, if P does not contain a time zone and Q does, compare as follows:

1. P < Q if (P with time zone -14:00) < Q.
2. P > Q if (P with time zone +14:00) > Q.
3. P <> Q otherwise, that is, if (P with time zone +14:00) < Q < (P with time zone -14:00)

Examples:

Determinate	Indeterminate
2000-01-15T00:00:00 **<** 2000-02-15T00:00:00	2000-01-01T12:00:00 **<>** 1999-12-31T23:00:00Z
2000-01-15T12:00:00 **<** 2000-01-16T12:00:00Z	2000-01-16T12:00:00 **<>** 2000-01-16T12:00:00Z
	2000-01-16T00:00:00 **<>** 2000-01-16T12:00:00Z

3.2.7.4 Totally ordered dateTimes

Certain derived types from **dateTime** can be guaranteed have a total order. To do so, they must require that a specific set of fields are always specified, and that remaining fields (if any) are always unspecified. For example, the date datatype without time zone is defined to contain exactly year, month, and day. Thus dates without time zone have a total order among themselves.

3.2.7.5 Constraining facets

dateTime has the following ·constraining facets·:

- pattern
- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.8 time

[Definition:] **time** represents an instant of time that recurs every day. The ·value space· of **time** is the space of *time of day* values as defined in § 5.3 of [ISO 8601]. Specifically, it is a set of zero-duration daily time instances.

Since the lexical representation allows an optional time zone indicator, **time** values are partially ordered because it may not be able to determine the order of two values one of which has a time zone and the other does not. The order relation on **time** values is the Order relation on dateTime (§3.2.7.3) using an arbitrary date. See also Adding durations to dateTimes (§E). Pairs of **time** values with or without time zone indicators are totally ordered.

3.2.8.1 Lexical representation

The lexical representation for **time** is the left truncated lexical representation for dateTime: hh:mm:ss.sss with optional following time zone indicator. For example, to indicate 1:20 pm for Eastern Standard Time which is 5 hours behind Coordinated Universal Time (UTC), one would write: 13:20:00-05:00. See also ISO 8601 Date and Time Formats (§D).

3.2.8.2 Canonical representation

The canonical representation for **time** is defined by prohibiting certain options from the Lexical representation (§3.2.8.1). Specifically, either the time zone must be omitted or, if present, the time zone must be Coordinated Universal Time (UTC) indicated by a "Z". Additionally, the canonical representation for midnight is 00:00:00.

3.2.8.3 Constraining facets

time has the following ·constraining facets·:

- pattern
- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.9 date

[Definition:] **date** represents a calendar date. The ·value space· of **date** is the set of Gregorian calendar dates as defined in § 5.2.1 of [ISO 8601]. Specifically, it is a set of one-day long, non-periodic instances e.g. lexical 1999-10-26 to represent the calendar date 1999-10-26, independent of how many hours this day has.

Since the lexical representation allows an optional time zone indicator, **date** values are partially ordered because it may not be possible to unequivocally determine the order of two values one of which has a time zone and the other does not. If **date** values are considered as periods of time, the order relation on **date** values is the order relation on their starting instants. This is discussed in Order relation on dateTime (§3.2.7.3). See also Adding durations to dateTimes (§E). Pairs of **date** values with or without time zone indicators are totally ordered.

3.2.9.1 Lexical representation

The lexical representation for **date** is the reduced (right truncated) lexical representation for dateTime: CCYY-MM-DD. No left truncation is allowed. An optional following time zone qualifier is allowed as for dateTime. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" sign is allowed.

For example, to indicate May the 31st, 1999, one would write: 1999-05-31. See also ISO 8601 Date and Time Formats (§D).

3.2.9.2 Constraining facets

date has the following ·constraining facets·:

- pattern
- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.10 gYearMonth

[Definition:] **gYearMonth** represents a specific gregorian month in a specific gregorian year. The ·value space· of **gYearMonth** is the set of Gregorian calendar months as defined in § 5.2.1 of [ISO 8601]. Specifically, it is a set of one-month long, non-periodic instances e.g. 1999-10 to represent the whole month of 1999-10, independent of how many days this month has.

Since the lexical representation allows an optional time zone indicator, **gYearMonth** values are partially ordered because it may not be possible to unequivocally determine the order of two values one of which has a time zone and the other does not. If **gYearMonth** values are considered as periods of time, the order relation on **gYearMonth** values is the order relation on their starting instants. This is discussed in Order relation on dateTime (§3.2.7.3). See also Adding durations to dateTimes (§E). Pairs of **gYearMonth** values with or without time zone indicators are totally ordered.

> **NOTE:** Because month/year combinations in one calendar only rarely correspond to month/year combinations in other calendars, values of this type are not, in general, convertible to simple values corresponding to month/year combinations in other calendars. This type should therefore be used with caution in contexts where conversion to other calendars is desired.

3.2.10.1 Lexical representation

The lexical representation for **gYearMonth** is the reduced (right truncated) lexical representation for dateTime: CCYY-MM. No left truncation is allowed. An optional following time zone qualifier is allowed. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" sign is allowed.

For example, to indicate the month of May 1999, one would write: 1999-05. See also ISO 8601 Date and Time Formats (§D).

3.2.10.2 Constraining facets

gYearMonth has the following ·constraining facets·:

- pattern
- enumeration
- whiteSpace

- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.11 gYear

[Definition:] **gYear** represents a gregorian calendar year. The ·value space· of **gYear** is the set of Gregorian calendar years as defined in § 5.2.1 of [ISO 8601]. Specifically, it is a set of one-year long, non-periodic instances e.g. lexical 1999 to represent the whole year 1999, independent of how many months and days this year has.

Since the lexical representation allows an optional time zone indicator, **gYear** values are partially ordered because it may not be possible to unequivocally determine the order of two values one of which has a time zone and the other does not. If **gYear** values are considered as periods of time, the order relation on **gYear** values is the order relation on their starting instants. This is discussed in Order relation on dateTime (§3.2.7.3). See also Adding durations to dateTimes (§E). Pairs of **gYear** values with or without time zone indicators are totally ordered.

> **NOTE:** Because years in one calendar only rarely correspond to years in other calendars, values of this type are not, in general, convertible to simple values corresponding to years in other calendars. This type should therefore be used with caution in contexts where conversion to other calendars is desired.

3.2.11.1 Lexical representation

The lexical representation for **gYear** is the reduced (right truncated) lexical representation for dateTime: CCYY. No left truncation is allowed. An optional following time zone qualifier is allowed as for dateTime. To accommodate year values outside the range from 0001 to 9999, additional digits can be added to the left of this representation and a preceding "-" sign is allowed.

For example, to indicate 1999, one would write: 1999. See also ISO 8601 Date and Time Formats (§D).

3.2.11.2 Constraining facets

gYear has the following ·constraining facets·:

- pattern
- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.12 gMonthDay

[Definition:] **gMonthDay** is a gregorian date that recurs, specifically a day of the year such as the third of May. Arbitrary recurring dates are not supported by this datatype. The ·value space· of **gMonthDay** is the set of *calendar dates*, as defined in § 3 of [ISO 8601]. Specifically, it is a set of one-day long, annually periodic instances.

Since the lexical representation allows an optional time zone indicator, **gMonthDay** values are partially ordered because it may not be possible to unequivocally determine the order of two values one of which has a time zone and the other does not. If **gMonthDay** values are considered as periods of time, the order relation on **gMonthDay** values is the order relation on their starting instants. This is discussed in Order relation on dateTime (§3.2.7.3). See also Adding durations to dateTimes (§E). Pairs of **gMonthDay** values with or without time zone indicators are totally ordered.

> **NOTE:** Because day/month combinations in one calendar only rarely correspond to day/month combinations in other calendars, values of this type do not, in general, have any straightforward or intuitive representation in terms of most other calendars. This type should therefore be used with caution in contexts where conversion to other calendars is desired.

3.2.12.1 Lexical representation

The lexical representation for **gMonthDay** is the left truncated lexical representation for date: --MM-DD. An optional following time zone qualifier is allowed as for date. No preceding sign is allowed. No other formats are allowed. See also ISO 8601 Date and Time Formats (§D).

This datatype can be used to represent a specific day in a month. To say, for example, that my birthday occurs on the 14th of September ever year.

3.2.12.2 Constraining facets

gMonthDay has the following ·constraining facets·:

- pattern
- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.13 gDay

[Definition:] **gDay** is a gregorian day that recurs, specifically a day of the month such as the 5th of the month. Arbitrary recurring days are not supported by this datatype. The ·value space· of **gDay** is the space of a set of *calendar dates* as defined in § 3 of [ISO 8601]. Specifically, it is a set of one-day long, monthly periodic instances.

This datatype can be used to represent a specific day of the month. To say, for example, that I get my paycheck on the 15th of each month.

Since the lexical representation allows an optional time zone indicator, **gDay** values are partially ordered because it may not be possible to unequivocally determine the order of two values one of which has a time zone and the other does not. If **gDay** values are considered as periods of time, the order relation on **gDay** values is the order relation on their starting instants. This is discussed in Order relation on dateTime (§3.2.7.3). See also Adding durations to dateTimes (§E). Pairs of **gDay** values with or without time zone indicators are totally ordered.

> **NOTE:** Because days in one calendar only rarely correspond to days in other calendars, values of this type do not, in general, have any straightforward or intuitive representation in terms of most other calendars. This type should therefore be used with caution in contexts where conversion to other calendars is desired.

3.2.13.1 Lexical representation

The lexical representation for **gDay** is the left truncated lexical representation for date: ---DD . An optional following time zone qualifier is allowed as for date. No preceding sign is allowed. No other formats are allowed. See also ISO 8601 Date and Time Formats (§D).

3.2.13.2 Constraining facets

gDay has the following ·constraining facets·:

- pattern

- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.14 gMonth

[Definition:] **gMonth** is a gregorian month that recurs every year. The ·value space· of **gMonth** is the space of a set of *calendar months* as defined in § 3 of [ISO 8601]. Specifically, it is a set of one-month long, yearly periodic instances.

This datatype can be used to represent a specific month. To say, for example, that Thanksgiving falls in the month of November.

Since the lexical representation allows an optional time zone indicator, **gMonth** values are partially ordered because it may not be possible to unequivocally determine the order of two values one of which has a time zone and the other does not. If **gMonth** values are considered as periods of time, the order relation on **gMonth** is the order relation on their starting instants. This is discussed in Order relation on dateTime (§3.2.7.3). See also Adding durations to dateTimes (§E). Pairs of **gMonth** values with or without time zone indicators are totally ordered.

> **NOTE:** Because months in one calendar only rarely correspond to months in other calendars, values of this type do not, in general, have any straightforward or intuitive representation in terms of most other calendars. This type should therefore be used with caution in contexts where conversion to other calendars is desired.

3.2.14.1 Lexical representation

The lexical representation for **gMonth** is the left and right truncated lexical representation for date: --MM--. An optional following time zone qualifier is allowed as for date. No preceding sign is allowed. No other formats are allowed. See also ISO 8601 Date and Time Formats (§D).

3.2.14.2 Constraining facets

gMonth has the following ·constraining facets·:

- pattern
- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.2.15 hexBinary

[Definition:] **hexBinary** represents arbitrary hex-encoded binary data. The ·value space· of **hexBinary** is the set of finite-length sequences of binary octets.

3.2.15.1 Lexical Representation

hexBinary has a lexical representation where each binary octet is encoded as a character tuple, consisting of two hexadecimal digits ([0-9a-fA-F]) representing the octet code. For example, "0FB7" is a *hex* encoding for the 16-bit integer 4023 (whose binary representation is 111110110111).

3.2.15.2 Canonical Rrepresentation

The canonical representation for **hexBinary** is defined by prohibiting certain options from the Lexical Representation (§3.2.15.1). Specifically, the lower case hexadecimal digits ([a-f]) are not allowed.

3.2.15.3 Constraining facets

hexBinary has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.2.16 base64Binary

[Definition:] **base64Binary** represents Base64-encoded arbitrary binary data. The ·value space· of **base64Binary** is the set of finite-length sequences of binary octets. For **base64Binary** data the entire binary stream is encoded using the Base64 Content-Transfer-Encoding defined in Section 6.8 of [RFC 2045].

3.2.16.1 Constraining facets

base64Binary has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.2.17 anyURI

[Definition:] **anyURI** represents a Uniform Resource Identifier Reference (URI). An **anyURI** value can be absolute or relative, and may have an optional fragment identifier (i.e., it may be a URI Reference). This type should be used to specify the intention that the value fulfills the role of a URI as defined by [RFC 2396], as amended by [RFC 2732].

The mapping from **anyURI** values to URIs is as defined in Section 5.4 Locator Attribute of [XML Linking Language] (see also Section 8 Character Encoding in URI References of [Character Model]). This means that a wide range of internationalized resource identifiers can be specified when an **anyURI** is called for, and still be understood as URIs per [RFC 2396], as amended by [RFC 2732], where appropriate to identify resources.

> **NOTE:** Each URI scheme imposes specialized syntax rules for URIs in that scheme, including restrictions on the syntax of allowed fragement identifiers. Because it is impractical for processors to check that a value is a context-appropriate URI reference, this specification follows the lead of [RFC 2396] (as amended by [RFC 2732]) in this matter: such rules and restrictions are not part of type validity and are not checked by ·minimally conforming· processors. Thus in practice the above definition imposes only very modest obligations on ·minimally conforming· processors.

3.2.17.1 Lexical representation

The ·lexical space· of **anyURI** is finite-length character sequences which, when the algorithm defined in Section 5.4 of [XML Linking Language] is applied to them, result in strings which are legal URIs according to [RFC 2396], as amended by [RFC 2732].

> **NOTE:** Spaces are, in principle, allowed in the ·lexical space· of **anyURI**, however, their use is highly discouraged (unless they are encoded by %20).

3.2.17.2 Constraining facets

anyURI has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.2.18 QName

[Definition:] **QName** represents XML qualified names. The ·value space· of **QName** is the set of tuples {namespace name, local part}, where namespace name is an anyURI and local part is an NCName. The ·lexical space· of **QName** is the set of strings that ·match· the QName production of [Namespaces in XML].

> **NOTE:** The mapping between literals in the ·lexical space· and values in the ·value space· of **QName** requires a namespace declaration to be in scope for the context in which **QName** is used.

3.2.18.1 Constraining facets

QName has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.2.19 NOTATION

[Definition:] **NOTATION** represents the NOTATION attribute type from [XML 1.0 (Second Edition)]. The ·value space· of **NOTATION** is the set QNames. The ·lexical space· of **NOTATION** is the set of all names of notations declared in the current schema.

Schema Component Constraint: enumeration facet value required for NOTATION
It is an ·error· for **NOTATION** to be used directly in a schema. Only datatypes that are ·derived· from **NOTATION** by specifying a value for ·enumeration· can be used in a schema.

For compatibility (see Terminology (§1.4)) **NOTATION** should be used only on attributes.

3.2.19.1 Constraining facets

NOTATION has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.3 Derived datatypes

3.3.1 normalizedString

This section gives conceptual definitions for all ·built-in· ·derived· datatypes defined by this specification. The XML representation used to define ·derived· datatypes (whether ·built-in· or ·user-derived·) is given in section XML Representation of Simple Type Definition Schema Components (§4.1.2) and the complete definitions of the ·built-in· ·derived· datatypes are provided in Appendix ASchema for Datatype Definitions (normative) (§A).

3.3.1 normalizedString

[Definition:] **normalizedString** represents white space normalized strings. The ·value space· of **normalizedString** is the set of strings that do not contain the carriage return (#xD), line feed (#xA) nor tab (#x9) characters. The ·lexical space· of **normalizedString** is the set of strings that do not contain the carriage return (#xD) nor tab (#x9) characters. The ·base type· of **normalizedString** is string.

3.3.1.1 Constraining facets

normalizedString has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.3.1.2 Derived datatypes

The following ·built-in· datatypes are ·derived· from **normalizedString**:

- token

3.3.2 token

[Definition:] **token** represents tokenized strings. The ·value space· of **token** is the set of strings that do not

contain the line feed (#xA) nor tab (#x9) characters, that have no leading or trailing spaces (#x20) and that have no internal sequences of two or more spaces. The ·lexical space· of **token** is the set of strings that do not contain the line feed (#xA) nor tab (#x9) characters, that have no leading or trailing spaces (#x20) and that have no internal sequences of two or more spaces. The ·base type· of **token** is normalizedString.

3.3.2.1 Constraining facets

token has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.3.2.2 Derived datatypes

The following ·built-in· datatypes are ·derived· from **token**:

- language
- NMTOKEN
- Name

3.3.3 language

[Definition:] **language** represents natural language identifiers as defined by [RFC 1766]. The ·value space· of **language** is the set of all strings that are valid language identifiers as defined in the language identification section of [XML 1.0 (Second Edition)]. The ·lexical space· of **language** is the set of all strings that are valid language identifiers as defined in the language identification section of [XML 1.0 (Second Edition)]. The ·base type· of **language** is token.

3.3.3.1 Constraining facets

language has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.3.4 NMTOKEN

[Definition:] **NMTOKEN** represents the NMTOKEN attribute type from [XML 1.0 (Second Edition)]. The ·value space· of **NMTOKEN** is the set of tokens that ·match· the Nmtoken production in [XML 1.0 (Second Edition)]. The ·lexical space· of **NMTOKEN** is the set of strings that ·match· the Nmtoken production in [XML 1.0 (Second Edition)]. The ·base type· of **NMTOKEN** is token.

For compatibility (see Terminology (§1.4)) **NMTOKEN** should be used only on attributes.

3.3.4.1 Constraining facets

NMTOKEN has the following ·constraining facets·:

- length

- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.3.4.2 Derived datatypes

The following ·built-in· datatypes are ·derived· from **NMTOKEN**:

- NMTOKENS

3.3.5 NMTOKENS

[Definition:] **NMTOKENS** represents the NMTOKENS attribute type from [XML 1.0 (Second Edition)]. The ·value space· of **NMTOKENS** is the set of finite, non-zero-length sequences of ·NMTOKEN·s. The ·lexical space· of **NMTOKENS** is the set of white space separated lists of tokens, of which each token is in the ·lexical space· of NMTOKEN. The ·itemType· of **NMTOKENS** is NMTOKEN.

For compatibility (see Terminology (§1.4)) **NMTOKENS** should be used only on attributes.

3.3.5.1 Constraining facets

NMTOKENS has the following ·constraining facets·:

- length
- minLength
- maxLength
- enumeration
- whiteSpace

3.3.6 Name

[Definition:] **Name** represents XML Names. The ·value space· of **Name** is the set of all strings which ·match· the Name production of [XML 1.0 (Second Edition)]. The ·lexical space· of **Name** is the set of all strings which ·match· the Name production of [XML 1.0 (Second Edition)]. The ·base type· of **Name** is token.

3.3.6.1 Constraining facets

Name has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.3.6.2 Derived datatypes

The following ·built-in· datatypes are ·derived· from **Name**:

- NCName

3.3.7 NCName

[Definition:] **NCName** represents XML "non-colonized" Names. The ·value space· of **NCName** is the set of all

strings which ·match· the NCName production of [Namespaces in XML]. The ·lexical space· of **NCName** is the set of all strings which ·match· the NCName production of [Namespaces in XML]. The ·base type· of **NCName** is Name.

3.3.7.1 Constraining facets

NCName has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.3.7.2 Derived datatypes

The following ·built-in· datatypes are ·derived· from **NCName**:

- ID
- IDREF
- ENTITY

3.3.8 ID

[Definition:] **ID** represents the ID attribute type from [XML 1.0 (Second Edition)]. The ·value space· of **ID** is the set of all strings that ·match· the NCName production in [Namespaces in XML]. The ·lexical space· of **ID** is the set of all strings that ·match· the NCName production in [Namespaces in XML]. The ·base type· of **ID** is NCName.

For compatibility (see Terminology (§1.4)) **ID** should be used only on attributes.

3.3.8.1 Constraining facets

ID has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.3.9 IDREF

[Definition:] **IDREF** represents the IDREF attribute type from [XML 1.0 (Second Edition)]. The ·value space· of **IDREF** is the set of all strings that ·match· the NCName production in [Namespaces in XML]. The ·lexical space· of **IDREF** is the set of strings that ·match· the NCName production in [Namespaces in XML]. The ·base type· of **IDREF** is NCName.

For compatibility (see Terminology (§1.4)) this datatype should be used only on attributes.

3.3.9.1 Constraining facets

IDREF has the following ·constraining facets·:

- length

- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.3.9.2 Derived datatypes

The following ·built-in· datatypes are ·derived· from **IDREF**:

- IDREFS

3.3.10 IDREFS

[Definition:] **IDREFS** represents the IDREFS attribute type from [XML 1.0 (Second Edition)]. The ·value space· of **IDREFS** is the set of finite, non-zero-length sequences of IDREFs. The ·lexical space· of **IDREFS** is the set of white space separated lists of tokens, of which each token is in the ·lexical space· of IDREF. The ·itemType· of **IDREFS** is IDREF.

For compatibility (see Terminology (§1.4)) **IDREFS** should be used only on attributes.

3.3.10.1 Constraining facets

IDREFS has the following ·constraining facets·:

- length
- minLength
- maxLength
- enumeration
- whiteSpace

3.3.11 ENTITY

[Definition:] **ENTITY** represents the ENTITY attribute type from [XML 1.0 (Second Edition)]. The ·value space· of **ENTITY** is the set of all strings that ·match· the NCName production in [Namespaces in XML] and have been declared as an unparsed entity in a document type definition. The ·lexical space· of **ENTITY** is the set of all strings that ·match· the NCName production in [Namespaces in XML]. The ·base type· of **ENTITY** is NCName.

> **NOTE:** The ·value space· of **ENTITY** is scoped to a specific instance document.

For compatibility (see Terminology (§1.4)) **ENTITY** should be used only on attributes.

3.3.11.1 Constraining facets

ENTITY has the following ·constraining facets·:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace

3.3.11.2 Derived datatypes

The following ·built-in· datatypes are ·derived· from **ENTITY**:

- ENTITIES

3.3.12 ENTITIES

[Definition:] **ENTITIES** represents the ENTITIES attribute type from [XML 1.0 (Second Edition)]. The ·value space· of **ENTITIES** is the set of finite, non-zero-length sequences of ·ENTITY·s that have been declared as unparsed entities in a document type definition. The ·lexical space· of **ENTITIES** is the set of white space separated lists of tokens, of which each token is in the ·lexical space· of ENTITY. The ·itemType· of **ENTITIES** is ENTITY.

> **NOTE:** The ·value space· of **ENTITIES** is scoped to a specific instance document.

For compatibility (see Terminology (§1.4)) **ENTITIES** should be used only on attributes.

3.3.12.1 Constraining facets

ENTITIES has the following ·constraining facets·:

- length
- minLength
- maxLength
- enumeration
- whiteSpace

3.3.13 integer

[Definition:] **integer** is ·derived· from decimal by fixing the value of ·fractionDigits· to be 0. This results in the standard mathematical concept of the integer numbers. The ·value space· of **integer** is the infinite set {...,-2,-1,0,1,2,...}. The ·base type· of **integer** is decimal.

3.3.13.1 Lexical representation

integer has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39) with an optional leading sign. If the sign is omitted, "+" is assumed. For example: -1, 0, 12678967543233, +100000.

3.3.13.2 Canonical representation

The canonical representation for **integer** is defined by prohibiting certain options from the Lexical representation (§3.3.13.1). Specifically, the preceding optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.13.3 Constraining facets

integer has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.13.4 Derived datatypes

The following ·built-in· datatypes are ·derived· from **integer**:

- nonPositiveInteger
- long
- nonNegativeInteger

3.3.14 nonPositiveInteger

[Definition:] **nonPositiveInteger** is ·derived· from integer by setting the value of ·maxInclusive· to be 0. This results in the standard mathematical concept of the non-positive integers. The ·value space· of **nonPositiveInteger** is the infinite set {...,-2,-1,0}. The ·base type· of **nonPositiveInteger** is integer.

3.3.14.1 Lexical representation

nonPositiveInteger has a lexical representation consisting of a negative sign ("-") followed by a finite-length sequence of decimal digits (#x30-#x39). If the sequence of digits consists of all zeros then the sign is optional. For example: -1, 0, -12678967543233, -100000.

3.3.14.2 Canonical representation

The canonical representation for **nonPositiveInteger** is defined by prohibiting certain options from the Lexical representation (§3.3.14.1). Specifically, the negative sign ("-") is required with the token "0" and leading zeroes are prohibited.

3.3.14.3 Constraining facets

nonPositiveInteger has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.14.4 Derived datatypes

The following ·built-in· datatypes are ·derived· from **nonPositiveInteger**:

- negativeInteger

3.3.15 negativeInteger

[Definition:] **negativeInteger** is ·derived· from nonPositiveInteger by setting the value of ·maxInclusive· to be -1. This results in the standard mathematical concept of the negative integers. The ·value space· of **negativeInteger** is the infinite set {...,-2,-1}. The ·base type· of **negativeInteger** is nonPositiveInteger.

3.3.15.1 Lexical representation

negativeInteger has a lexical representation consisting of a negative sign ("-") followed by a finite-length sequence of decimal digits (#x30-#x39). For example: -1, -12678967543233, -100000.

3.3.15.2 Canonical representation

The canonical representation for **negativeInteger** is defined by prohibiting certain options from the Lexical representation (§3.3.15.1). Specifically, leading zeroes are prohibited.

3.3.15.3 Constraining facets

negativeInteger has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.16 long

[Definition:] **long** is ·derived· from integer by setting the value of ·maxInclusive· to be 9223372036854775807 and ·minInclusive· to be -9223372036854775808. The ·base type· of **long** is integer.

3.3.16.1 Lexical representation

long has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits (#x30-#x39). If the sign is omitted, "+" is assumed. For example: -1, 0, 12678967543233, +100000.

3.3.16.2 Canonical representation

The canonical representation for **long** is defined by prohibiting certain options from the Lexical representation (§3.3.16.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.16.3 Constraining facets

long has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.16.4 Derived datatypes

The following ·built-in· datatypes are ·derived· from **long**:

- int

3.3.17 int

[Definition:] **int** is ·derived· from long by setting the value of ·maxInclusive· to be 2147483647 and ·minInclusive· to be -2147483648. The ·base type· of **int** is long.

3.3.17.1 Lexical representation

int has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal

digits (#x30-#x39). If the sign is omitted, "+" is assumed. For example: -1, 0, 126789675, +100000.

3.3.17.2 Canonical representation

The canonical representation for **int** is defined by prohibiting certain options from the Lexical representation (§3.3.17.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.17.3 Constraining facets

int has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.17.4 Derived datatypes

The following ·built-in· datatypes are ·derived· from **int**:

- short

3.3.18 short

[Definition:] **short** is ·derived· from int by setting the value of ·maxInclusive· to be 32767 and ·minInclusive· to be -32768. The ·base type· of **short** is int.

3.3.18.1 Lexical representation

short has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits (#x30-#x39). If the sign is omitted, "+" is assumed. For example: -1, 0, 12678, +10000.

3.3.18.2 Canonical representation

The canonical representation for **short** is defined by prohibiting certain options from the Lexical representation (§3.3.18.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.18.3 Constraining facets

short has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.18.4 Derived datatypes

The following ·built-in· datatypes are ·derived· from **short**:

- byte

3.3.19 byte

[Definition:] **byte** is ·derived· from short by setting the value of ·maxInclusive· to be 127 and ·minInclusive· to be -128. The ·base type· of **byte** is short.

3.3.19.1 Lexical representation

byte has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits (#x30-#x39). If the sign is omitted, "+" is assumed. For example: -1, 0, 126, +100.

3.3.19.2 Canonical representation

The canonical representation for **byte** is defined by prohibiting certain options from the Lexical representation (§3.3.19.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.19.3 Constraining facets

byte has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.20 nonNegativeInteger

[Definition:] **nonNegativeInteger** is ·derived· from integer by setting the value of ·minInclusive· to be 0. This results in the standard mathematical concept of the non-negative integers. The ·value space· of **nonNegativeInteger** is the infinite set {0,1,2,...}. The ·base type· of **nonNegativeInteger** is integer.

3.3.20.1 Lexical representation

nonNegativeInteger has a lexical representation consisting of an optional sign followed by a finite-length sequence of decimal digits (#x30-#x39). If the sign is omitted, "+" is assumed. For example: 1, 0, 12678967543233, +100000.

3.3.20.2 Canonical representation

The canonical representation for **nonNegativeInteger** is defined by prohibiting certain options from the Lexical representation (§3.3.20.1). Specifically, the the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.20.3 Constraining facets

nonNegativeInteger has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace

- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.20.4 Derived datatypes

The following ·built-in· datatypes are ·derived· from **nonNegativeInteger**:

- unsignedLong
- positiveInteger

3.3.21 unsignedLong

[Definition:] **unsignedLong** is ·derived· from nonNegativeInteger by setting the value of ·maxInclusive· to be 18446744073709551615. The ·base type· of **unsignedLong** is nonNegativeInteger.

3.3.21.1 Lexical representation

unsignedLong has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39). For example: 0, 12678967543233, 100000.

3.3.21.2 Canonical representation

The canonical representation for **unsignedLong** is defined by prohibiting certain options from the Lexical representation (§3.3.21.1). Specifically, leading zeroes are prohibited.

3.3.21.3 Constraining facets

unsignedLong has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.21.4 Derived datatypes

The following ·built-in· datatypes are ·derived· from **unsignedLong**:

- unsignedInt

3.3.22 unsignedInt

[Definition:] **unsignedInt** is ·derived· from unsignedLong by setting the value of ·maxInclusive· to be 4294967295. The ·base type· of **unsignedInt** is unsignedLong.

3.3.22.1 Lexical representation

unsignedInt has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39). For example: 0, 1267896754, 100000.

3.3.22.2 Canonical representation

The canonical representation for **unsignedInt** is defined by prohibiting certain options from the Lexical representation (§3.3.22.1). Specifically, leading zeroes are prohibited.

3.3.22.3 Constraining facets

unsignedInt has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.22.4 Derived datatypes

The following ·built-in· datatypes are ·derived· from **unsignedInt**:

- unsignedShort

3.3.23 unsignedShort

[Definition:] **unsignedShort** is ·derived· from unsignedInt by setting the value of ·maxInclusive· to be 65535. The ·base type· of **unsignedShort** is unsignedInt.

3.3.23.1 Lexical representation

unsignedShort has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39). For example: 0, 12678, 10000.

3.3.23.2 Canonical representation

The canonical representation for **unsignedShort** is defined by prohibiting certain options from the Lexical representation (§3.3.23.1). Specifically, the leading zeroes are prohibited.

3.3.23.3 Constraining facets

unsignedShort has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.23.4 Derived datatypes

The following ·built-in· datatypes are ·derived· from **unsignedShort**:

- unsignedByte

3.3.24 unsignedByte

[Definition:] **unsignedByte** is ·derived· from unsignedShort by setting the value of ·maxInclusive· to be 255. The ·base type· of **unsignedByte** is unsignedShort.

3.3.24.1 Lexical representation

unsignedByte has a lexical representation consisting of a finite-length sequence of decimal digits (#x30-#x39). For example: 0, 126, 100.

3.3.24.2 Canonical representation

The canonical representation for **unsignedByte** is defined by prohibiting certain options from the Lexical representation (§3.3.24.1). Specifically, leading zeroes are prohibited.

3.3.24.3 Constraining facets

unsignedByte has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive

3.3.25 positiveInteger

[Definition:] **positiveInteger** is ·derived· from nonNegativeInteger by setting the value of ·minInclusive· to be 1. This results in the standard mathematical concept of the positive integer numbers. The ·value space· of **positiveInteger** is the infinite set {1,2,...}. The ·base type· of **positiveInteger** is nonNegativeInteger.

3.3.25.1 Lexical representation

positiveInteger has a lexical representation consisting of an optional positive sign ("+") followed by a finite-length sequence of decimal digits (#x30-#x39). For example: 1, 12678967543233, +100000.

3.3.25.2 Canonical representation

The canonical representation for **positiveInteger** is defined by prohibiting certain options from the Lexical representation (§3.3.25.1). Specifically, the optional "+" sign is prohibited and leading zeroes are prohibited.

3.3.25.3 Constraining facets

positiveInteger has the following ·constraining facets·:

- totalDigits
- fractionDigits
- pattern
- whiteSpace
- enumeration
- maxInclusive

- maxExclusive
- minInclusive
- minExclusive

4 Datatype components

The following sections provide full details on the properties and significance of each kind of schema component involved in datatype definitions. For each property, the kinds of values it is allowed to have is specified. Any property not identified as optional is required to be present; optional properties which are not present have absent as their value. Any property identified as a having a set, subset or ·list· value may have an empty value unless this is explicitly ruled out: this is not the same as absent. Any property value identified as a superset or a subset of some set may be equal to that set, unless a proper superset or subset is explicitly called for.

For more information on the notion of datatype (schema) components, see Schema Component Details of [XML Schema Part 1: Structures].

4.1 Simple Type Definition

Simple Type definitions provide for:

- Establishing the ·value space· and ·lexical space· of a datatype, through the combined set of ·constraining facet·s specified in the definition;
- Attaching a unique name (actually a QName) to the ·value space· and ·lexical space·.

4.1.1 The Simple Type Definition Schema Component

The Simple Type Definition schema component has the following properties:

Schema Component: Simple Type Definition

{name}
Optional. An NCName as defined by [Namespaces in XML].

{target namespace}
Either absent or a namespace name, as defined in [Namespaces in XML].

{variety}
One of {*atomic*, *list*, *union*}. Depending on the value of {variety}, further properties are defined as follows:

atomic

{primitive type definition}
A ·built-in· ·primitive· datatype definition (or the simple ur-type definition).

list

{item type definition}
An ·atomic· or ·union· simple type definition.

union

{member type definitions}
A non-empty sequence of simple type definitions.

{facets}
A possibly empty set of Facets (§2.4).

{fundamental facets}
A set of Fundamental facets (§2.4.1)

{base type definition}

If the datatype has been ·derived· by ·restriction· then the Simple Type Definition component from which it is ·derived·, otherwise the Simple Type Definition for anySimpleType (§4.1.6).

{final}
A subset of *{restriction, list, union}*.

{annotation}
Optional. An annotation.

Datatypes are identified by their {name} and {target namespace}. Except for anonymous datatypes (those with no {name}), datatype definitions ·must· be uniquely identified within a schema.

If {variety} is ·atomic· then the ·value space· of the datatype defined will be a subset of the ·value space· of {base type definition} (which is a subset of the ·value space· of {primitive type definition}). If {variety} is ·list· then the ·value space· of the datatype defined will be the set of finite-length sequence of values from the ·value space· of {item type definition}. If {variety} is ·union· then the ·value space· of the datatype defined will be the union of the ·value space·s of each datatype in {member type definitions}.

If {variety} is ·atomic· then the {variety} of {base type definition} must be ·atomic·. If {variety} is ·list· then the {variety} of {item type definition} must be either ·atomic· or ·union·. If {variety} is ·union· then {member type definitions} must be a list of datatype definitions.

The value of {facets} consists of the set of ·facet·s specified directly in the datatype definition unioned with the possibly empty set of {facets} of {base type definition}.

The value of {fundamental facets} consists of the set of ·fundamental facet·s and their values.

If {final} is the empty set then the type can be used in deriving other types; the explicit values *restriction*, *list* and *union* prevent further derivations by ·restriction·, ·list· and ·union· respectively.

4.1.2 XML Representation of Simple Type Definition Schema Components

The XML representation for a Simple Type Definition schema component is a <simpleType> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `simpleType` Element Information Item

```
<simpleType
  final = (#all | (list | union | restriction))
  id = ID
  name = NCName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (restriction | list | union))
</simpleType>
```

Datatype Definition Schema Component

Property	Representation
{name}	The actual value of the `name` [attribute], if present, otherwise null

{final}	A set corresponding to the actual value of the `final` [attribute], if present, otherwise of the actual value of the `finalDefault` [attribute] the ancestor schema element information item, if present, otherwise the empty string, as follows: **the empty string** the empty set; **`#all`** *{restriction, list, union}*; ***otherwise*** a set with members drawn from the set above, each being present or absent depending on whether the string contains an equivalently named space-delimited substring. **NOTE:** Although the `finalDefault` [attribute] of schema may include values other than *restriction*, *list* or *union*, those values are ignored in the determination of {final}
{target namespace}	The actual value of the `targetNamespace` [attribute] of the parent `schema` element information item.
{annotation}	The annotation corresponding to the <annotation> element information item in the [children], if present, otherwise null

A ·derived· datatype can be ·derived· from a ·primitive· datatype or another ·derived· datatype by one of three means: by *restriction*, by *list* or by *union*.

4.1.2.1 Derivation by restriction

XML Representation Summary: `restriction` Element Information Item

```
<restriction
  base = QName
  id = ID
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleType?, (minExclusive | minInclusive | maxExclusive |
maxInclusive | totalDigits | fractionDigits | length | minLength | maxLength | enumeration
| whiteSpace | pattern)*))
</restriction>
```

Simple Type Definition Schema Component

Property	Representation
{variety}	The actual value of {variety} of {base type definition}
{facets}	The union of the set of Facets (§2.4) components resolved to by the facet [children] merged with {facets} from {base type definition}, subject to the Facet Restriction Valid constraints specified in Facets (§2.4).
{base type definition}	The Simple Type Definition component resolved to by the actual value of the `base` [attribute] or the <simpleType> [children], whichever is present.

Example

An electronic commerce schema might define a datatype called *Sku* (the barcode number that appears on products) from the ·built-in· datatype string by supplying a value for the ·pattern· facet.

```
<simpleType name='Sku'>
    <restriction base='string'>
      <pattern value='\d{3}-[A-Z]{2}'/>
```

```
    </restriction>
  </simpleType>
```

In this case, *Sku* is the name of the new ·user-derived· datatype, string is its ·base type· and ·pattern· is the facet.

4.1.2.2 Derivation by list

XML Representation Summary: `list` Element Information Item

```
<list
  id = ID
  itemType = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleType?))
</list>
```

Simple Type Definition Schema Component

Property	Representation
{variety}	list
{item type definition}	The Simple Type Definition component resolved to by the actual value of the `itemType` [attribute] or the <simpleType> [children], whichever is present.

A ·list· datatype must be ·derived· from an ·atomic· or a ·union· datatype, known as the ·itemType· of the ·list· datatype. This yields a datatype whose ·value space· is composed of finite-length sequences of values from the ·value space· of the ·itemType· and whose ·lexical space· is composed of white space separated lists of literals of the ·itemType·.

Example

A system might want to store lists of floating point values.

```
  <simpleType name='listOfFloat'>
    <list itemType='float'/>
  </simpleType>
```

In this case, *listOfFloat* is the name of the new ·user-derived· datatype, float is its ·itemType· and ·list· is the derivation method.

As mentioned in List datatypes (§2.5.1.2), when a datatype is ·derived· from a ·list· datatype, the following ·constraining facet·s can be used:

- ·length·
- ·maxLength·
- ·minLength·
- ·enumeration·
- ·pattern·
- ·whiteSpace·

regardless of the ·constraining facet·s that are applicable to the ·atomic· datatype that serves as the ·itemType· of the ·list·.

For each of ·length·, ·maxLength· and ·minLength·, the *unit of length* is measured in number of list items. The value of ·whiteSpace· is fixed to the value *collapse*.

4.1.2.3 Derivation by union

XML Representation Summary: union Element Information Item

```
<union
  id = ID
  memberTypes = List of QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, (simpleType*))
</union>
```

Simple Type Definition Schema Component

Property	Representation
{variety}	union
{member type definitions}	The sequence of Simple Type Definition components resolved to by the items in the actual value of the memberTypes [attribute], if any, in order, followed by the Simple Type Definition components resolved to by the <simpleType> [children], if any, in order. If {variety} is *union* for any Simple Type Definition components resolved to above, then the that Simple Type Definition is replaced by its {member type definitions}.

A ·union· datatype can be ·derived· from one or more ·atomic·, ·list· or other ·union· datatypes, known as the ·memberTypes· of that ·union· datatype.

Example

As an example, taken from a typical display oriented text markup language, one might want to express font sizes as an integer between 8 and 72, or with one of the tokens "small", "medium" or "large". The ·union· type definition below would accomplish that.

```
<xsd:attribute name="size">
  <xsd:simpleType>
    <xsd:union>
      <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
          <xsd:minInclusive value="8"/>
          <xsd:maxInclusive value="72"/>
        </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="small"/>
          <xsd:enumeration value="medium"/>
          <xsd:enumeration value="large"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:union>
  </xsd:simpleType>
</xsd:attribute>
<p>
<font size='large'>A header</font>
</p>
<p>
<font size='12'>this is a test</font>
</p>
```

As mentioned in Union datatypes (§2.5.1.3), when a datatype is ·derived· from a ·union· datatype, the only following ·constraining facet·s can be used:

- ·pattern·

- ·enumeration·

regardless of the ·constraining facet·s that are applicable to the datatypes that participate in the ·union·

4.1.3 Constraints on XML Representation of Simple Type Definition

Schema Representation Constraint: Single Facet Value
Unless otherwise specifically allowed by this specification (Multiple patterns (§4.3.4.3) and Multiple enumerations (§4.3.5.3)) any given ·constraining facet· can only be specifed once within a single derivation step.

Schema Representation Constraint: itemType attribute or simpleType child
Either the `itemType` [attribute] or the <simpleType> [child] of the <list> element must be present, but not both.

Schema Representation Constraint: base attribute or simpleType child
Either the `base` [attribute] or the `simpleType` [child] of the <restriction> element must be present, but not both.

Schema Representation Constraint: memberTypes attribute or simpleType children
Either the `memberTypes` [attribute] of the <union> element must be non-empty or there must be at least one `simpleType` [child].

4.1.4 Simple Type Definition Validation Rules

Validation Rule: Facet Valid
A value in a ·value space· is facet-valid with respect to a ·constraining facet· component if:
1 the value is facet-valid with respect to the particular ·constraining facet· as specified below.

Validation Rule: Datatype Valid
A string is datatype-valid with respect to a datatype definition if:
1 it ·match·es a literal in the ·lexical space· of the datatype, determined as follows:
1.1 if ·pattern· is a member of {facets}, then the string must be pattern valid (§4.3.4.4);
1.2 if ·pattern· is not a member of {facets}, then
1.2.1 if {variety} is ·atomic· then the string must ·match· a literal in the ·lexical space· of {base type definition}
1.2.2 if {variety} is ·list· then the string must be a sequence of white space separated tokens, each of which ·match·es a literal in the ·lexical space· of {item type definition}
1.2.3 if {variety} is ·union· then the string must ·match· a literal in the ·lexical space· of at least one member of {member type definitions}
2 the value denoted by the literal ·match·ed in the previous step is a member of the ·value space· of the datatype, as determined by it being Facet Valid (§4.1.4) with respect to each member of {facets} (except for ·pattern·).

4.1.5 Constraints on Simple Type Definition Schema Components

Schema Component Constraint: applicable facets
The ·constraining facet·s which are allowed to be members of {facets} are dependent on {base type definition} as specified in the following table:

{base type definition}	**applicable {facets}**
If {variety} is list, then	
[all datatypes]	length, minLength, maxLength, pattern, enumeration, whiteSpace
If {variety} is union, then	
[all datatypes]	pattern, enumeration
else if {variety} is atomic, then	
string	length, minLength, maxLength, pattern, enumeration, whiteSpace

boolean	pattern, whiteSpace
float	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
double	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
decimal	totalDigits, fractionDigits, pattern, whiteSpace, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
duration	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
dateTime	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
time	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
date	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
gYearMonth	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
gYear	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
gMonthDay	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
gDay	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
gMonth	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
hexBinary	length, minLength, maxLength, pattern, enumeration, whiteSpace
base64Binary	length, minLength, maxLength, pattern, enumeration, whiteSpace
anyURI	length, minLength, maxLength, pattern, enumeration, whiteSpace
QName	length, minLength, maxLength, pattern, enumeration, whiteSpace
NOTATION	length, minLength, maxLength, pattern, enumeration, whiteSpace

Schema Component Constraint: list of atomic

If {variety} is ·list·, then the {variety} of {item type definition} ·must· be ·atomic· or ·union·.

Schema Component Constraint: no circular unions

If {variety} is ·union·, then it is an ·error· if {name} and {target namespace} ·match· {name} and {target namespace} of any member of {member type definitions}.

4.1.6 Simple Type Definition for anySimpleType

There is a simple type definition nearly equivalent to the simple version of the ur-type definition present in every schema by definition. It has the following properties:

Schema Component: anySimpleType

{name}
anySimpleType
{target namespace}
http://www.w3.org/2001/XMLSchema
{basetype definition}
the ur-type definition
{final}

the empty set
{variety}
absent

4.2 Fundamental Facets

4.2.1 equal

Every ·value space· supports the notion of equality, with the following rules:

- for any *a* and *b* in the ·value space·, either *a* is equal to *b*, denoted *a* = *b*, or *a* is not equal to *b*, denoted *a != b*
- there is no pair *a* and *b* from the ·value space· such that both *a* = *b* and *a != b*
- for all *a* in the ·value space·, *a* = *a*
- for any *a* and *b* in the ·value space·, *a* = *b* if and only if *b* = *a*
- for any *a*, *b* and *c* in the ·value space·, if *a* = *b* and *b* = *c*, then *a* = *c*
- for any *a* and *b* in the ·value space· if *a* = *b*, then *a* and *b* cannot be distinguished (i.e., equality is identity)

Note that a consequence of the above is that, given ·value space· *A* and ·value space· *B* where *A* and *B* are not related by ·restriction· or ·union·, for every pair of values *a* from *A* and *b* from *B*, *a != b*.

On every datatype, the operation Equal is defined in terms of the equality property of the ·value space·: for any values *a, b* drawn from the ·value space·, *Equal(a,b)* is true if *a* = *b*, and false otherwise.

NOTE: There is no schema component corresponding to the **equal** ·fundamental facet·.

4.2.2 ordered

[Definition:] An **order relation** on a ·value space· is a mathematical relation that imposes a ·total order· or a ·partial order· on the members of the ·value space·.

[Definition:] A ·value space·, and hence a datatype, is said to be **ordered** if there exists an ·order-relation· defined for that ·value space·.

[Definition:] A **partial order** is an ·order-relation· that is **irreflexive**, **asymmetric** and **transitive**.

A ·partial order· has the following properties:

- for no *a* in the ·value space·, *a* < *a* (irreflexivity)
- for all *a* and *b* in the ·value space·, *a* < *b* implies not(*b* < *a*) (asymmetry)
- for all *a*, *b* and *c* in the ·value space·, *a* < *b* and *b* < *c* implies *a* < *c* (transitivity)

The notation *a* <> *b* is used to indicate the case when *a != b* and neither *a* < *b* nor *b* < *a*

[Definition:] A **total order** is an ·partial order· such that for no *a* and *b* is it the case that *a* <> *b*.

A ·total order· has all of the properties specified above for ·partial order·, plus the following property:

- for all *a* and *b* in the ·value space·, either *a* < *b* or *b* < *a* or *a* = *b*

NOTE: The fact that this specification does not define an ·order-relation· for some datatype does

not mean that some other application cannot treat that datatype as being ordered by imposing its own order relation.

·ordered· provides for:

- indicating whether an ·order-relation· is defined on a ·value space·, and if so, whether that ·order-relation· is a ·partial order· or a ·total order·

4.2.2.1 The ordered Schema Component

Schema Component: ordered
{value} One of *{false, partial, total}*.

{value} depends on {variety}, {facets} and {member type definitions} in the Simple Type Definition component in which a ·ordered· component appears as a member of {fundamental facets}.

When {variety} is ·atomic·, {value} is inherited from {value} of {base type definition}. For all ·primitive· types {value} is as specified in the table in Fundamental Facets (§C.1).

When {variety} is ·list·, {value} is *false*.

When {variety} is ·union·, if {value} is *true* for every member of {member type definitions} and all members of {member type definitions} share a common ancestor, then {value} is *true*; else {value} is *false*.

4.2.3 bounded

[Definition:] A value *u* in an ·ordered· ·value space· *U* is said to be an **inclusive upper bound** of a ·value space· *V* (where *V* is a subset of *U*) if for all *v* in *V*, *u* >= *v*.

[Definition:] A value *u* in an ·ordered· ·value space· *U* is said to be an **exclusive upper bound** of a ·value space· *V* (where *V* is a subset of *U*) if for all *v* in *V*, *u* > *v*.

[Definition:] A value *l* in an ·ordered· ·value space· *L* is said to be an **inclusive lower bound** of a ·value space· *V* (where *V* is a subset of *L*) if for all *v* in *V*, *l* <= *v*.

[Definition:] A value *l* in an ·ordered· ·value space· *L* is said to be an **exclusive lower bound** of a ·value space· *V* (where *V* is a subset of *L*) if for all *v* in *V*, *l* < *v*.

[Definition:] A datatype is **bounded** if its ·value space· has either an ·inclusive upper bound· or an ·exclusive upper bound· and either an ·inclusive lower bound· and an ·exclusive lower bound·.

·bounded· provides for:

- indicating whether a ·value space· is ·bounded·

4.2.3.1 The bounded Schema Component

Schema Component: bounded
{value} A boolean.

{value} depends on {variety}, {facets} and {member type definitions} in the Simple Type Definition component in which a ·bounded· component appears as a member of {fundamental facets}.

When {variety} is ·atomic·, if one of ·minInclusive· or ·minExclusive· and one of ·maxInclusive· or ·maxExclusive· are among {facets} , then {value} is *true*; else {value} is *false*.

When {variety} is ·list·, if ·length· or both of ·minLength· and ·maxLength· are among {facets}, then {value} is *true*; else {value} is *false*.

When {variety} is ·union·, if {value} is *true* for every member of {member type definitions} and all members of {member type definitions} share a common ancestor, then {value} is *true*; else {value} is *false*.

4.2.4 cardinality

[Definition:] Every ·value space· has associated with it the concept of **cardinality**. Some ·value space·s are finite, some are countably infinite while still others could conceivably be uncountably infinite (although no ·value space· defined by this specification is uncountable infinite). A datatype is said to have the cardinality of its ·value space·.

It is sometimes useful to categorize ·value space·s (and hence, datatypes) as to their cardinality. There are two significant cases:

- ·value space·s that are finite
- ·value space·s that are countably infinite

·cardinality· provides for:

- indicating whether the ·cardinality· of a ·value space· is *finite* or *countably infinite*

4.2.4.1 The cardinality Schema Component

Schema Component: cardinality
{value} One of *{finite, countably infinite}*.

{value} depends on {variety}, {facets} and {member type definitions} in the Simple Type Definition component in which a ·cardinality· component appears as a member of {fundamental facets}.

When {variety} is ·atomic· and {value} of {base type definition} is *finite*, then {value} is *finite*.

When {variety} is ·atomic· and {value} of {base type definition} is *countably infinite* and **either** of the following conditions are true, then {value} is *finite*; else {value} is *countably infinite*:

1. one of ·length·, ·maxLength·, ·totalDigits· is among {facets},
2. **all** of the following are true:
    1. one of ·minInclusive· or ·minExclusive· is among {facets}
    2. one of ·maxInclusive· or ·maxExclusive· is among {facets}
    3. **either** of the following are true:
        1. ·fractionDigits· is among {facets}
        2. {base type definition} is one of date, gYearMonth, gYear, gMonthDay, gDay or gMonth or any type ·derived· from them

When {variety} is ·list·, if ·length· or both of ·minLength· and ·maxLength· are among {facets}, then {value} is *finite*; else {value} is *countably infinite*.

When {variety} is ·union·, if {value} is *finite* for every member of {member type definitions}, then {value} is *finite*; else {value} is *countably infinite*.

4.2.5 numeric

[Definition:] A datatype is said to be **numeric** if its values are conceptually quantities (in some mathematical number system).

[Definition:] A datatype whose values are not ·numeric· is said to be **non-numeric**.

·numeric· provides for:

- indicating whether a ·value space· is ·numeric·

4.2.5.1 The numeric Schema Component

Schema Component: numeric
{value} A boolean

{value} depends on {variety}, {facets}, {base type definition} and {member type definitions} in the Simple Type Definition component in which a ·cardinality· component appears as a member of {fundamental facets}.

When {variety} is ·atomic·, {value} is inherited from {value} of {base type definition}. For all ·primitive· types {value} is as specified in the table in Fundamental Facets (§C.1).

When {variety} is ·list·, {value} is *false*.

When {variety} is ·union·, if {value} is *true* for every member of {member type definitions}, then {value} is *true*; else {value} is *false*.

4.3 Constraining Facets

4.3.1 length

[Definition:] **length** is the number of *units of length*, where *units of length* varies depending on the type that is being ·derived· from. The value of **length** ·must· be a nonNegativeInteger.

For string and datatypes ·derived· from string, **length** is measured in units of characters as defined in [XML 1.0 (Second Edition)]. For anyURI, **length** is measured in units of characters (as for string). For hexBinary and base64Binary and datatypes ·derived· from them, **length** is measured in octets (8 bits) of binary data. For datatypes ·derived· by ·list·, **length** is measured in number of list items.

> **NOTE:** For string and datatypes ·derived· from string, **length** will not always coincide with "string length" as perceived by some users or with the number of storage units in some digital representation. Therefore, care should be taken when specifying a value for **length** and in attempting to infer storage requirements from a given value for **length**.

·length· provides for:

- Constraining a ·value space· to values with a specific number of *units of length*, where *units of length* varies depending on {base type definition}.

Example

The following is the definition of a ·user-derived· datatype to represent product codes which must be exactly 8 characters in length. By fixing the value of the **length** facet we ensure that types derived from productCode can change or set the values of other facets, such as **pattern**, but cannot change the length.

```
<simpleType name='productCode'>
   <restriction base='string'>
     <length value='8' fixed='true'/>
   </restriction>
</simpleType>
```

4.3.1.1 The length Schema Component

Schema Component: length

{value}
A nonNegativeInteger.

{fixed}
A boolean.

{annotation}
Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} cannot specify a value for length other than {value}.

4.3.1.2 XML Representation of length Schema Components

The XML representation for a length schema component is a <length> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `length` Element Information Item

```
<length
  fixed = boolean : false
  id = ID
  value = nonNegativeInteger
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</length>
```

length Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{fixed}	The actual value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.1.3 length Validation Rules

Validation Rule: Length Valid

A value in a ·value space· is facet-valid with respect to ·length·, determined as follows:

1 if the {variety} is ·atomic· then

1.1 if {primitive type definition} is string, then the length of the value, as measured in characters ·must· be equal to {value};

1.2 if {primitive type definition} is hexBinary or base64Binary, then the length of the value, as measured in octets of the binary data, ·must· be equal to {value};

2 if the {variety} is ·list·, then the length of the value, as measured in list items, ·must· be equal to {value}

4.3.1.4 Constraints on length Schema Components

Schema Component Constraint: length and minLength or maxLength

It is an ·error· for both length and either of minLength or maxLength to be members of {facets}.

Schema Component Constraint: length valid restriction

It is an ·error· if length is among the members of {facets} of {base type definition} and {value} is not equal to the {value} of the parent length.

4.3.2 minLength

[Definition:] **minLength** is the minimum number of *units of length*, where *units of length* varies depending on the type that is being ·derived· from. The value of **minLength** ·must· be a nonNegativeInteger.

For string and datatypes ·derived· from string, **minLength** is measured in units of characters as defined in [XML 1.0 (Second Edition)]. For hexBinary and base64Binary and datatypes ·derived· from them, **minLength** is measured in octets (8 bits) of binary data. For datatypes ·derived· by ·list·, **minLength** is measured in number of list items.

> **NOTE:** For string and datatypes ·derived· from string, **minLength** will not always coincide with "string length" as perceived by some users or with the number of storage units in some digital representation. Therefore, care should be taken when specifying a value for **minLength** and in attempting to infer storage requirements from a given value for **minLength**.

·minLength· provides for:

- Constraining a ·value space· to values with at least a specific number of *units of length*, where *units of length* varies depending on {base type definition}.

Example

The following is the definition of a ·user-derived· datatype which requires strings to have at least one character (i.e., the empty string is not in the ·value space· of this datatype).

```
<simpleType name='non-empty-string'>
  <restriction base='string'>
    <minLength value='1'/>
  </restriction>
</simpleType>
```

4.3.2.1 The minLength Schema Component

Schema Component: minLength

{value}
A nonNegativeInteger.

{fixed}
A boolean.

{annotation}
Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} cannot specify a value for minLength other than {value}.

4.3.2.2 XML Representation of minLength Schema Component

The XML representation for a minLength schema component is a <minLength> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `minLength` Element Information Item

```
<minLength
  fixed = boolean : false
  id = ID
  value = nonNegativeInteger
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</minLength>
```

minLength Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{fixed}	The actual value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.2.3 minLength Validation Rules

Validation Rule: minLength Valid

A value in a ·value space· is facet-valid with respect to ·minLength·, determined as follows:

1 if the {variety} is ·atomic· then

1.1 if {primitive type definition} is string, then the length of the value, as measured in characters ·must· be greater than or equal to {value};

1.2 if {primitive type definition} is hexBinary or base64Binary, then the length of the value, as measured in octets of the binary data, ·must· be greater than or equal to {value};

2 if the {variety} is ·list·, then the length of the value, as measured in list items, ·must· be greater than or equal to {value}

4.3.2.4 Constraints on minLength Schema Components

Schema Component Constraint: minLength <= maxLength

If both minLength and maxLength are members of {facets}, then the {value} of minLength ·must· be less than or equal to the {value} of maxLength.

Schema Component Constraint: minLength valid restriction

It is an ·error· if minLength is among the members of {facets} of {base type definition} and {value} is less than the {value} of the parent minLength.

4.3.3 maxLength

[Definition:] **maxLength** is the maximum number of *units of length*, where *units of length* varies depending on the type that is being ·derived· from. The value of **maxLength** ·must· be a nonNegativeInteger.

For string and datatypes ·derived· from string, **maxLength** is measured in units of characters as defined in [XML 1.0 (Second Edition)]. For hexBinary and base64Binary and datatypes ·derived· from them, **maxLength** is measured in octets (8 bits) of binary data. For datatypes ·derived· by ·list·, **maxLength** is measured in number of

list items.

> **NOTE:** For string and datatypes ·derived· from string, **maxLength** will not always coincide with "string length" as perceived by some users or with the number of storage units in some digital representation. Therefore, care should be taken when specifying a value for **maxLength** and in attempting to infer storage requirements from a given value for **maxLength**.

·maxLength· provides for:

- Constraining a ·value space· to values with at most a specific number of *units of length*, where *units of length* varies depending on {base type definition}.

Example
The following is the definition of a ·user-derived· datatype which might be used to accept form input with an upper limit to the number of characters that are acceptable.

```
<simpleType name='form-input'>
  <restriction base='string'>
    <maxLength value='50'/>
  </restriction>
</simpleType>
```

4.3.3.1 The maxLength Schema Component

Schema Component: maxLength

{value}
A nonNegativeInteger.
{fixed}
A boolean.
{annotation}
Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} cannot specify a value for maxLength other than {value}.

4.3.3.2 XML Representation of maxLength Schema Components

The XML representation for a maxLength schema component is a <maxLength> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `maxLength` Element Information Item

```
<maxLength
  fixed = boolean : false
  id = ID
  value = nonNegativeInteger
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</maxLength>
```

maxLength Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{fixed}	The actual value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.3.3 maxLength Validation Rules

Validation Rule: maxLength Valid

A value in a ·value space· is facet-valid with respect to ·maxLength·, determined as follows:

1 if the {variety} is ·atomic· then

1.1 if {primitive type definition} is string, then the length of the value, as measured in characters ·must· be less than or equal to {value};

1.2 if {primitive type definition} is hexBinary or base64Binary, then the length of the value, as measured in octets of the binary data, ·must· be less than or equal to {value};

2 if the {variety} is ·list·, then the length of the value, as measured in list items, ·must· be less than or equal to {value}

4.3.3.4 Constraints on maxLength Schema Components

Schema Component Constraint: maxLength valid restriction

It is an ·error· if maxLength is among the members of {facets} of {base type definition} and {value} is greater than the {value} of the parent maxLength.

4.3.4 pattern

[Definition:] **pattern** is a constraint on the ·value space· of a datatype which is achieved by constraining the ·lexical space· to literals which match a specific pattern. The value of **pattern** ·must· be a ·regular expression·.

·pattern· provides for:

- Constraining a ·value space· to values that are denoted by literals which match a specific ·regular expression·.

Example

The following is the definition of a ·user-derived· datatype which is a better representation of postal codes in the United States, by limiting strings to those which are matched by a specific ·regular expression·.

```
<simpleType name='better-us-zipcode'>
  <restriction base='string'>
    <pattern value='[0-9]{5}(-[0-9]{4})?'/>
  </restriction>
</simpleType>
```

4.3.4.1 The pattern Schema Component

Schema Component: pattern

{value}
A ·regular expression·.

{annotation}
Optional. An annotation.

4.3.4.2 XML Representation of pattern Schema Components

The XML representation for a pattern schema component is a <pattern> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `pattern` Element Information Item

```
<pattern
  id = ID
  value = anySimpleType
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</pattern>
```

{value} must be a valid regular expression.

pattern Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.4.3 Constraints on XML Representation of pattern

Schema Representation Constraint: Multiple patterns

If multiple <pattern> element information items appear as [children] of a <simpleType>, the [value]s should be combined as if they appeared in a single regular expression as separate branches.

> **NOTE:** It is a consequence of the schema representation constraint Multiple patterns (§4.3.4.3) and of the rules for restriction that pattern facets specified on the *same* step in a type derivation are **OR**ed together, while pattern facets specified on *different* steps of a type derivation are **AND**ed together.
>
> Thus, to impose two pattern constraints simultaneously, schema authors may either write a single pattern which expresses the intersection of the two patterns they wish to impose, or define each pattern on a separate type derivation step.

4.3.4.4 pattern Validation Rules

Validation Rule: pattern valid

A literal in a lexical space is facet-valid with respect to pattern if:

1 the literal is among the set of character sequences denoted by the regular expression specified in {value}.

4.3.5 enumeration

[Definition:] **enumeration** constrains the value space to a specified set of values.

enumeration does not impose an order relation on the value space it creates; the value of the ordered property of the derived datatype remains that of the datatype from which it is derived.

enumeration provides for:

- Constraining a value space to a specified set of values.

Example

The following example is a datatype definition for a user-derived datatype which limits the values of dates to

the three US holidays enumerated. This datatype definition would appear in a schema authored by an "end-user" and shows how to define a datatype by enumerating the values in its ·value space·. The enumerated values must be type-valid literals for the ·base type·.

```
<simpleType name='holidays'>
    <annotation>
        <documentation>some US holidays</documentation>
    </annotation>
    <restriction base='gMonthDay'>
      <enumeration value='--01-01'>
        <annotation>
            <documentation>New Year's day</documentation>
        </annotation>
      </enumeration>
      <enumeration value='--07-04'>
        <annotation>
            <documentation>4th of July</documentation>
        </annotation>
      </enumeration>
      <enumeration value='--12-25'>
        <annotation>
            <documentation>Christmas</documentation>
        </annotation>
      </enumeration>
    </restriction>
</simpleType>
```

4.3.5.1 The enumeration Schema Component

Schema Component: enumeration

{value}
A set of values from the ·value space· of the {base type definition}.

{annotation}
Optional. An annotation.

4.3.5.2 XML Representation of enumeration Schema Components

The XML representation for an enumeration schema component is an <enumeration> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `enumeration` Element Information Item

```
<enumeration
  id = ID
  value = anySimpleType
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</enumeration>
```

{value} ·must· be in the ·value space· of {base type definition}.

enumeration Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.5.3 Constraints on XML Representation of enumeration

Schema Representation Constraint: Multiple enumerations
If multiple <enumeration> element information items appear as [children] of a <simpleType> the {value} of the enumeration component should be the set of all such [value]s.

4.3.5.4 enumeration Validation Rules

Validation Rule: enumeration valid
A value in a ·value space· is facet-valid with respect to ·enumeration· if the value is one of the values specified in {value}

4.3.5.5 Constraints on enumeration Schema Components

Schema Component Constraint: enumeration valid restriction
It is an ·error· if any member of {value} is not in the ·value space· of {base type definition}.

4.3.6 whiteSpace

[Definition:] **whiteSpace** constrains the ·value space· of types ·derived· fromstring such that the various behaviors specified in Attribute Value Normalization in [XML 1.0 (Second Edition)] are realized. The value of **whiteSpace** must be one of {preserve, replace, collapse}.

preserve
No normalization is done, the value is not changed (this is the behavior required by [XML 1.0 (Second Edition)] for element content)
replace
All occurrences of #x9 (tab), #xA (line feed) and #xD (carriage return) are replaced with #x20 (space)
collapse
After the processing implied by **replace**, contiguous sequences of #x20's are collapsed to a single #x20, and leading and trailing #x20's are removed.

NOTE: The notation #xA used here (and elsewhere in this specification) represents the Universal Character Set (UCS) code point `hexadecimal` A (line feed), which is denoted by U+000A. This notation is to be distinguished from `
`, which is the XML character reference to that same UCS code point.

whiteSpace is applicable to all ·atomic· and ·list· datatypes. For all ·atomic· datatypes other thanstring (and types ·derived· by ·restriction· from it) the value of **whiteSpace** is `collapse` and cannot be changed by a schema author; for string the value of **whiteSpace** is `preserve`; for any type ·derived· by ·restriction· fromstring the value of **whiteSpace** can be any of the three legal values. For all datatypes ·derived· by ·list· the value of **whiteSpace** is `collapse` and cannot be changed by a schema author. For all datatypes ·derived· by ·union· **whiteSpace** does not apply directly; however, the normalization behavior of ·union· types is controlled by the value of **whiteSpace** on that one of the ·memberTypes· against which the ·union· is successfully validated.

NOTE: For more information on **whiteSpace**, see the discussion on white space normalization in Schema Component Details in [XML Schema Part 1: Structures].

·whiteSpace· provides for:

- Constraining a ·value space· according to the white space normalization rules.

Example
The following example is the datatype definition for the token ·built-in· ·derived· datatype.

```
<simpleType name='token'>
    <restriction base='normalizedString'>
      <whiteSpace value='collapse'/>
```

```
    </restriction>
  </simpleType>
```

4.3.6.1 The whiteSpace Schema Component

Schema Component: whiteSpace

{value}
One of {preserve, replace, collapse}.
{fixed}
A boolean.
{annotation}
Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} cannot specify a value for whiteSpace other than {value}.

4.3.6.2 XML Representation of whiteSpace Schema Components

The XML representation for a whiteSpace schema component is a <whiteSpace> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: whiteSpace Element Information Item

```
<whiteSpace
  fixed = boolean : false
  id = ID
  value = (collapse | preserve | replace)
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</whiteSpace>
```

whiteSpace Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{fixed}	The actual value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.6.3 whiteSpace Validation Rules

NOTE: There are no ·Validation Rule·s associated ·whiteSpace·. For more information, see the discussion on white space normalization in Schema Component Details in [XML Schema Part 1: Structures].

4.3.6.4 Constraints on whiteSpace Schema Components

Schema Component Constraint: whiteSpace valid restriction
It is an ·error· if whiteSpace is among the members of {facets} of {base type definition} and any of the following conditions is true:
1 {value} is *replace* or *preserve* and the {value} of the parent whiteSpace is *collapse*
2 {value} is *preserve* and the {value} of the parent whiteSpace is *replace*

4.3.7 maxInclusive

[Definition:] **maxInclusive** is the ·inclusive upper bound· of the ·value space· for a datatype with the ·ordered· property. The value of **maxInclusive** ·must· be in the ·value space· of the ·base type·.

·maxInclusive· provides for:

- Constraining a ·value space· to values with a specific ·inclusive upper bound·.

Example

The following is the definition of a ·user-derived· datatype which limits values to integers less than or equal to 100, using ·maxInclusive·.

```
<simpleType name='one-hundred-or-less'>
  <restriction base='integer'>
    <maxInclusive value='100'/>
  </restriction>
</simpleType>
```

4.3.7.1 The maxInclusive Schema Component

Schema Component: maxInclusive

{value}
A value from the ·value space· of the {base type definition}.

{fixed}
A boolean.

{annotation}
Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} cannot specify a value for maxInclusive other than {value}.

4.3.7.2 XML Representation of maxInclusive Schema Components

The XML representation for a maxInclusive schema component is a <maxInclusive> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `maxInclusive` Element Information Item

```
<maxInclusive
  fixed = boolean : false
  id = ID
  value = anySimpleType
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</maxInclusive>
```

{value} ·must· be in the ·value space· of {base type definition}.

maxInclusive Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{fixed}	The actual value of the `fixed` [attribute], if present, otherwise false, if present, otherwise false
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.7.3 maxInclusive Validation Rules

Validation Rule: maxInclusive Valid

A value in an ·ordered· ·value space· is facet-valid with respect to ·maxInclusive·, determined as follows:

1 if the ·numeric· property in {fundamental facets} is *true*, then the value ·must· be numerically less than or equal to {value};

2 if the ·numeric· property in {fundamental facets} is *false* (i.e., {base type definition} is one of the date and time related datatypes), then the value ·must· be chronologically less than or equal to {value};

4.3.7.4 Constraints on maxInclusive Schema Components

Schema Component Constraint: minInclusive <= maxInclusive

It is an ·error· for the value specified for ·minInclusive· to be greater than the value specified for ·maxInclusive· for the same datatype.

Schema Component Constraint: maxInclusive valid restriction

It is an ·error· if any of the following conditions is true:

1 maxInclusive is among the members of {facets} of {base type definition} and {value} is greater than the {value} of the parent maxInclusive

2 maxExclusive is among the members of {facets} of {base type definition} and {value} is greater than or equal to the {value} of the parent maxExclusive

3 minInclusive is among the members of {facets} of {base type definition} and {value} is less than the {value} of the parent minInclusive

4 minExclusive is among the members of {facets} of {base type definition} and {value} is less than or equal to the {value} of the parent minExclusive

4.3.8 maxExclusive

[Definition:] **maxExclusive** is the ·exclusive upper bound· of the ·value space· for a datatype with the ·ordered· property. The value of **maxExclusive** ·must· be in the ·value space· of the ·base type·.

·maxExclusive· provides for:

- Constraining a ·value space· to values with a specific ·exclusive upper bound·.

Example

The following is the definition of a ·user-derived· datatype which limits values to integers less than or equal to 100, using ·maxExclusive·.

```
<simpleType name='less-than-one-hundred-and-one'>
  <restriction base='integer'>
    <maxExclusive value='101'/>
  </restriction>
</simpleType>
```

Note that the ·value space· of this datatype is identical to the previous one (named 'one-hundred-or-less').

4.3.8.1 The maxExclusive Schema Component

Schema Component: maxExclusive

{value}
A value from the ·value space· of the {base type definition}.

{fixed}
A boolean.

{annotation}
Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} cannot specify a value for maxExclusive other than {value}.

4.3.8.2 XML Representation of maxExclusive Schema Components

The XML representation for a maxExclusive schema component is a <maxExclusive> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `maxExclusive` Element Information Item

```
<maxExclusive
  fixed = boolean : false
  id = ID
  value = anySimpleType
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</maxExclusive>
```

{value} ·must· be in the ·value space· of {base type definition}.

maxExclusive Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{fixed}	The actual value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.8.3 maxExclusive Validation Rules

Validation Rule: maxExclusive Valid

A value in an ·ordered· ·value space· is facet-valid with respect to ·maxExclusive·, determined as follows:

1 if the ·numeric· property in {fundamental facets} is *true*, then the value ·must· be numerically less than {value};

2 if the ·numeric· property in {fundamental facets} is *false* (i.e., {base type definition} is one of the date and time related datatypes), then the value ·must· be chronologically less than {value};

4.3.8.4 Constraints on maxExclusive Schema Components

Schema Component Constraint: maxInclusive and maxExclusive

It is an ·error· for both ·maxInclusive· and ·maxExclusive· to be specified in the same derivation step of a datatype definition.

Schema Component Constraint: minExclusive <= maxExclusive

It is an ·error· for the value specified for ·minExclusive· to be greater than the value specified for ·maxExclusive· for the same datatype.

Schema Component Constraint: maxExclusive valid restriction

It is an ·error· if any of the following conditions is true:

1 maxExclusive is among the members of {facets} of {base type definition} and {value} is greater than the {value} of the parent maxExclusive

2 maxInclusive is among the members of {facets} of {base type definition} and {value} is greater than the {value} of the parent maxInclusive

3 minInclusive is among the members of {facets} of {base type definition} and {value} is less than or equal to the {value} of the parent minInclusive

4 minExclusive is among the members of {facets} of {base type definition} and {value} is less than or equal to the {value} of the parent minExclusive

4.3.9 minExclusive

[Definition:] **minExclusive** is the ·exclusive lower bound· of the ·value space· for a datatype with the ·ordered· property. The value of **minExclusive** ·must· be in the ·value space· of the ·base type·.

·minExclusive· provides for:

- Constraining a ·value space· to values with a specific ·exclusive lower bound·.

Example

The following is the definition of a ·user-derived· datatype which limits values to integers greater than or equal to 100, using ·minExclusive·.

```
<simpleType name='more-than-ninety-nine'>
  <restriction base='integer'>
    <minExclusive value='99'/>
  </restriction>
</simpleType>
```

Note that the ·value space· of this datatype is identical to the previous one (named 'one-hundred-or-more').

4.3.9.1 The minExclusive Schema Component

Schema Component: minExclusive

{value}
A value from the ·value space· of the {base type definition}.

{fixed}
A boolean.

{annotation}
Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} cannot specify a value for minExclusive other than {value}.

4.3.9.2 XML Representation of minExclusive Schema Components

The XML representation for a minExclusive schema component is a <minExclusive> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `minExclusive` Element Information Item

```
<minExclusive
  fixed = boolean : false
  id = ID
  value = anySimpleType
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</minExclusive>
```

{value} ·must· be in the ·value space· of {base type definition}.

minExclusive Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{fixed}	The actual value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.9.3 minExclusive Validation Rules

Validation Rule: minExclusive Valid

A value in an ·ordered· ·value space· is facet-valid with respect to ·minExclusive· if:

1 if the ·numeric· property in {fundamental facets} is *true*, then the value ·must· be numerically greater than {value};

2 if the ·numeric· property in {fundamental facets} is *false* (i.e., {base type definition} is one of the date and time related datatypes), then the value ·must· be chronologically greater than {value};

4.3.9.4 Constraints on minExclusive Schema Components

Schema Component Constraint: minInclusive and minExclusive

It is an ·error· for both ·minInclusive· and ·minExclusive· to be specified for the same datatype.

Schema Component Constraint: minExclusive < maxInclusive

It is an ·error· for the value specified for ·minExclusive· to be greater than or equal to the value specified for ·maxInclusive· for the same datatype.

Schema Component Constraint: minExclusive valid restriction

It is an ·error· if any of the following conditions is true:

1 minExclusive is among the members of {facets} of {base type definition} and {value} is less than the {value} of the parent minExclusive

2 maxInclusive is among the members of {facets} of {base type definition} and {value} is greater the {value} of the parent maxInclusive

3 minInclusive is among the members of {facets} of {base type definition} and {value} is less than the {value} of the parent minInclusive

4 maxExclusive is among the members of {facets} of {base type definition} and {value} is greater than or equal to the {value} of the parent maxExclusive

4.3.10 minInclusive

[Definition:] **minInclusive** is the ·inclusive lower bound· of the ·value space· for a datatype with the ·ordered· property. The value of **minInclusive** ·must· be in the ·value space· of the ·base type·.

·minInclusive· provides for:

- Constraining a ·value space· to values with a specific ·inclusive lower bound·.

Example

The following is the definition of a ·user-derived· datatype which limits values to integers greater than or equal to 100, using ·minInclusive·.

```
<simpleType name='one-hundred-or-more'>
  <restriction base='integer'>
    <minInclusive value='100'/>
  </restriction>
</simpleType>
```

4.3.10.1 The minInclusive Schema Component

Schema Component: minInclusive

{value}
A value from the ·value space· of the {base type definition}.

{fixed}
A boolean.

{annotation}
Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} cannot specify a value for minInclusive other than {value}.

4.3.10.2 XML Representation of minInclusive Schema Components

The XML representation for a minInclusive schema component is a <minInclusive> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `minInclusive` Element Information Item

```
<minInclusive
  fixed = boolean : false
  id = ID
  value = anySimpleType
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</minInclusive>
```

{value} ·must· be in the ·value space· of {base type definition}.

minInclusive Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{fixed}	The actual value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.10.3 minInclusive Validation Rules

Validation Rule: minInclusive Valid
A value in an ·ordered· ·value space· is facet-valid with respect to ·minInclusive· if:
1 if the ·numeric· property in {fundamental facets} is *true*, then the value ·must· be numerically greater than or equal to {value};
2 if the ·numeric· property in {fundamental facets} is *false* (i.e., {base type definition} is one of the date and time related datatypes), then the value ·must· be chronologically greater than or equal to {value};

4.3.10.4 Constraints on minInclusive Schema Components

Schema Component Constraint: minInclusive < maxExclusive
It is an ·error· for the value specified for ·minInclusive· to be greater than or equal to the value specified for ·maxExclusive· for the same datatype.

Schema Component Constraint: minInclusive valid restriction
It is an ·error· if any of the following conditions is true:
1 minInclusive is among the members of {facets} of {base type definition} and {value} is less than the

{value} of the parent minInclusive
2 maxInclusive is among the members of {facets} of {base type definition} and {value} is greater the {value} of the parent maxInclusive
3 minExclusive is among the members of {facets} of {base type definition} and {value} is less than or equal to the {value} of the parent minExclusive
4 maxExclusive is among the members of {facets} of {base type definition} and {value} is greater than or equal to the {value} of the parent maxExclusive

4.3.11 totalDigits

[Definition:] **totalDigits** is the maximum number of digits in values of datatypes ·derived· from decimal. The value of **totalDigits** ·must· be a positiveInteger.

·totalDigits· provides for:

- Constraining a ·value space· to values with a specific maximum number of decimal digits (#x30-#x39).

Example

The following is the definition of a ·user-derived· datatype which could be used to represent monetary amounts, such as in a financial management application which does not have figures of $1M or more and only allows whole cents. This definition would appear in a schema authored by an "end-user" and shows how to define a datatype by specifying facet values which constrain the range of the ·base type· in a manner specific to the ·base type· (different than specifying max/min values as before).

```
<simpleType name='amount'>
  <restriction base='decimal'>
    <totalDigits value='8'/>
    <fractionDigits value='2' fixed='true'/>
  </restriction>
</simpleType>
```

4.3.11.1 The totalDigits Schema Component

Schema Component: totalDigits

{value}
A positiveInteger.
{fixed}
A boolean.
{annotation}
Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} cannot specify a value for totalDigits other than {value}.

4.3.11.2 XML Representation of totalDigits Schema Components

The XML representation for a totalDigits schema component is a <totalDigits> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `totalDigits` Element Information Item

```
<totalDigits
  fixed = boolean : false
  id = ID
  value = positiveInteger
  {any attributes with non-schema namespace . . .}>
```

```
  Content: (annotation?)
</totalDigits>
```

totalDigits Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{fixed}	The actual value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.11.3 totalDigits Validation Rules

Validation Rule: totalDigits Valid

A value in a ·value space· is facet-valid with respect to ·totalDigits· if:

1 the number of decimal digits in the value is less than or equal to {value};

4.3.11.4 Constraints on totalDigits Schema Components

Schema Component Constraint: totalDigits valid restriction

It is an ·error· if totalDigits is among the members of {facets} of {base type definition} and {value} is greater than the {value} of the parent totalDigits

4.3.12 fractionDigits

[Definition:] **fractionDigits** is the maximum number of digits in the fractional part of values of datatypes ·derived· from decimal. The value of **fractionDigits** ·must· be a nonNegativeInteger .

·fractionDigits· provides for:

- Constraining a ·value space· to values with a specific maximum number of decimal digits in the fractional part.

Example

The following is the definition of a ·user-derived· datatype which could be used to represent the magnitude of a person's body temperature on the Celsius scale. This definition would appear in a schema authored by an "end-user" and shows how to define a datatype by specifying facet values which constrain the range of the ·base type·.

```
<simpleType name='celsiusBodyTemp'>
  <restriction base='decimal'>
    <totalDigits value='4'/>
    <fractionDigits value='1'/>
    <minInclusive value='36.4'/>
    <maxInclusive value='40.5'/>
  </restriction>
</simpleType>
```

4.3.12.1 The fractionDigits Schema Component

Schema Component: fractionDigits

{value}
A nonNegativeInteger.

{fixed}
A boolean.

{annotation}
Optional. An annotation.

If {fixed} is *true*, then types for which the current type is the {base type definition} cannot specify a value for fractionDigits other than {value}.

4.3.12.2 XML Representation of fractionDigits Schema Components

The XML representation for a fractionDigits schema component is a <fractionDigits> element information item. The correspondences between the properties of the information item and properties of the component are as follows:

XML Representation Summary: `fractionDigits` Element Information Item

```
<fractionDigits
  fixed = boolean : false
  id = ID
  value = nonNegativeInteger
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?)
</fractionDigits>
```

fractionDigits Schema Component

Property	Representation
{value}	The actual value of the `value` [attribute]
{fixed}	The actual value of the `fixed` [attribute], if present, otherwise false
{annotation}	The annotations corresponding to all the <annotation> element information items in the [children], if any.

4.3.12.3 fractionDigits Validation Rules

Validation Rule: fractionDigits Valid
A value in a ·value space· is facet-valid with respect to ·fractionDigits· if:
1 the number of decimal digits in the fractional part of the value is less than or equal to {value};

4.3.12.4 Constraints on fractionDigits Schema Components

Schema Component Constraint: fractionDigits less than or equal to totalDigits
It is an ·error· for ·fractionDigits· to be greater than ·totalDigits·.

5 Conformance

This specification describes two levels of conformance for datatype processors. The first is required of all processors. Support for the other will depend on the application environments for which the processor is intended.

[Definition:] **Minimally conforming** processors ·must· completely and correctly implement the ·Constraint on Schemas· and ·Validation Rule· .

[Definition:] Processors which accept schemas in the form of XML documents as described in XML Representation of Simple Type Definition Schema Components (§4.1.2) (and other relevant portions of Datatype components (§4)) are additionally said to provide **conformance to the XML Representation of Schemas**, and ·must·, when processing schema documents, completely and correctly implement all ·Schema Representation Constraint·s in this specification, and ·must· adhere exactly to the specifications in XML Representation of Simple Type Definition Schema Components (§4.1.2) (and other relevant portions of Datatype components (§4))

for mapping the contents of such documents to schema components for use in validation.

NOTE: By separating the conformance requirements relating to the concrete syntax of XML schema documents, this specification admits processors which validate using schemas stored in optimized binary representations, dynamically created schemas represented as programming language data structures, or implementations in which particular schemas are compiled into executable code such as C or Java. Such processors can be said to be ·minimally conforming· but not necessarily in ·conformance to the XML Representation of Schemas·.

A Schema for Datatype Definitions (normative)

```
<?xml version='1.0'?>
<!-- XML Schema schema for XML Schemas: Part 2: Datatypes -->
<!DOCTYPE xs:schema PUBLIC "-//W3C//DTD XMLSCHEMA 200102//EN"
          "XMLSchema.dtd" [

<!--
     keep this schema XML1.0 DTD valid
  -->
        <!ENTITY % schemaAttrs 'xmlns:hfp CDATA #IMPLIED'>

        <!ELEMENT hfp:hasFacet EMPTY>
        <!ATTLIST hfp:hasFacet
                name NMTOKEN #REQUIRED>

        <!ELEMENT hfp:hasProperty EMPTY>
        <!ATTLIST hfp:hasProperty
                name NMTOKEN #REQUIRED
                value CDATA #REQUIRED>
<!--
        Make sure that processors that do not read the external
        subset will know about the various IDs we declare
  -->
        <!ATTLIST xs:simpleType id ID #IMPLIED>
        <!ATTLIST xs:maxExclusive id ID #IMPLIED>
        <!ATTLIST xs:minExclusive id ID #IMPLIED>
        <!ATTLIST xs:maxInclusive id ID #IMPLIED>
        <!ATTLIST xs:minInclusive id ID #IMPLIED>
        <!ATTLIST xs:totalDigits id ID #IMPLIED>
        <!ATTLIST xs:fractionDigits id ID #IMPLIED>
        <!ATTLIST xs:length id ID #IMPLIED>
        <!ATTLIST xs:minLength id ID #IMPLIED>
        <!ATTLIST xs:maxLength id ID #IMPLIED>
        <!ATTLIST xs:enumeration id ID #IMPLIED>
        <!ATTLIST xs:pattern id ID #IMPLIED>
        <!ATTLIST xs:appinfo id ID #IMPLIED>
        <!ATTLIST xs:documentation id ID #IMPLIED>
        <!ATTLIST xs:list id ID #IMPLIED>
        <!ATTLIST xs:union id ID #IMPLIED>
        ]>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.w3.org/2001/XMLSchema"
        version="Id: datatypes.xsd,v 1.52 2001/04/27 11:49:21 ht Exp "
        xmlns:hfp="http://www.w3.org/2001/XMLSchema-hasFacetAndProperty"
        elementFormDefault="qualified"
        blockDefault="#all"
        xml:lang="en">

  <xs:annotation>
   <xs:documentation source="http://www.w3.org/TR/2001/REC-xmlschema-2-
```

```
20010502/datatypes">
      The schema corresponding to this document is normative,
      with respect to the syntactic constraints it expresses in the
      XML Schema language.  The documentation (within &lt;documentation>
      elements) below, is not normative, but rather highlights important
      aspects of the W3C Recommendation of which this is a part
    </xs:documentation>
  </xs:annotation>

  <xs:annotation>
    <xs:documentation>
      First the built-in primitive datatypes.  These definitions are for
      information only, the real built-in definitions are magic.  Note in
      particular that there is no type named 'anySimpleType'.  The
      primitives should really be derived from no type at all, and
      anySimpleType should be derived as a union of all the primitives.
    </xs:documentation>

    <xs:documentation>
      For each built-in datatype in this schema (both primitive and
      derived) can be uniquely addressed via a URI constructed
      as follows:
        1) the base URI is the URI of the XML Schema namespace
        2) the fragment identifier is the name of the datatype

      For example, to address the int datatype, the URI is:

        http://www.w3.org/2001/XMLSchema#int

      Additionally, each facet definition element can be uniquely
      addressed via a URI constructed as follows:
        1) the base URI is the URI of the XML Schema namespace
        2) the fragment identifier is the name of the facet

      For example, to address the maxInclusive facet, the URI is:

        http://www.w3.org/2001/XMLSchema#maxInclusive

      Additionally, each facet usage in a built-in datatype definition
      can be uniquely addressed via a URI constructed as follows:
        1) the base URI is the URI of the XML Schema namespace
        2) the fragment identifier is the name of the datatype, followed
           by a period (".") followed by the name of the facet

      For example, to address the usage of the maxInclusive facet in
      the definition of int, the URI is:

        http://www.w3.org/2001/XMLSchema#int.maxInclusive

    </xs:documentation>
  </xs:annotation>

  <xs:simpleType name="string" id="string">
    <xs:annotation>
      <xs:appinfo>
        <hfp:hasFacet name="length"/>
        <hfp:hasFacet name="minLength"/>
        <hfp:hasFacet name="maxLength"/>
        <hfp:hasFacet name="pattern"/>
        <hfp:hasFacet name="enumeration"/>
        <hfp:hasFacet name="whiteSpace"/>
        <hfp:hasProperty name="ordered" value="false"/>
        <hfp:hasProperty name="bounded" value="false"/>
```

```
      <hfp:hasProperty name="cardinality" value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
             source="http://www.w3.org/TR/xmlschema-2/#string"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="preserve" id="string.preserve"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="boolean" id="boolean">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasFacet name="pattern"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasProperty name="ordered" value="false"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality" value="finite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#boolean"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="collapse" fixed="true"
      id="boolean.whiteSpace"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="float" id="float">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasFacet name="pattern"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasFacet name="maxInclusive"/>
      <hfp:hasFacet name="maxExclusive"/>
      <hfp:hasFacet name="minInclusive"/>
      <hfp:hasFacet name="minExclusive"/>
      <hfp:hasProperty name="ordered" value="total"/>
      <hfp:hasProperty name="bounded" value="true"/>
      <hfp:hasProperty name="cardinality" value="finite"/>
      <hfp:hasProperty name="numeric" value="true"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#float"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="collapse" fixed="true"
      id="float.whiteSpace"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="double" id="double">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasFacet name="pattern"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasFacet name="maxInclusive"/>
      <hfp:hasFacet name="maxExclusive"/>
      <hfp:hasFacet name="minInclusive"/>
```

```
      <hfp:hasFacet name="minExclusive"/>
      <hfp:hasProperty name="ordered" value="total"/>
      <hfp:hasProperty name="bounded" value="true"/>
      <hfp:hasProperty name="cardinality" value="finite"/>
      <hfp:hasProperty name="numeric" value="true"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#double"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="collapse"  fixed="true"
      id="double.whiteSpace"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="decimal" id="decimal">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasFacet name="totalDigits"/>
      <hfp:hasFacet name="fractionDigits"/>
      <hfp:hasFacet name="pattern"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="maxInclusive"/>
      <hfp:hasFacet name="maxExclusive"/>
      <hfp:hasFacet name="minInclusive"/>
      <hfp:hasFacet name="minExclusive"/>
      <hfp:hasProperty name="ordered" value="total"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="true"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#decimal"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="collapse"  fixed="true"
      id="decimal.whiteSpace"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="duration" id="duration">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasFacet name="pattern"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasFacet name="maxInclusive"/>
      <hfp:hasFacet name="maxExclusive"/>
      <hfp:hasFacet name="minInclusive"/>
      <hfp:hasFacet name="minExclusive"/>
      <hfp:hasProperty name="ordered" value="partial"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#duration"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="collapse"  fixed="true"
```

```
        id="duration.whiteSpace"/>
    </xs:restriction>
   </xs:simpleType>

  <xs:simpleType name="dateTime" id="dateTime">
    <xs:annotation>
    <xs:appinfo>
        <hfp:hasFacet name="pattern"/>
        <hfp:hasFacet name="enumeration"/>
        <hfp:hasFacet name="whiteSpace"/>
        <hfp:hasFacet name="maxInclusive"/>
        <hfp:hasFacet name="maxExclusive"/>
        <hfp:hasFacet name="minInclusive"/>
        <hfp:hasFacet name="minExclusive"/>
        <hfp:hasProperty name="ordered" value="partial"/>
        <hfp:hasProperty name="bounded" value="false"/>
        <hfp:hasProperty name="cardinality"
                value="countably infinite"/>
        <hfp:hasProperty name="numeric" value="false"/>
      </xs:appinfo>
      <xs:documentation
        source="http://www.w3.org/TR/xmlschema-2/#dateTime"/>
    </xs:annotation>
    <xs:restriction base="xs:anySimpleType">
      <xs:whiteSpace value="collapse"  fixed="true"
        id="dateTime.whiteSpace"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="time" id="time">
    <xs:annotation>
    <xs:appinfo>
        <hfp:hasFacet name="pattern"/>
        <hfp:hasFacet name="enumeration"/>
        <hfp:hasFacet name="whiteSpace"/>
        <hfp:hasFacet name="maxInclusive"/>
        <hfp:hasFacet name="maxExclusive"/>
        <hfp:hasFacet name="minInclusive"/>
        <hfp:hasFacet name="minExclusive"/>
        <hfp:hasProperty name="ordered" value="partial"/>
        <hfp:hasProperty name="bounded" value="false"/>
        <hfp:hasProperty name="cardinality"
                value="countably infinite"/>
        <hfp:hasProperty name="numeric" value="false"/>
      </xs:appinfo>
      <xs:documentation
        source="http://www.w3.org/TR/xmlschema-2/#time"/>
    </xs:annotation>
    <xs:restriction base="xs:anySimpleType">
      <xs:whiteSpace value="collapse"  fixed="true"
        id="time.whiteSpace"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="date" id="date">
   <xs:annotation>
    <xs:appinfo>
        <hfp:hasFacet name="pattern"/>
        <hfp:hasFacet name="enumeration"/>
        <hfp:hasFacet name="whiteSpace"/>
        <hfp:hasFacet name="maxInclusive"/>
        <hfp:hasFacet name="maxExclusive"/>
        <hfp:hasFacet name="minInclusive"/>
```

```
      <hfp:hasFacet name="minExclusive"/>
      <hfp:hasProperty name="ordered" value="partial"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#date"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="collapse"  fixed="true"
      id="date.whiteSpace"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="gYearMonth" id="gYearMonth">
 <xs:annotation>
  <xs:appinfo>
      <hfp:hasFacet name="pattern"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasFacet name="maxInclusive"/>
      <hfp:hasFacet name="maxExclusive"/>
      <hfp:hasFacet name="minInclusive"/>
      <hfp:hasFacet name="minExclusive"/>
      <hfp:hasProperty name="ordered" value="partial"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#gYearMonth"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="collapse"  fixed="true"
      id="gYearMonth.whiteSpace"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="gYear" id="gYear">
  <xs:annotation>
  <xs:appinfo>
      <hfp:hasFacet name="pattern"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasFacet name="maxInclusive"/>
      <hfp:hasFacet name="maxExclusive"/>
      <hfp:hasFacet name="minInclusive"/>
      <hfp:hasFacet name="minExclusive"/>
      <hfp:hasProperty name="ordered" value="partial"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#gYear"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="collapse"  fixed="true"
      id="gYear.whiteSpace"/>
```

```
    </xs:restriction>
  </xs:simpleType>

 <xs:simpleType name="gMonthDay" id="gMonthDay">
    <xs:annotation>
      <xs:appinfo>
        <hfp:hasFacet name="pattern"/>
        <hfp:hasFacet name="enumeration"/>
        <hfp:hasFacet name="whiteSpace"/>
        <hfp:hasFacet name="maxInclusive"/>
        <hfp:hasFacet name="maxExclusive"/>
        <hfp:hasFacet name="minInclusive"/>
        <hfp:hasFacet name="minExclusive"/>
        <hfp:hasProperty name="ordered" value="partial"/>
        <hfp:hasProperty name="bounded" value="false"/>
        <hfp:hasProperty name="cardinality"
                value="countably infinite"/>
        <hfp:hasProperty name="numeric" value="false"/>
      </xs:appinfo>
       <xs:documentation
        source="http://www.w3.org/TR/xmlschema-2/#gMonthDay"/>
    </xs:annotation>
    <xs:restriction base="xs:anySimpleType">
         <xs:whiteSpace value="collapse" fixed="true"
                id="gMonthDay.whiteSpace"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="gDay" id="gDay">
    <xs:annotation>
  <xs:appinfo>
        <hfp:hasFacet name="pattern"/>
        <hfp:hasFacet name="enumeration"/>
        <hfp:hasFacet name="whiteSpace"/>
        <hfp:hasFacet name="maxInclusive"/>
        <hfp:hasFacet name="maxExclusive"/>
        <hfp:hasFacet name="minInclusive"/>
        <hfp:hasFacet name="minExclusive"/>
        <hfp:hasProperty name="ordered" value="partial"/>
        <hfp:hasProperty name="bounded" value="false"/>
        <hfp:hasProperty name="cardinality"
                value="countably infinite"/>
        <hfp:hasProperty name="numeric" value="false"/>
      </xs:appinfo>
      <xs:documentation
        source="http://www.w3.org/TR/xmlschema-2/#gDay"/>
    </xs:annotation>
    <xs:restriction base="xs:anySimpleType">
         <xs:whiteSpace value="collapse"  fixed="true"
                id="gDay.whiteSpace"/>
    </xs:restriction>
  </xs:simpleType>

 <xs:simpleType name="gMonth" id="gMonth">
    <xs:annotation>
  <xs:appinfo>
        <hfp:hasFacet name="pattern"/>
        <hfp:hasFacet name="enumeration"/>
        <hfp:hasFacet name="whiteSpace"/>
        <hfp:hasFacet name="maxInclusive"/>
        <hfp:hasFacet name="maxExclusive"/>
        <hfp:hasFacet name="minInclusive"/>
        <hfp:hasFacet name="minExclusive"/>
```

```
            <hfp:hasProperty name="ordered" value="partial"/>
            <hfp:hasProperty name="bounded" value="false"/>
            <hfp:hasProperty name="cardinality"
                    value="countably infinite"/>
            <hfp:hasProperty name="numeric" value="false"/>
          </xs:appinfo>
          <xs:documentation
            source="http://www.w3.org/TR/xmlschema-2/#gMonth"/>
        </xs:annotation>
        <xs:restriction base="xs:anySimpleType">
             <xs:whiteSpace value="collapse"  fixed="true"
                    id="gMonth.whiteSpace"/>
        </xs:restriction>
      </xs:simpleType>

       <xs:simpleType name="hexBinary" id="hexBinary">
        <xs:annotation>
          <xs:appinfo>
            <hfp:hasFacet name="length"/>
            <hfp:hasFacet name="minLength"/>
            <hfp:hasFacet name="maxLength"/>
            <hfp:hasFacet name="pattern"/>
            <hfp:hasFacet name="enumeration"/>
            <hfp:hasFacet name="whiteSpace"/>
            <hfp:hasProperty name="ordered" value="false"/>
            <hfp:hasProperty name="bounded" value="false"/>
            <hfp:hasProperty name="cardinality"
                    value="countably infinite"/>
            <hfp:hasProperty name="numeric" value="false"/>
          </xs:appinfo>
          <xs:documentation
            source="http://www.w3.org/TR/xmlschema-2/#binary"/>
        </xs:annotation>
        <xs:restriction base="xs:anySimpleType">
          <xs:whiteSpace value="collapse" fixed="true"
            id="hexBinary.whiteSpace"/>
        </xs:restriction>
       </xs:simpleType>

     <xs:simpleType name="base64Binary" id="base64Binary">
        <xs:annotation>
          <xs:appinfo>
            <hfp:hasFacet name="length"/>
            <hfp:hasFacet name="minLength"/>
            <hfp:hasFacet name="maxLength"/>
            <hfp:hasFacet name="pattern"/>
            <hfp:hasFacet name="enumeration"/>
            <hfp:hasFacet name="whiteSpace"/>
            <hfp:hasProperty name="ordered" value="false"/>
            <hfp:hasProperty name="bounded" value="false"/>
            <hfp:hasProperty name="cardinality"
                    value="countably infinite"/>
            <hfp:hasProperty name="numeric" value="false"/>
          </xs:appinfo>
          <xs:documentation
                    source="http://www.w3.org/TR/xmlschema-2/#base64Binary"/>
        </xs:annotation>
        <xs:restriction base="xs:anySimpleType">
          <xs:whiteSpace value="collapse" fixed="true"
            id="base64Binary.whiteSpace"/>
        </xs:restriction>
       </xs:simpleType>
```

```
 <xs:simpleType name="anyURI" id="anyURI">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasFacet name="length"/>
      <hfp:hasFacet name="minLength"/>
      <hfp:hasFacet name="maxLength"/>
      <hfp:hasFacet name="pattern"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasProperty name="ordered" value="false"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#anyURI"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="collapse"  fixed="true"
      id="anyURI.whiteSpace"/>
  </xs:restriction>
 </xs:simpleType>

<xs:simpleType name="QName" id="QName">
  <xs:annotation>
      <xs:appinfo>
      <hfp:hasFacet name="length"/>
      <hfp:hasFacet name="minLength"/>
      <hfp:hasFacet name="maxLength"/>
      <hfp:hasFacet name="pattern"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasProperty name="ordered" value="false"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#QName"/>
  </xs:annotation>
  <xs:restriction base="xs:anySimpleType">
    <xs:whiteSpace value="collapse"  fixed="true"
      id="QName.whiteSpace"/>
  </xs:restriction>
</xs:simpleType>

 <xs:simpleType name="NOTATION" id="NOTATION">
  <xs:annotation>
      <xs:appinfo>
      <hfp:hasFacet name="length"/>
      <hfp:hasFacet name="minLength"/>
      <hfp:hasFacet name="maxLength"/>
      <hfp:hasFacet name="pattern"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasProperty name="ordered" value="false"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
```

```
        <xs:documentation
          source="http://www.w3.org/TR/xmlschema-2/#NOTATION"/>
        <xs:documentation>
          NOTATION cannot be used directly in a schema; rather a type
          must be derived from it by specifying at least one enumeration
          facet whose value is the name of a NOTATION declared in the
          schema.
        </xs:documentation>
      </xs:annotation>
      <xs:restriction base="xs:anySimpleType">
        <xs:whiteSpace value="collapse"  fixed="true"
          id="NOTATION.whiteSpace"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:annotation>
      <xs:documentation>
        Now the derived primitive types
      </xs:documentation>
    </xs:annotation>

    <xs:simpleType name="normalizedString" id="normalizedString">
      <xs:annotation>
        <xs:documentation
          source="http://www.w3.org/TR/xmlschema-2/#normalizedString"/>
      </xs:annotation>
      <xs:restriction base="xs:string">
        <xs:whiteSpace value="replace"
          id="normalizedString.whiteSpace"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="token" id="token">
      <xs:annotation>
        <xs:documentation
          source="http://www.w3.org/TR/xmlschema-2/#token"/>
      </xs:annotation>
      <xs:restriction base="xs:normalizedString">
        <xs:whiteSpace value="collapse" id="token.whiteSpace"/>
      </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="language" id="language">
      <xs:annotation>
        <xs:documentation
          source="http://www.w3.org/TR/xmlschema-2/#language"/>
      </xs:annotation>
      <xs:restriction base="xs:token">
        <xs:pattern
          value="([a-zA-Z]{2}|[iI]-[a-zA-Z]+|[xX]-[a-zA-Z]{1,8})(-[a-zA-Z]{1,8})
*"
                id="language.pattern">
          <xs:annotation>
            <xs:documentation
                  source="http://www.w3.org/TR/REC-xml#NT-LanguageID">
              pattern specifies the content of section 2.12 of XML 1.0e2
              and RFC 1766
            </xs:documentation>
          </xs:annotation>
        </xs:pattern>
      </xs:restriction>
    </xs:simpleType>
```

```
<xs:simpleType name="IDREFS" id="IDREFS">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasFacet name="length"/>
      <hfp:hasFacet name="minLength"/>
      <hfp:hasFacet name="maxLength"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasProperty name="ordered" value="false"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#IDREFS"/>
  </xs:annotation>
  <xs:restriction>
    <xs:simpleType>
      <xs:list itemType="xs:IDREF"/>
    </xs:simpleType>
      <xs:minLength value="1" id="IDREFS.minLength"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="ENTITIES" id="ENTITIES">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasFacet name="length"/>
      <hfp:hasFacet name="minLength"/>
      <hfp:hasFacet name="maxLength"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasProperty name="ordered" value="false"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#ENTITIES"/>
  </xs:annotation>
  <xs:restriction>
    <xs:simpleType>
      <xs:list itemType="xs:ENTITY"/>
    </xs:simpleType>
      <xs:minLength value="1" id="ENTITIES.minLength"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="NMTOKEN" id="NMTOKEN">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#NMTOKEN"/>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:pattern value="\c+" id="NMTOKEN.pattern">
      <xs:annotation>
        <xs:documentation
              source="http://www.w3.org/TR/REC-xml#NT-Nmtoken">
          pattern matches production 7 from the XML spec
        </xs:documentation>
      </xs:annotation>
```

```
    </xs:pattern>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="NMTOKENS" id="NMTOKENS">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasFacet name="length"/>
      <hfp:hasFacet name="minLength"/>
      <hfp:hasFacet name="maxLength"/>
      <hfp:hasFacet name="enumeration"/>
      <hfp:hasFacet name="whiteSpace"/>
      <hfp:hasProperty name="ordered" value="false"/>
      <hfp:hasProperty name="bounded" value="false"/>
      <hfp:hasProperty name="cardinality"
              value="countably infinite"/>
      <hfp:hasProperty name="numeric" value="false"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#NMTOKENS"/>
  </xs:annotation>
  <xs:restriction>
    <xs:simpleType>
      <xs:list itemType="xs:NMTOKEN"/>
    </xs:simpleType>
      <xs:minLength value="1" id="NMTOKENS.minLength"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="Name" id="Name">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#Name"/>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:pattern value="\i\c*" id="Name.pattern">
      <xs:annotation>
        <xs:documentation
                    source="http://www.w3.org/TR/REC-xml#NT-Name">
          pattern matches production 5 from the XML spec
        </xs:documentation>
      </xs:annotation>
    </xs:pattern>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="NCName" id="NCName">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#NCName"/>
  </xs:annotation>
  <xs:restriction base="xs:Name">
    <xs:pattern value="[\i-[:]][\c-[:]]*" id="NCName.pattern">
      <xs:annotation>
        <xs:documentation
              source="http://www.w3.org/TR/REC-xml-names/#NT-NCName">
          pattern matches production 4 from the Namespaces in XML spec
        </xs:documentation>
      </xs:annotation>
    </xs:pattern>
  </xs:restriction>
</xs:simpleType>
```

```
 <xs:simpleType name="ID" id="ID">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#ID"/>
  </xs:annotation>
  <xs:restriction base="xs:NCName"/>
 </xs:simpleType>

 <xs:simpleType name="IDREF" id="IDREF">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#IDREF"/>
  </xs:annotation>
  <xs:restriction base="xs:NCName"/>
 </xs:simpleType>

 <xs:simpleType name="ENTITY" id="ENTITY">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#ENTITY"/>
  </xs:annotation>
  <xs:restriction base="xs:NCName"/>
 </xs:simpleType>

<xs:simpleType name="integer" id="integer">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#integer"/>
  </xs:annotation>
  <xs:restriction base="xs:decimal">
    <xs:fractionDigits value="0" fixed="true" id="integer.fractionDigits"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="nonPositiveInteger" id="nonPositiveInteger">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#nonPositiveInteger"/>
  </xs:annotation>
  <xs:restriction base="xs:integer">
    <xs:maxInclusive value="0" id="nonPositiveInteger.maxInclusive"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="negativeInteger" id="negativeInteger">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#negativeInteger"/>
  </xs:annotation>
  <xs:restriction base="xs:nonPositiveInteger">
    <xs:maxInclusive value="-1" id="negativeInteger.maxInclusive"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="long" id="long">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasProperty name="bounded" value="true"/>
      <hfp:hasProperty name="cardinality" value="finite"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#long"/>
  </xs:annotation>
```

```
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="-9223372036854775808" id="long.minInclusive"/>
    <xs:maxInclusive value="9223372036854775807" id="long.maxInclusive"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="int" id="int">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#int"/>
  </xs:annotation>
  <xs:restriction base="xs:long">
    <xs:minInclusive value="-2147483648" id="int.minInclusive"/>
    <xs:maxInclusive value="2147483647" id="int.maxInclusive"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="short" id="short">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#short"/>
  </xs:annotation>
  <xs:restriction base="xs:int">
    <xs:minInclusive value="-32768" id="short.minInclusive"/>
    <xs:maxInclusive value="32767" id="short.maxInclusive"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="byte" id="byte">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#byte"/>
  </xs:annotation>
  <xs:restriction base="xs:short">
    <xs:minInclusive value="-128" id="byte.minInclusive"/>
    <xs:maxInclusive value="127" id="byte.maxInclusive"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="nonNegativeInteger" id="nonNegativeInteger">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#nonNegativeInteger"/>
  </xs:annotation>
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0" id="nonNegativeInteger.minInclusive"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="unsignedLong" id="unsignedLong">
  <xs:annotation>
    <xs:appinfo>
      <hfp:hasProperty name="bounded" value="true"/>
      <hfp:hasProperty name="cardinality" value="finite"/>
    </xs:appinfo>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#unsignedLong"/>
  </xs:annotation>
  <xs:restriction base="xs:nonNegativeInteger">
    <xs:maxInclusive value="18446744073709551615"
      id="unsignedLong.maxInclusive"/>
  </xs:restriction>
</xs:simpleType>
```

```
 <xs:simpleType name="unsignedInt" id="unsignedInt">
   <xs:annotation>
     <xs:documentation
       source="http://www.w3.org/TR/xmlschema-2/#unsignedInt"/>
   </xs:annotation>
   <xs:restriction base="xs:unsignedLong">
     <xs:maxInclusive value="4294967295"
       id="unsignedInt.maxInclusive"/>
   </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="unsignedShort" id="unsignedShort">
   <xs:annotation>
     <xs:documentation
       source="http://www.w3.org/TR/xmlschema-2/#unsignedShort"/>
   </xs:annotation>
   <xs:restriction base="xs:unsignedInt">
     <xs:maxInclusive value="65535"
       id="unsignedShort.maxInclusive"/>
   </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="unsignedByte" id="unsignedBtype">
   <xs:annotation>
     <xs:documentation
       source="http://www.w3.org/TR/xmlschema-2/#unsignedByte"/>
   </xs:annotation>
   <xs:restriction base="xs:unsignedShort">
     <xs:maxInclusive value="255" id="unsignedByte.maxInclusive"/>
   </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="positiveInteger" id="positiveInteger">
   <xs:annotation>
     <xs:documentation
       source="http://www.w3.org/TR/xmlschema-2/#positiveInteger"/>
   </xs:annotation>
   <xs:restriction base="xs:nonNegativeInteger">
     <xs:minInclusive value="1" id="positiveInteger.minInclusive"/>
   </xs:restriction>
 </xs:simpleType>

<xs:simpleType name="derivationControl">
 <xs:annotation>
  <xs:documentation>
  A utility type, not for public use</xs:documentation>
 </xs:annotation>
 <xs:restriction base="xs:NMTOKEN">
  <xs:enumeration value="substitution"/>
  <xs:enumeration value="extension"/>
  <xs:enumeration value="restriction"/>
  <xs:enumeration value="list"/>
  <xs:enumeration value="union"/>
 </xs:restriction>
</xs:simpleType>

<xs:group name="simpleDerivation">
 <xs:choice>
   <xs:element ref="xs:restriction"/>
   <xs:element ref="xs:list"/>
   <xs:element ref="xs:union"/>
 </xs:choice>
```

```
</xs:group>

<xs:simpleType name="simpleDerivationSet">
 <xs:annotation>
  <xs:documentation>
  #all or (possibly empty) subset of {restriction, union, list}
  </xs:documentation>
  <xs:documentation>
  A utility type, not for public use</xs:documentation>
 </xs:annotation>
 <xs:union>
  <xs:simpleType>
   <xs:restriction base="xs:token">
    <xs:enumeration value="#all"/>
   </xs:restriction>
  </xs:simpleType>
  <xs:simpleType>
   <xs:restriction base="xs:derivationControl">
    <xs:enumeration value="list"/>
    <xs:enumeration value="union"/>
    <xs:enumeration value="restriction"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:union>
</xs:simpleType>

 <xs:complexType name="simpleType" abstract="true">
   <xs:complexContent>
     <xs:extension base="xs:annotated">
       <xs:group ref="xs:simpleDerivation"/>
       <xs:attribute name="final" type="xs:simpleDerivationSet"/>
       <xs:attribute name="name" type="xs:NCName">
         <xs:annotation>
           <xs:documentation>
             Can be restricted to required or forbidden
           </xs:documentation>
         </xs:annotation>
       </xs:attribute>
     </xs:extension>
   </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="topLevelSimpleType">
   <xs:complexContent>
     <xs:restriction base="xs:simpleType">
       <xs:sequence>
         <xs:element ref="xs:annotation" minOccurs="0"/>
         <xs:group ref="xs:simpleDerivation"/>
       </xs:sequence>
       <xs:attribute name="name" use="required"
            type="xs:NCName">
         <xs:annotation>
           <xs:documentation>
             Required at the top level
           </xs:documentation>
         </xs:annotation>
       </xs:attribute>
     </xs:restriction>
   </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="localSimpleType">
   <xs:complexContent>
```

```
    <xs:restriction base="xs:simpleType">
      <xs:sequence>
        <xs:element ref="xs:annotation" minOccurs="0"/>
        <xs:group ref="xs:simpleDerivation"/>
      </xs:sequence>
      <xs:attribute name="name" use="prohibited">
        <xs:annotation>
          <xs:documentation>
            Forbidden when nested
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="final" use="prohibited"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:element name="simpleType" type="xs:topLevelSimpleType" id="simpleType">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#element-simpleType"/>
  </xs:annotation>
</xs:element>

<xs:group name="facets">
 <xs:annotation>
  <xs:documentation>
     We should use a substitution group for facets, but
     that's ruled out because it would allow users to
     add their own, which we're not ready for yet.
  </xs:documentation>
 </xs:annotation>
 <xs:choice>
  <xs:element ref="xs:minExclusive"/>
  <xs:element ref="xs:minInclusive"/>
  <xs:element ref="xs:maxExclusive"/>
  <xs:element ref="xs:maxInclusive"/>
  <xs:element ref="xs:totalDigits"/>
  <xs:element ref="xs:fractionDigits"/>
  <xs:element ref="xs:length"/>
  <xs:element ref="xs:minLength"/>
  <xs:element ref="xs:maxLength"/>
  <xs:element ref="xs:enumeration"/>
  <xs:element ref="xs:whiteSpace"/>
  <xs:element ref="xs:pattern"/>
 </xs:choice>
</xs:group>

<xs:group name="simpleRestrictionModel">
 <xs:sequence>
  <xs:element name="simpleType" type="xs:localSimpleType" minOccurs="0"/>
  <xs:group ref="xs:facets" minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
</xs:group>

<xs:element name="restriction" id="restriction">
 <xs:complexType>
  <xs:annotation>
    <xs:documentation
              source="http://www.w3.org/TR/xmlschema-2/#element-restriction">
      base attribute and simpleType child are mutually
      exclusive, but one or other is required
    </xs:documentation>
```

```
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="xs:annotated">
       <xs:group ref="xs:simpleRestrictionModel"/>
       <xs:attribute name="base" type="xs:QName" use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="list" id="list">
 <xs:complexType>
  <xs:annotation>
    <xs:documentation
             source="http://www.w3.org/TR/xmlschema-2/#element-list">
       itemType attribute and simpleType child are mutually
       exclusive, but one or other is required
      </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="xs:annotated">
        <xs:sequence>
          <xs:element name="simpleType" type="xs:localSimpleType"
              minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="itemType" type="xs:QName" use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="union" id="union">
 <xs:complexType>
  <xs:annotation>
    <xs:documentation
             source="http://www.w3.org/TR/xmlschema-2/#element-union">
       memberTypes attribute must be non-empty or there must be
       at least one simpleType child
      </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="xs:annotated">
        <xs:sequence>
          <xs:element name="simpleType" type="xs:localSimpleType"
              minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="memberTypes" use="optional">
          <xs:simpleType>
            <xs:list itemType="xs:QName"/>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:complexType name="facet">
  <xs:complexContent>
    <xs:extension base="xs:annotated">
      <xs:attribute name="value" use="required"/>
      <xs:attribute name="fixed" type="xs:boolean" use="optional"
                    default="false"/>
    </xs:extension>
```

```
    </xs:complexContent>
   </xs:complexType>

  <xs:complexType name="noFixedFacet">
   <xs:complexContent>
    <xs:restriction base="xs:facet">
     <xs:sequence>
      <xs:element ref="xs:annotation" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="fixed" use="prohibited"/>
    </xs:restriction>
   </xs:complexContent>
  </xs:complexType>

   <xs:element name="minExclusive" id="minExclusive" type="xs:facet">
     <xs:annotation>
       <xs:documentation
         source="http://www.w3.org/TR/xmlschema-2/#element-minExclusive"/>
     </xs:annotation>
   </xs:element>
   <xs:element name="minInclusive" id="minInclusive" type="xs:facet">
     <xs:annotation>
       <xs:documentation
         source="http://www.w3.org/TR/xmlschema-2/#element-minInclusive"/>
     </xs:annotation>
   </xs:element>

   <xs:element name="maxExclusive" id="maxExclusive" type="xs:facet">
     <xs:annotation>
       <xs:documentation
         source="http://www.w3.org/TR/xmlschema-2/#element-maxExclusive"/>
     </xs:annotation>
   </xs:element>
   <xs:element name="maxInclusive" id="maxInclusive" type="xs:facet">
     <xs:annotation>
       <xs:documentation
         source="http://www.w3.org/TR/xmlschema-2/#element-maxInclusive"/>
     </xs:annotation>
   </xs:element>

   <xs:complexType name="numFacet">
     <xs:complexContent>
       <xs:restriction base="xs:facet">
        <xs:sequence>
          <xs:element ref="xs:annotation" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="value" type="xs:nonNegativeInteger" use="required"/>
       </xs:restriction>
     </xs:complexContent>
   </xs:complexType>

   <xs:element name="totalDigits" id="totalDigits">
     <xs:annotation>
       <xs:documentation
         source="http://www.w3.org/TR/xmlschema-2/#element-totalDigits"/>
     </xs:annotation>
     <xs:complexType>
       <xs:complexContent>
         <xs:restriction base="xs:numFacet">
           <xs:sequence>
             <xs:element ref="xs:annotation" minOccurs="0"/>
           </xs:sequence>
           <xs:attribute name="value" type="xs:positiveInteger" use="required"/>
```

```
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="fractionDigits" id="fractionDigits" type="xs:numFacet">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#element-fractionDigits"/>
  </xs:annotation>
</xs:element>

<xs:element name="length" id="length" type="xs:numFacet">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#element-length"/>
  </xs:annotation>
</xs:element>
<xs:element name="minLength" id="minLength" type="xs:numFacet">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#element-minLength"/>
  </xs:annotation>
</xs:element>
<xs:element name="maxLength" id="maxLength" type="xs:numFacet">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#element-maxLength"/>
  </xs:annotation>
</xs:element>

<xs:element name="enumeration" id="enumeration" type="xs:noFixedFacet">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#element-enumeration"/>
  </xs:annotation>
</xs:element>

<xs:element name="whiteSpace" id="whiteSpace">
  <xs:annotation>
    <xs:documentation
      source="http://www.w3.org/TR/xmlschema-2/#element-whiteSpace"/>
  </xs:annotation>
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:facet">
        <xs:sequence>
          <xs:element ref="xs:annotation" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="value" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
              <xs:enumeration value="preserve"/>
              <xs:enumeration value="replace"/>
              <xs:enumeration value="collapse"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>

<xs:element name="pattern" id="pattern" type="xs:noFixedFacet">
```

```
    <xs:annotation>
      <xs:documentation
        source="http://www.w3.org/TR/xmlschema-2/#element-pattern"/>
    </xs:annotation>
  </xs:element>

</xs:schema>
```

B DTD for Datatype Definitions (non-normative)

```
<!--
        DTD for XML Schemas: Part 2: Datatypes
        Id: datatypes.dtd,v 1.23 2001/03/16 17:36:30 ht Exp
        Note this DTD is NOT normative, or even definitive.
  -->

<!--
        This DTD cannot be used on its own, it is intended
        only for incorporation in XMLSchema.dtd, q.v.
  -->

<!-- Define all the element names, with optional prefix -->
<!ENTITY % simpleType "%p;simpleType">
<!ENTITY % restriction "%p;restriction">
<!ENTITY % list "%p;list">
<!ENTITY % union "%p;union">
<!ENTITY % maxExclusive "%p;maxExclusive">
<!ENTITY % minExclusive "%p;minExclusive">
<!ENTITY % maxInclusive "%p;maxInclusive">
<!ENTITY % minInclusive "%p;minInclusive">
<!ENTITY % totalDigits "%p;totalDigits">
<!ENTITY % fractionDigits "%p;fractionDigits">
<!ENTITY % length "%p;length">
<!ENTITY % minLength "%p;minLength">
<!ENTITY % maxLength "%p;maxLength">
<!ENTITY % enumeration "%p;enumeration">
<!ENTITY % whiteSpace "%p;whiteSpace">
<!ENTITY % pattern "%p;pattern">

<!--
        Customisation entities for the ATTLIST of each element
        type. Define one of these if your schema takes advantage
        of the anyAttribute='##other' in the schema for schemas
  -->

<!ENTITY % simpleTypeAttrs "">
<!ENTITY % restrictionAttrs "">
<!ENTITY % listAttrs "">
<!ENTITY % unionAttrs "">
<!ENTITY % maxExclusiveAttrs "">
<!ENTITY % minExclusiveAttrs "">
<!ENTITY % maxInclusiveAttrs "">
<!ENTITY % minInclusiveAttrs "">
<!ENTITY % totalDigitsAttrs "">
<!ENTITY % fractionDigitsAttrs "">
<!ENTITY % lengthAttrs "">
<!ENTITY % minLengthAttrs "">
<!ENTITY % maxLengthAttrs "">
<!ENTITY % enumerationAttrs "">
<!ENTITY % whiteSpaceAttrs "">
<!ENTITY % patternAttrs "">

<!-- Define some entities for informative use as attribute
```

```
        types -->
<!ENTITY % URIref "CDATA">
<!ENTITY % XPathExpr "CDATA">
<!ENTITY % QName "NMTOKEN">
<!ENTITY % QNames "NMTOKENS">
<!ENTITY % NCName "NMTOKEN">
<!ENTITY % nonNegativeInteger "NMTOKEN">
<!ENTITY % boolean "(true|false)">
<!ENTITY % simpleDerivationSet "CDATA">
<!--
        #all or space-separated list drawn from derivationChoice
  -->

<!--
        Note that the use of 'facet' below is less restrictive
        than is really intended:  There should in fact be no
        more than one of each of minInclusive, minExclusive,
        maxInclusive, maxExclusive, totalDigits, fractionDigits,
        length, maxLength, minLength within datatype,
        and the min- and max- variants of Inclusive and Exclusive
        are mutually exclusive. On the other hand,  pattern and
        enumeration may repeat.
  -->
<!ENTITY % minBound "(%minInclusive; | %minExclusive;)">
<!ENTITY % maxBound "(%maxInclusive; | %maxExclusive;)">
<!ENTITY % bounds "%minBound; | %maxBound;">
<!ENTITY % numeric "%totalDigits; | %fractionDigits;">
<!ENTITY % ordered "%bounds; | %numeric;">
<!ENTITY % unordered
   "%pattern; | %enumeration; | %whiteSpace; | %length; |
   %maxLength; | %minLength;">
<!ENTITY % facet "%ordered; | %unordered;">
<!ENTITY % facetAttr
        "value CDATA #REQUIRED
        id ID #IMPLIED">
<!ENTITY % fixedAttr "fixed %boolean; #IMPLIED">
<!ENTITY % facetModel "(%annotation;)?">
<!ELEMENT %simpleType;
        ((%annotation;)?, (%restriction; | %list; | %union;))>
<!ATTLIST %simpleType;
    name      %NCName; #IMPLIED
    final     %simpleDerivationSet; #IMPLIED
    id        ID       #IMPLIED
    %simpleTypeAttrs;>
<!-- name is required at top level -->
<!ELEMENT %restriction; ((%annotation;)?,
                         (%restriction1; |
                          ((%simpleType;)?,(%facet;)*)),
                         (%attrDecls;))>
<!ATTLIST %restriction;
    base      %QName;                  #IMPLIED
    id        ID       #IMPLIED
    %restrictionAttrs;>
<!--
        base and simpleType child are mutually exclusive,
        one is required.

        restriction is shared between simpleType and
        simpleContent and complexContent (in XMLSchema.xsd).
        restriction1 is for the latter cases, when this
        is restricting a complex type, as is attrDecls.
  -->
<!ELEMENT %list; ((%annotation;)?,(%simpleType;)?)>
```

```
<!ATTLIST %list;
    itemType      %QName;             #IMPLIED
    id        ID       #IMPLIED
    %listAttrs;>
<!--
        itemType and simpleType child are mutually exclusive,
        one is required
  -->
<!ELEMENT %union; ((%annotation;)?,(%simpleType;)*)>
<!ATTLIST %union;
    id            ID       #IMPLIED
    memberTypes   %QNames;            #IMPLIED
    %unionAttrs;>
<!--
        At least one item in memberTypes or one simpleType
        child is required
  -->

<!ELEMENT %maxExclusive; %facetModel;>
<!ATTLIST %maxExclusive;
        %facetAttr;
        %fixedAttr;
        %maxExclusiveAttrs;>
<!ELEMENT %minExclusive; %facetModel;>
<!ATTLIST %minExclusive;
        %facetAttr;
        %fixedAttr;
        %minExclusiveAttrs;>

<!ELEMENT %maxInclusive; %facetModel;>
<!ATTLIST %maxInclusive;
        %facetAttr;
        %fixedAttr;
        %maxInclusiveAttrs;>
<!ELEMENT %minInclusive; %facetModel;>
<!ATTLIST %minInclusive;
        %facetAttr;
        %fixedAttr;
        %minInclusiveAttrs;>

<!ELEMENT %totalDigits; %facetModel;>
<!ATTLIST %totalDigits;
        %facetAttr;
        %fixedAttr;
        %totalDigitsAttrs;>
<!ELEMENT %fractionDigits; %facetModel;>
<!ATTLIST %fractionDigits;
        %facetAttr;
        %fixedAttr;
        %fractionDigitsAttrs;>

<!ELEMENT %length; %facetModel;>
<!ATTLIST %length;
        %facetAttr;
        %fixedAttr;
        %lengthAttrs;>
<!ELEMENT %minLength; %facetModel;>
<!ATTLIST %minLength;
        %facetAttr;
        %fixedAttr;
        %minLengthAttrs;>
<!ELEMENT %maxLength; %facetModel;>
<!ATTLIST %maxLength;
```

```
            %facetAttr;
            %fixedAttr;
            %maxLengthAttrs;>

    <!-- This one can be repeated -->
    <!ELEMENT %enumeration; %facetModel;>
    <!ATTLIST %enumeration;
            %facetAttr;
            %enumerationAttrs;>

    <!ELEMENT %whiteSpace; %facetModel;>
    <!ATTLIST %whiteSpace;
            %facetAttr;
            %fixedAttr;
            %whiteSpaceAttrs;>

    <!-- This one can be repeated -->
    <!ELEMENT %pattern; %facetModel;>
    <!ATTLIST %pattern;
            %facetAttr;
            %patternAttrs;>
```

C Datatypes and Facets

C.1 Fundamental Facets

The following table shows the values of the fundamental facets for each ·built-in· datatype.

	Datatype	ordered	bounded	cardinality	numeric
primitive	string	false	false	countably infinite	false
	boolean	false	false	finite	false
	float	total	true	finite	true
	double	total	true	finite	true
	decimal	total	false	countably infinite	true
	duration	partial	false	countably infinite	false
	dateTime	partial	false	countably infinite	false
	time	partial	false	countably infinite	false
	date	partial	false	countably infinite	false
	gYearMonth	partial	false	countably infinite	false
	gYear	partial	false	countably infinite	false
	gMonthDay	partial	false	countably infinite	false
	gDay	partial	false	countably infinite	false
	gMonth	partial	false	countably infinite	false
	hexBinary	false	false	countably infinite	false
	base64Binary	false	false	countably infinite	false
	anyURI	false	false	countably infinite	false
	QName	false	false	countably infinite	false
	NOTATION	false	false	countably infinite	false
	normalizedString	false	false	countably infinite	false
	token	false	false	countably infinite	false
	language	false	false	countably infinite	false

	IDREFS	false	false	countably infinite	false
	ENTITIES	false	false	countably infinite	false
	NMTOKEN	false	false	countably infinite	false
	NMTOKENS	false	false	countably infinite	false
	Name	false	false	countably infinite	false
	NCName	false	false	countably infinite	false
	ID	false	false	countably infinite	false
	IDREF	false	false	countably infinite	false
	ENTITY	false	false	countably infinite	false
derived	integer	total	false	countably infinite	true
	nonPositiveInteger	total	false	countably infinite	true
	negativeInteger	total	false	countably infinite	true
	long	total	true	finite	true
	int	total	true	finite	true
	short	total	true	finite	true
	byte	total	true	finite	true
	nonNegativeInteger	total	false	countably infinite	true
	unsignedLong	total	true	finite	true
	unsignedInt	total	true	finite	true
	unsignedShort	total	true	finite	true
	unsignedByte	total	true	finite	true
	positiveInteger	total	false	countably infinite	true

D ISO 8601 Date and Time Formats

D.1 ISO 8601 Conventions

The ·primitive· datatypes duration, dateTime, time, date, gYearMonth, gMonthDay, gDay, gMonth and gYear use lexical formats inspired by [ISO 8601]. This appendix provides more detail on the ISO formats and discusses some deviations from them for the datatypes defined in this specification.

[ISO 8601] "specifies the representation of dates in the proleptic Gregorian calendar and times and representations of periods of time". The proleptic Gregorian calendar includes dates prior to 1582 (the year it came into use as an ecclesiastical calendar). It should be pointed out that the datatypes described in this specification do not cover all the types of data covered by [ISO 8601], nor do they support all the lexical representations for those types of data.

[ISO 8601] lexical formats are described using "pictures" in which characters are used in place of digits. For the primitive datatypes dateTime, time, date, gYearMonth, gMonthDay, gDay, gMonth and gYear. these characters have the following meanings:

- C -- represents a digit used in the thousands and hundreds components, the "century" component, of the time element "year". Legal values are from 0 to 9.
- Y -- represents a digit used in the tens and units components of the time element "year". Legal values are from 0 to 9.
- M -- represents a digit used in the time element "month". The two digits in a MM format can have values from 1 to 12.
- D -- represents a digit used in the time element "day". The two digits in a DD format can have values from 1 to 28 if the month value equals 2, 1 to 29 if the month value equals 2 and the year is a leap year, 1 to 30 if the month value equals 4, 6, 9 or 11, and 1 to 31 if the month value equals 1, 3, 5, 7, 8, 10 or 12.

- h -- represents a digit used in the time element "hour". The two digits in a hh format can have values from 0 to 23.
- m -- represents a digit used in the time element "minute". The two digits in a mm format can have values from 0 to 59.
- s -- represents a digit used in the time element "second". The two digits in a ss format can have values from 0 to 60. In the formats described in this specification the whole number of seconds ·may· be followed by decimal seconds to an arbitrary level of precision. This is represented in the picture by "ss.sss". A value of 60 or more is allowed only in the case of leap seconds.

  Strictly speaking, a value of 60 or more is not sensible unless the month and day could represent March 31, June 30, September 30, or December 31 *in UTC*. Because the leap second is added or subtracted as the last second of the day in UTC time, the long (or short) minute could occur at other times in local time. In cases where the leap second is used with an inappropriate month and day it, and any fractional seconds, should considered as added or subtracted from the following minute.

For all the information items indicated by the above characters, leading zeros are required where indicated.

In addition to the above, certain characters are used as designators and appear as themselves in lexical formats.

- T -- is used as time designator to indicate the start of the representation of the time of day in dateTime.
- Z -- is used as time-zone designator, immediately (without a space) following a data element expressing the time of day in Coordinated Universal Time (UTC) in dateTime, time, date, gYearMonth, gMonthDay, gDay, gMonth, and gYear.

In the lexical format for duration the following characters are also used as designators and appear as themselves in lexical formats:

- P -- is used as the time duration designator, preceding a data element representing a given duration of time.
- Y -- follows the number of years in a time duration.
- M -- follows the number of months or minutes in a time duration.
- D -- follows the number of days in a time duration.
- H -- follows the number of hours in a time duration.
- S -- follows the number of seconds in a time duration.

The values of the Year, Month, Day, Hour and Minutes components are not restricted but allow an arbitrary integer. Similarly, the value of the Seconds component allows an arbitrary decimal. Thus, the lexical format for duration and datatypes derived from it does not follow the alternative format of § 5.5.3.2.1 of [ISO 8601].

D.2 Truncated and Reduced Formats

[ISO 8601] supports a variety of "truncated" formats in which some of the characters on the left of specific formats, for example, the century, can be omitted. Truncated formats are, in general, not permitted for the datatypes defined in this specification with three exceptions. The time datatype uses a truncated format for dateTime which represents an instant of time that recurs every day. Similarly, the gMonthDay and gDay datatypes use left-truncated formats for date. The datatype gMonth uses a right and left truncated format for date.

[ISO 8601] also supports a variety of "reduced" or right-truncated formats in which some of the characters to the right of specific formats, such as the time specification, can be omitted. Right truncated formats are also, in general, not permitted for the datatypes defined in this specification with the following exceptions: right-truncated representations of dateTime are used as lexical representations for date, gMonth, gYear.

D.3 Deviations from ISO 8601 Formats

D.3.1 Sign Allowed

An optional minus sign is allowed immediately preceding, without a space, the lexical representations for duration, dateTime, date, gMonth, gYear.

D.3.2 No Year Zero

The year "0000" is an illegal year value.

D.3.3 More Than 9999 Years

To accommodate year values greater than 9999, more than four digits are allowed in the year representations of dateTime, date, gYearMonth, and gYear. This follows [ISO 8601 Draft Revision].

E Adding durations to dateTimes

Given a dateTime S and a duration D, this appendix specifies how to compute a dateTime E where E is the end of the time period with start S and duration D i.e. E = S + D. Such computations are used, for example, to determine whether a dateTime is within a specific time period. This appendix also addresses the addition of durations to the datatypes date, gYearMonth, gYear, gDay and gMonth, which can be viewed as a set of dateTimes. In such cases, the addition is made to the first or starting dateTime in the set.

This is a logical explanation of the process. Actual implementations are free to optimize as long as they produce the same results. The calculation uses the notation S[year] to represent the year field of S, S[month] to represent the month field, and so on. It also depends on the following functions:

- fQuotient(a, b) = the greatest integer less than or equal to a/b
  - fQuotient(-1,3) = -1
  - fQuotient(0,3)...fQuotient(2,3) = 0
  - fQuotient(3,3) = 1
  - fQuotient(3.123,3) = 1
- modulo(a, b) = a - fQuotient(a,b)*b
  - modulo(-1,3) = 2
  - modulo(0,3)...modulo(2,3) = 0...2
  - modulo(3,3) = 0
  - modulo(3.123,3) = 0.123
- fQuotient(a, low, high) = fQuotient(a - low, high - low)
  - fQuotient(0, 1, 13) = -1
  - fQuotient(1, 1, 13) ... fQuotient(12, 1, 13) = 0
  - fQuotient(13, 1, 13) = 1
  - fQuotient(13.123, 1, 13) = 1
- modulo(a, low, high) = modulo(a - low, high - low) + low
  - modulo(0, 1, 13) = 12
  - modulo(1, 1, 13) ... modulo(12, 1, 13) = 1...12
  - modulo(13, 1, 13) = 1
  - modulo(13.123, 1, 13) = 1.123
- maximumDayInMonthFor(yearValue, monthValue) =
  - M := modulo(monthValue, 1, 13)
  - Y := yearValue + fQuotient(monthValue, 1, 13)
  - Return a value based on M and Y:

31	M = January, March, May, July, August, October, or December
30	M = April, June, September, or November
29	M = February AND (modulo(Y, 400) = 0 OR (modulo(Y, 100) != 0) AND modulo(Y, 4) = 0)
28	Otherwise

E.1 Algorithm

Essentially, this calculation is equivalent to separating D into <year,month> and <day,hour,minute,second>

fields. The <year,month> is added to S. If the day is out of range, it is *pinned* to be within range. Thus April 31 turns into April 30. Then the <day,hour,minute,second> is added. This latter addition can cause the year and month to change.

Leap seconds are handled by the computation by treating them as overflows. Essentially, a value of 60 seconds in S is treated as if it were a duration of 60 seconds added to S (with a zero seconds field). All calculations thereafter use 60 seconds per minute.

Thus the addition of either PT1M or PT60S to any dateTime will always produce the same result. This is a special definition of addition which is designed to match common practice, and -- most importantly -- be stable over time.

A definition that attempted to take leap-seconds into account would need to be constantly updated, and could not predict the results of future implementation's additions. The decision to introduce a leap second in UTC is the responsibility of the [International Earth Rotation Service (IERS)]. They make periodic announcements as to when leap seconds are to be added, but this is not known more than a year in advance. For more information on leap seconds, see [U.S. Naval Observatory Time Service Department].

The following is the precise specification. These steps must be followed in the same order. If a field in D is not specified, it is treated as if it were zero. If a field in S is not specified, it is treated in the calculation as if it were the minimum allowed value in that field, however, after the calculation is concluded, the corresponding field in E is removed (set to unspecified).

- *Months (may be modified additionally below)*
  - temp := S[month] + D[month]
  - E[month] := modulo(temp, 1, 13)
  - carry := fQuotient(temp, 1, 13)
- *Years (may be modified additionally below)*
  - E[year] := S[year] + D[year] + carry
- *Zone*
  - E[zone] := S[zone]
- *Seconds*
  - temp := S[second] + D[second]
  - E[second] := modulo(temp, 60)
  - carry := fQuotient(temp, 60)
- *Minutes*
  - temp := S[minute] + D[minute] + carry
  - E[minute] := modulo(temp, 60)
  - carry := fQuotient(temp, 60)
- *Hours*
  - temp := S[hour] + D[hour] + carry
  - E[hour] := modulo(temp, 24)
  - carry := fQuotient(temp, 24)
- *Days*
  - if S[day] > maximumDayInMonthFor(E[year], E[month])
    - tempDays := maximumDayInMonthFor(E[year], E[month])
  - else if S[day] < 1
    - tempDays := 1
  - else
    - tempDays := S[day]
  - E[day] := tempDays + D[day] + carry
  - **START LOOP**
    - **IF** E[day] < 1
      - E[day] := E[day] + maximumDayInMonthFor(E[year], E[month] - 1)
      - carry := -1
    - **ELSE IF** E[day] > maximumDayInMonthFor(E[year], E[month])
      - E[day] := E[day] - maximumDayInMonthFor(E[year], E[month])
      - carry := 1
    - **ELSE EXIT LOOP**
    - temp := E[month] + carry
    - E[month] := modulo(temp, 1, 13)
    - E[year] := E[year] + fQuotient(temp, 1, 13)

- **GOTO START LOOP**

Examples:

dateTime	duration	result
2000-01-12T12:13:14Z	P1Y3M5DT7H10M3.3S	2001-04-17T19:23:17.3Z
2000-01	-P3M	1999-10
2000-01-12	PT33H	2000-01-13

E.2 Commutativity and Associativity

Time durations are added by simply adding each of their fields, respectively, without overflow.

The order of addition of durations to instants *is* significant. For example, there are cases where:

((dateTime + duration1) + duration2) != ((dateTime + duration2) + duration1)

Example:

(2000-03-30 + P1D) + P1M = 2000-03-31 + P1M = 2001-**04-30**

(2000-03-30 + P1M) + P1D = 2000-04-30 + P1D = 2000-**05-01**

F Regular Expressions

A ·regular expression· *R* is a sequence of characters that denote a **set of strings** *L(R)*. When used to constrain a ·lexical space·, a **regular expression** *R* asserts that only strings in *L(R)* are valid literals for values of that type.

[Definition:] A **regular expression** is composed from zero or more ·branch·es, separated by | characters.

Regular Expression
`[1] regExp ::= branch ( '|' branch )*`

For all ·branch·es *S*, and for all ·regular expression·s *T*, valid ·regular expression·s *R* are:	Denoting the set of strings *L(R)* containing:
(empty string)	the set containing just the empty string
S	all strings in *L(S)*
S|*T*	all strings in *L(S)* and all strings in *L(T)*

[Definition:] A **branch** consists of zero or more ·piece·s, concatenated together.

Branch
`[2] branch ::= piece*`

For all ·piece·s *S*, and for all ·branch·es *T*, valid ·branch·es *R* are:	Denoting the set of strings *L(R)* containing:
S	all strings in *L(S)*

ST	all strings *st* with *s* in *L(S)* and *t* in *L(T)*

[Definition:] A **piece** is an ·atom·, possibly followed by a ·quantifier·.

Piece

```
[3]    piece    ::=    atom quantifier?
```

For all ·atom·s *S* and non-negative integers *n*, *m* such that *n* <= *m*, valid ·piece·s *R* are:	Denoting the set of strings *L(R)* containing:
S	all strings in *L(S)*
S?	the empty string, and all strings in *L(S)*.
*S**	All strings in *L(S?)* and all strings *st* with *s* in *L(S*)* and *t* in *L(S)*. *(all concatenations of zero or more strings from L(S))*
S+	All strings *st* with *s* in *L(S)* and *t* in *L(S*)*. *(all concatenations of one or more strings from L(S))*
S{n,m}	All strings *st* with *s* in *L(S)* and *t* in *L(S{n-1,m-1})*. *(All sequences of at least n, and at most m, strings from L(S))*
S{n}	All strings in *L(S{n,n})*. *(All sequences of exactly n strings from L (S))*
S{n,}	All strings in L(S{n}S*) *(All sequences of at least n, strings from L (S))*
S{0,m}	All strings *st* with *s* in *L(S?)* and *t* in *L(S{0,m-1})*. *(All sequences of at most m, strings from L(S))*
S{0,0}	The set containing only the empty string

NOTE: The regular expression language in the Perl Programming Language [Perl] does not include a quantifier of the form `S{,m)`, since it is logically equivalent to `S{0,m}`. We have, therefore, left this logical possibility out of the regular expression language defined by this specification. We welcome further input from implementors and schema authors on this issue.

[Definition:] A **quantifier** is one of ?, *, +, `{n,m}` or `{n,}`, which have the meanings defined in the table above.

Quanitifer

```
[4]    quantifier    ::=    [?*+] | ( '{' quantity '}' )
[5]    quantity      ::=    quantRange | quantMin | QuantExact
[6]    quantRange    ::=    QuantExact ',' QuantExact
[7]    quantMin      ::=    QuantExact ','
[8]    QuantExact    ::=    [0-9]+
```

[Definition:] An **atom** is either a ·normal character·, a ·character class·, or a parenthesized ·regular expression·.

Atom

```
[9]    atom    ::=    Char | charClass | ( '(' regExp ')' )
```

For all ·normal character·s *c*, ·character class·es *C*, and ·regular expression·s *S*, valid ·atom·s *R* are:	Denoting the set of strings *L(R)* containing:

c	the single string consisting only of *c*
C	all strings in *L(C)*
(*S*)	all strings in *L(S)*

[Definition:] A **metacharacter** is either `.`, `\`, `?`, `*`, `+`, `{`, `}` `(`, `)`, `[` or `]`. These characters have special meanings in ·regular expression·s, but can be escaped to form ·atom·s that denote the sets of strings containing only themselves, i.e., an escaped ·metacharacter· behaves like a ·normal character·.

[Definition:] A **normal character** is any XML character that is not a metacharacter. In ·regular expression·s, a normal character is an atom that denotes the singleton set of strings containing only itself.

Normal Character

```
[10]    Char    ::=    [^.\?*+()|#x5B#x5D]
```

Note that a ·normal character· can be represented either as itself, or with a character reference.

F.1 Character Classes

[Definition:] A **character class** is an ·atom· *R* that identifies a **set of characters** *C(R)*. The set of strings *L(R)* denoted by a character class *R* contains one single-character string "*c*" for each character *c* in *C(R)*.

Character Class

```
[11]    charClass    ::=    charClassEsc |
                            charClassExpr
```

A character class is either a ·character class escape· or a ·character class expression·.

[Definition:] A **character class expression** is a ·character group· surrounded by `[` and `]` characters. For all character groups *G*, [*G*] is a valid **character class expression**, identifying the set of characters *C*([*G*]) = *C*(*G*).

Character Class Expression

```
[12]    charClassExpr    ::=    '[' charGroup ']'
```

[Definition:] A **character group** is either a ·positive character group·, a ·negative character group·, or a ·character class subtraction·.

Character Group

```
[13]    charGroup    ::=    posCharGroup | negCharGroup |
                            charClassSub
```

[Definition:] A **positive character group** consists of one or more ·character range·s or ·character class escape·s, concatenated together. A **positive character group** identifies the set of characters containing all of the characters in all of the sets identified by its constituent ranges or escapes.

Positive Character Group

```
[14]    posCharGroup    ::=    ( charRange | charClassEsc )
                               +
```

For all ·character range·s *R*, all ·character class escape·s *E*, and all ·positive character group·s *P*, valid ·positive character group·s *G* are:	Identifying the set of characters *C(G)* containing:
R	all characters in *C(R)*.
E	all characters in *C(E)*.
RP	all characters in *C(R)* and all characters in *C(P)*.
EP	all characters in *C(E)* and all characters in *C(P)*.

[Definition:] A **negative character group** is a ·positive character group· preceded by the ^ character. For all ·positive character group·s *P*, *^P* is a valid **negative character group**, and *C(^P)* contains all XML characters that are *not* in *C(P)*.

Negative Character Group

```
[15]    negCharGroup    ::=    '^' posCharGroup
```

[Definition:] A **character class subtraction** is a ·character class expression· subtracted from a ·positive character group· or ·negative character group·, using the - character.

Character Class Subtraction

```
[16]    charClassSub    ::=    ( posCharGroup | negCharGroup ) '-'
                               charClassExpr
```

For any ·positive character group· or ·negative character group· *G*, and any ·character class expression· *C*, *G-C* is a valid ·character class subtraction·, identifying the set of all characters in *C(G)* that are not also in *C(C)*.

[Definition:] A **character range** *R* identifies a set of characters *C(R)* containing all XML characters with UCS code points in a specified range.

Character Range

```
[17]    charRange         ::=    seRange | XmlCharRef | XmlCharIncDash
[18]    seRange           ::=    charOrEsc '-' charOrEsc
[19]    XmlCharRef        ::=    ( '&#' [0-9]+ ';' ) | (' &#x' [0-9a-fA-F]+ ';' )
[20]    charOrEsc         ::=    XmlChar | SingleCharEsc
[21]    XmlChar           ::=    [^\#x2D#x5B#x5D]
[22]    XmlCharIncDash    ::=    [^\#x5B#x5D]
```

A single XML character is a ·character range· that identifies the set of characters containing only itself. All XML characters are valid character ranges, except as follows:

- The [,], and \ characters are not valid character ranges;
- The ^ character is only valid at the beginning of a ·positive character group· if it is part of a ·negative character group·; and
- The - character is a valid character range only at the beginning or end of a ·positive character group·.

A ·character range· ·may· also be written in the form *s-e*, identifying the set that contains all XML characters with UCS code points greater than or equal to the code point of *s*, but not greater than the code point of *e*.

s-e is a valid character range iff:

- *s* is a ·single character escape·, or an XML character;
- *s* is not \
- If s is the first character in a ·character class expression·, then *s* is not ^
- *e* is a ·single character escape·, or an XML character;
- *e* is not \ or [; and
- The code point of *e* is greater than or equal to the code point of *s*;

NOTE: The code point of a ·single character escape· is the code point of the single character in the set of characters that it identifies.

F.1.1 Character Class Escapes

[Definition:] A **character class escape** is a short sequence of characters that identifies predefined character class. The valid character class escapes are the ·single character escape·s, the ·multi-character escape·s, and the ·category escape·s (including the ·block escape·s).

Character Class Escape

```
[23]    charClassEsc    ::=    ( SingleCharEsc | MultiCharEsc | catEsc |
                               complEsc )
```

[Definition:] A **single character escape** identifies a set containing a only one character -- usually because that character is difficult or impossible to write directly into a ·regular expression·.

Single Character Escape

```
[24]    SingleCharEsc    ::=    '\' [nrt\|.?*+(){}#x2D#x5B#x5D#x5E]
```

The valid ·single character escape·s are:	Identifying the set of characters *C(R)* containing:
\n	the newline character (#xA)
\r	the return character (#xD)
\t	the tab character (#x9)
\\	\
|	|
\.	.
\-	-
\^	^
\?	?
*	*
\+	+
\{	{
\}	}
\(	(
\)	)
\[	[
\]	]

[Definition:] [Unicode Database] specifies a number of possible values for the "General Category" property and provides mappings from code points to specific character properties. The set containing all characters that have property X, can be identified with a **category escape** \p{X}. The complement of this set is specified with the

category escape `\P{X}`. (`[\P{X}]` = `[^\p{X}]`).

Category Escape

```
[25]    catEsc      ::=    '\p{' charProp '}'
[26]    complEsc    ::=    '\P{' charProp '}'
[27]    charProp    ::=    IsCategory | IsBlock
```

NOTE: [Unicode Database] is subject to future revision. For example, the mapping from code points to character properties might be updated. All ·minimally conforming· processors ·must· support the character properties defined in the version of [Unicode Database] that is current at the time this specification became a W3C Recommendation. However, implementors are encouraged to support the character properties defined in any future version.

The following table specifies the recognized values of the "General Category" property.

Category	Property	Meaning
Letters	L	All Letters
	Lu	uppercase
	Ll	lowercase
	Lt	titlecase
	Lm	modifier
	Lo	other
Marks	M	All Marks
	Mn	nonspacing
	Mc	spacing combining
	Me	enclosing
Numbers	N	All Numbers
	Nd	decimal digit
	Nl	letter
	No	other
Punctuation	P	All Punctuation
	Pc	connector
	Pd	dash
	Ps	open
	Pe	close
	Pi	initial quote (may behave like Ps or Pe depending on usage)
	Pf	final quote (may behave like Ps or Pe depending on usage)
	Po	other
Separators	Z	All Separators
	Zs	space
	Zl	line
	Zp	paragraph

Symbols	S	All Symbols
	Sm	math
	Sc	currency
	Sk	modifier
	So	other
Other	C	All Others
	Cc	control
	Cf	format
	Co	private use
	Cn	not assigned

Categories

```
[28]    IsCategory      ::=    Letters | Marks | Numbers | Punctuation |
                               Separators | Symbols | Others
[29]    Letters         ::=    'L' [ultmo]?
[30]    Marks           ::=    'M' [nce]?
[31]    Numbers         ::=    'N' [dlo]?
[32]    Punctuation     ::=    'P' [cdseifo]?
[33]    Separators      ::=    'Z' [slp]?
[34]    Symbols         ::=    'S' [mcko]?
[35]    Others          ::=    'C' [cfon]?
```

NOTE: The properties mentioned above exclude the `Cs` property. The `Cs` property identifies "surrogate" characters, which do not occur at the level of the "character abstraction" that XML instance documents operate on.

[Definition:] [Unicode Database] groups code points into a number of blocks such as Basic Latin (i.e., ASCII), Latin-1 Supplement, Hangul Jamo, CJK Compatibility, etc. The set containing all characters that have block name X (with all white space stripped out), can be identified with a **block escape** `\p{IsX}`. The complement of this set is specified with the **block escape** `\P{IsX}`. (`[\P{IsX}] = [^\p{IsX}]`).

Block Escape

```
[36]    IsBlock     ::=    'Is' [a-zA-Z0-9#x2D]+
```

The following table specifies the recognized block names (for more information, see the "Blocks.txt" file in [Unicode Database]).

Start Code	End Code	Block Name		Start Code	End Code	Block Name
#x0000	#x007F	BasicLatin		#x0080	#x00FF	Latin-1Supplement
#x0100	#x017F	LatinExtended-A		#x0180	#x024F	LatinExtended-B
#x0250	#x02AF	IPAExtensions		#x02B0	#x02FF	SpacingModifierLetters
#x0300	#x036F	CombiningDiacriticalMarks		#x0370	#x03FF	Greek
#x0400	#x04FF	Cyrillic		#x0530	#x058F	Armenian
#x0590	#x05FF	Hebrew		#x0600	#x06FF	Arabic
#x0700	#x074F	Syriac		#x0780	#x07BF	Thaana

#x0900	#x097F	Devanagari		#x0980	#x09FF	Bengali		
#x0A00	#x0A7F	Gurmukhi		#x0A80	#x0AFF	Gujarati		
#x0B00	#x0B7F	Oriya		#x0B80	#x0BFF	Tamil		
#x0C00	#x0C7F	Telugu		#x0C80	#x0CFF	Kannada		
#x0D00	#x0D7F	Malayalam		#x0D80	#x0DFF	Sinhala		
#x0E00	#x0E7F	Thai		#x0E80	#x0EFF	Lao		
#x0F00	#x0FFF	Tibetan		#x1000	#x109F	Myanmar		
#x10A0	#x10FF	Georgian		#x1100	#x11FF	HangulJamo		
#x1200	#x137F	Ethiopic		#x13A0	#x13FF	Cherokee		
#x1400	#x167F	UnifiedCanadianAboriginalSyllabics		#x1680	#x169F	Ogham		
#x16A0	#x16FF	Runic		#x1780	#x17FF	Khmer		
#x1800	#x18AF	Mongolian		#x1E00	#x1EFF	LatinExtendedAdditional		
#x1F00	#x1FFF	GreekExtended		#x2000	#x206F	GeneralPunctuation		
#x2070	#x209F	SuperscriptsandSubscripts		#x20A0	#x20CF	CurrencySymbols		
#x20D0	#x20FF	CombiningMarksforSymbols		#x2100	#x214F	LetterlikeSymbols		
#x2150	#x218F	NumberForms		#x2190	#x21FF	Arrows		
#x2200	#x22FF	MathematicalOperators		#x2300	#x23FF	MiscellaneousTechnical		
#x2400	#x243F	ControlPictures		#x2440	#x245F	OpticalCharacterRecognition		
#x2460	#x24FF	EnclosedAlphanumerics		#x2500	#x257F	BoxDrawing		
#x2580	#x259F	BlockElements		#x25A0	#x25FF	GeometricShapes		
#x2600	#x26FF	MiscellaneousSymbols		#x2700	#x27BF	Dingbats		
#x2800	#x28FF	BraillePatterns		#x2E80	#x2EFF	CJKRadicalsSupplement		
#x2F00	#x2FDF	KangxiRadicals		#x2FF0	#x2FFF	IdeographicDescriptionCharacters		
#x3000	#x303F	CJKSymbolsandPunctuation		#x3040	#x309F	Hiragana		
#x30A0	#x30FF	Katakana		#x3100	#x312F	Bopomofo		
#x3130	#x318F	HangulCompatibilityJamo		#x3190	#x319F	Kanbun		
#x31A0	#x31BF	BopomofoExtended		#x3200	#x32FF	EnclosedCJKLettersandMonths		
#x3300	#x33FF	CJKCompatibility		#x3400	#x4DB5	CJKUnifiedIdeographsExtensionA		
#x4E00	#x9FFF	CJKUnifiedIdeographs		#xA000	#xA48F	YiSyllables		
#xA490	#xA4CF	YiRadicals		#xAC00	#xD7A3	HangulSyllables		
#xD800	#xDB7F	HighSurrogates		#xDB80	#xDBFF	HighPrivateUseSurrogates		
#xDC00	#xDFFF	LowSurrogates		#xE000	#xF8FF	PrivateUse		
#xF900	#xFAFF	CJKCompatibilityIdeographs		#xFB00	#xFB4F	AlphabeticPresentationForms		
#xFB50	#xFDFF	ArabicPresentationForms-A		#xFE20	#xFE2F	CombiningHalfMarks		
#xFE30	#xFE4F	CJKCompatibilityForms		#xFE50	#xFE6F	SmallFormVariants		
#xFE70	#xFEFE	ArabicPresentationForms-B		#xFEFF	#xFEFF	Specials		
#xFF00	#xFFEF	HalfwidthandFullwidthForms		#xFFF0	#xFFFD	Specials		

#x10300	#x1032F	OldItalic		#x10330	#x1034F	Gothic	
#x10400	#x1044F	Deseret		#x1D000	#x1D0FF	ByzantineMusicalSymbols	
#x1D100	#x1D1FF	MusicalSymbols		#x1D400	#x1D7FF	MathematicalAlphanumericSymbols	
#x20000	#x2A6D6	CJKUnifiedIdeographsExtensionB		#x2F800	#x2FA1F	CJKCompatibilityIdeographsSupplement	
#xE0000	#xE007F	Tags		#xF0000	#xFFFFD	PrivateUse	
#x100000	#x10FFFD	PrivateUse					

NOTE: [Unicode Database] is subject to future revision. For example, the grouping of code points into blocks might be updated. All ·minimally conforming· processors ·must· support the blocks defined in the version of [Unicode Database] that is current at the time this specification became a W3C Recommendation. However, implementors are encouraged to support the blocks defined in any future version of the Unicode Standard.

For example, the ·block escape· for identifying the ASCII characters is `\p{IsBasicLatin}`.

[Definition:] A **multi-character escape** provides a simple way to identify a commonly used set of characters:

Multi-Character Escape
`[37] MultiCharEsc ::= '.' | ('\' [sSiIcCdDwW])`

Character sequence	Equivalent ·character class·
.	[^\n\r]
\s	[#x20\t\n\r]
\S	[^\s]
\i	the set of initial name characters, those ·match·ed by Letter | '_' | ':'
\I	[^\i]
\c	the set of name characters, those ·match·ed by NameChar
\C	[^\c]
\d	\p{Nd}
\D	[^\d]
\w	[#x0000-#x10FFFF]-[\p{P}\p{Z}\p{C}] (*all characters except the set of "punctuation", "separator" and "other" characters*)
\W	[^\w]

NOTE: The ·regular expression· language defined here does not attempt to provide a general solution to "regular expressions" over UCS character sequences. In particular, it does not easily provide for matching sequences of base characters and combining marks. The language is targeted at support of "Level 1" features as defined in [Unicode Regular Expression Guidelines]. It is hoped that future versions of this specification will provide support for "Level 2" features.

G Glossary (non-normative)

The listing below is for the benefit of readers of a printed version of this document: it collects together all the definitions which appear in the document above.

atomic

Atomic datatypes are those having values which are regarded by this specification as being indivisible.

base type

Every datatype that is ·derived· by **restriction** is defined in terms of an existing datatype, referred to as its **base type**. **base type**s can be either ·primitive· or ·derived·.

bounded

A datatype is **bounded** if its ·value space· has either an ·inclusive upper bound· or an ·exclusive upper bound· and either an ·inclusive lower bound· and an ·exclusive lower bound·.

built-in

Built-in datatypes are those which are defined in this specification, and can be either ·primitive· or ·derived·;

canonical lexical representation

A **canonical lexical representation** is a set of literals from among the valid set of literals for a datatype such that there is a one-to-one mapping between literals in the **canonical lexical representation** and values in the ·value space·.

cardinality

Every ·value space· has associated with it the concept of **cardinality**. Some ·value space·s are finite, some are countably infinite while still others could conceivably be uncountably infinite (although no ·value space· defined by this specification is uncountable infinite). A datatype is said to have the cardinality of its ·value space·.

conformance to the XML Representation of Schemas

Processors which accept schemas in the form of XML documents as described in XML Representation of Simple Type Definition Schema Components (§4.1.2) (and other relevant portions of Datatype components (§4)) are additionally said to provide **conformance to the XML Representation of Schemas**, and ·must·, when processing schema documents, completely and correctly implement all ·Schema Representation Constraint·s in this specification, and ·must· adhere exactly to the specifications in XML Representation of Simple Type Definition Schema Components (§4.1.2) (and other relevant portions of Datatype components (§4)) for mapping the contents of such documents to schema components for use in validation.

constraining facet

A **constraining facet** is an optional property that can be applied to a datatype to constrain its ·value space·.

Constraint on Schemas

Constraint on Schemas

datatype

In this specification, a **datatype** is a 3-tuple, consisting of a) a set of distinct values, called its ·value space·, b) a set of lexical representations, called its ·lexical space·, and c) a set of ·facet·s that characterize properties of the ·value space·, individual values or lexical items.

derived

Derived datatypes are those that are defined in terms of other datatypes.

error

error

exclusive lower bound

A value *l* in an ·ordered· ·value space· *L* is said to be an **exclusive lower bound** of a ·value space· *V* (where *V* is a subset of *L*) if for all *v* in *V*, $l < v$.

exclusive upper bound

A value *u* in an ·ordered· ·value space· *U* is said to be an **exclusive upper bound** of a ·value space· *V* (where *V* is a subset of *U*) if for all *v* in *V*, $u > v$.

facet

A **facet** is a single defining aspect of a ·value space·. Generally speaking, each facet characterizes a ·value space· along independent axes or dimensions.

for compatibility

for compatibility

fundamental facet

A **fundamental facet** is an abstract property which serves to semantically characterize the values in a ·value space·.

inclusive lower bound

A value *l* in an ·ordered· ·value space· *L* is said to be an **inclusive lower bound** of a ·value space· *V* (where *V* is a subset of *L*) if for all *v* in *V*, *l* <= *v*.

inclusive upper bound

A value *u* in an ·ordered· ·value space· *U* is said to be an **inclusive upper bound** of a ·value space· *V* (where *V* is a subset of *U*) if for all *v* in *V*, *u* >= *v*.

itemType

The ·atomic· datatype that participates in the definition of a ·list· datatype is known as the **itemType** of that ·list· datatype.

lexical space

A **lexical space** is the set of valid *literals* for a datatype.

list

List datatypes are those having values each of which consists of a finite-length (possibly empty) sequence of values of an ·atomic· datatype.

match

match

may

may

memberTypes

The datatypes that participate in the definition of a ·union· datatype are known as the **memberTypes** of that ·union· datatype.

minimally conforming

Minimally conforming processors ·must· completely and correctly implement the ·Constraint on Schemas· and ·Validation Rule· .

must

must

non-numeric

A datatype whose values are not ·numeric· is said to be **non-numeric**.

numeric

A datatype is said to be **numeric** if its values are conceptually quantities (in some mathematical number system).

ordered

A ·value space·, and hence a datatype, is said to be **ordered** if there exists an ·order-relation· defined for that ·value space·.

order-relation

An **order relation** on a ·value space· is a mathematical relation that imposes a ·total order· or a ·partial order· on the members of the ·value space·.

partial order

A **partial order** is an ·order-relation· that is **irreflexive**, **asymmetric** and **transitive**.

primitive

Primitive datatypes are those that are not defined in terms of other datatypes; they exist *ab initio*.

regular expression

A **regular expression** is composed from zero or more ·branch·es, separated by | characters.

restriction

A datatype is said to be ·derived· by **restriction** from another datatype when values for zero or more ·constraining facet·s are specified that serve to constrain its ·value space· and/or its ·lexical space· to a subset of those of its ·base type·.

Schema Representation Constraint

Schema Representation Constraint

total order

A **total order** is an ·partial order· such that for no *a* and *b* is it the case that *a* <> *b*.

union

Union datatypes are those whose ·value space·s and ·lexical space·s are the union of the ·value space·s and ·lexical space·s of one or more other datatypes.

user-derived

User-derived datatypes are those ·derived· datatypes that are defined by individual schema designers.

Validation Rule

Validation Rule

value space

A **value space** is the set of values for a given datatype. Each value in the **value space** of a datatype is denoted by one or more literals in its ·lexical space·.

H References

H.1 Normative

Clinger, WD (1990)
William D Clinger. *How to Read Floating Point Numbers Accurately.* In *Proceedings of Conference on Programming Language Design and Implementation*, pages 92-101. Available at: ftp://ftp.ccs.neu.edu/pub/people/will/howtoread.ps

IEEE 754-1985
IEEE. *IEEE Standard for Binary Floating-Point Arithmetic.* See http://standards.ieee.org/reading/ieee/std_public/description/busarch/754-1985_desc.html

Namespaces in XML
World Wide Web Consortium. *Namespaces in XML*. Available at: http://www.w3.org/TR/1999/REC-xml-names-19990114/

RFC 1766
H. Alvestrand, ed. *RFC 1766: Tags for the Identification of Languages* 1995. Available at: http://www.ietf.org/rfc/rfc1766.txt

RFC 2045
N. Freed and N. Borenstein. *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. 1996. Available at: http://www.ietf.org/rfc/rfc2045.txt

RFC 2396
Tim Berners-Lee, et. al. *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax.*. 1998. Available at: http://www.ietf.org/rfc/rfc2396.txt

RFC 2732
RFC 2732: Format for Literal IPv6 Addresses in URL's. 1999. Available at: http://www.ietf.org/rfc/rfc2732.txt

Unicode Database
The Unicode Consortium. *The Unicode Character Database*. Available at: http://www.unicode.org/Public/3.1-Update/UnicodeCharacterDatabase-3.1.0.html

XML 1.0 (Second Edition)
World Wide Web Consortium. *Extensible Markup Language (XML) 1.0, Second Edition.* Available at: http://www.w3.org/TR/2000/WD-xml-2e-20000814

XML Linking Language
World Wide Web Consortium. XML Linking Language (XLink). Available at: http://www.w3.org/TR/2000/PR-xlink-20001220/

XML Schema Part 1: Structures
XML Schema Part 1: Structures. Available at: http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

XML Schema Requirements
World Wide Web Consortium. XML Schema Requirements. Available at: http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215

H.2 Non-normative

Character Model
Martin J. Dürst and François Yergeau, eds. Character Model for the World Wide Web. World Wide Web Consortium Working Draft. 2001. Available at: http://www.w3.org/TR/2001/WD-charmod-20010126/

Gay, DM (1990)
David M. Gay. *Correctly Rounded Binary-Decimal and Decimal-Binary Conversions.* AT&T Bell Laboratories Numerical Analysis Manuscript 90-10, November 1990. Available at: http://cm.bell-labs.com/cm/cs/doc/90/4-10.ps.gz

HTML 4.01
World Wide Web Consortium. Hypertext Markup Language, version 4.01. Available at: http://www.w3.org/TR/1999/REC-html401-19991224/

IETF INTERNET-DRAFT: IRIs
L. Masinter and M. Durst. *Internationalized Resource Identifiers* 2001. Available at: http://www.ietf.org/

internet-drafts/draft-masinter-url-i18n-07.txt

International Earth Rotation Service (IERS)
International Earth Rotation Service (IERS). See http://maia.usno.navy.mil

ISO 11404
ISO (International Organization for Standardization). *Language-independent Datatypes.* See http://www.iso.ch/cate/d19346.html

ISO 8601
ISO (International Organization for Standardization). *Representations of dates and times, 1988-06-15.* Available at: http://www.iso.ch/markete/8601.pdf

ISO 8601 Draft Revision
ISO (International Organization for Standardization). *Representations of dates and times, draft revision, 2000.*

Perl
The Perl Programming Language. See http://www.perl.com/pub/language/info/software.html

RDF Schema
World Wide Web Consortium. *RDF Schema Specification.* Available at: http://www.w3.org/TR/2000/CR-rdf-schema-20000327/

Ruby
World Wide Web Consortium. Ruby Annotation. Available at: http://www.w3.org/TR/2001/WD-ruby-20010216/

SQL
ISO (International Organization for Standardization). *ISO/IEC 9075-2:1999, Information technology --- Database languages --- SQL --- Part 2: Foundation (SQL/Foundation)*. [Geneva]: International Organization for Standardization, 1999. See http://www.iso.ch/cate/d26197.html

U.S. Naval Observatory Time Service Department
Information about Leap Seconds Available at: http://tycho.usno.navy.mil/leapsec.990505.html

Unicode Regular Expression Guidelines
Mark Davis. *Unicode Regular Expression Guidelines*, 1988. Available at: http://www.unicode.org/unicode/reports/tr18/

XML Schema Language: Part 2 Primer
World Wide Web Consortium. XML Schema Language: Part 2 Primer. Available at: http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

XSL
World Wide Web Consortium. *Extensible Stylesheet Language (XSL).* Available at: http://www.w3.org/TR/2000/CR-xsl-20001121/

I Acknowledgements (non-normative)

The following have contributed material to this draft:

- Asir S. Vedamuthu, webMethods, Inc
- Mark Davis, IBM

Co-editor Ashok Malhotra's work on this specification from March 1999 until February 2001 was supported by IBM.

The editors acknowledge the members of the XML Schema Working Group, the members of other W3C Working Groups, and industry experts in other forums who have contributed directly or indirectly to the process or content of creating this document. The Working Group is particularly grateful to Lotus Development Corp. and IBM for providing teleconferencing facilities.

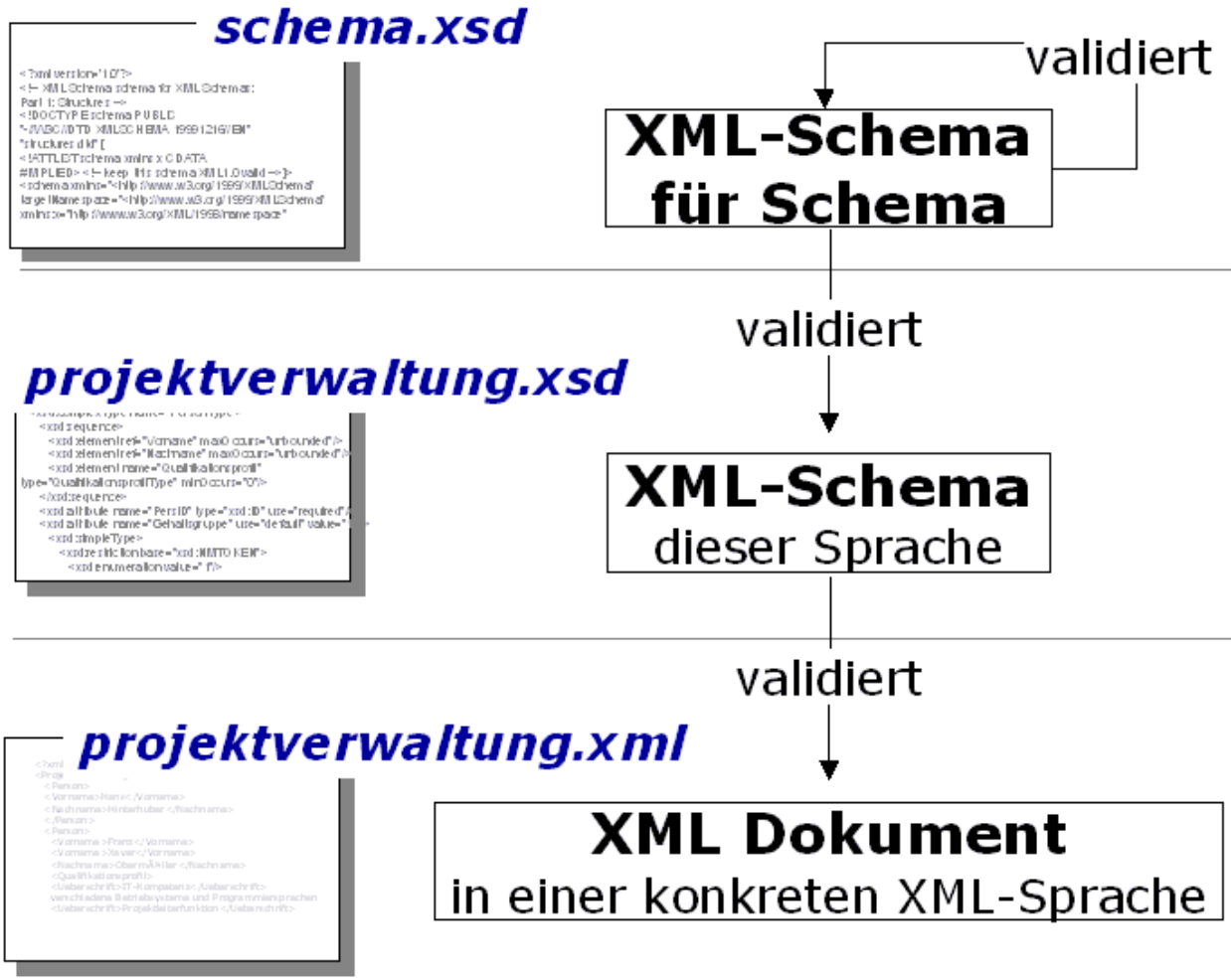
The current members of the XML Schema Working Group are:

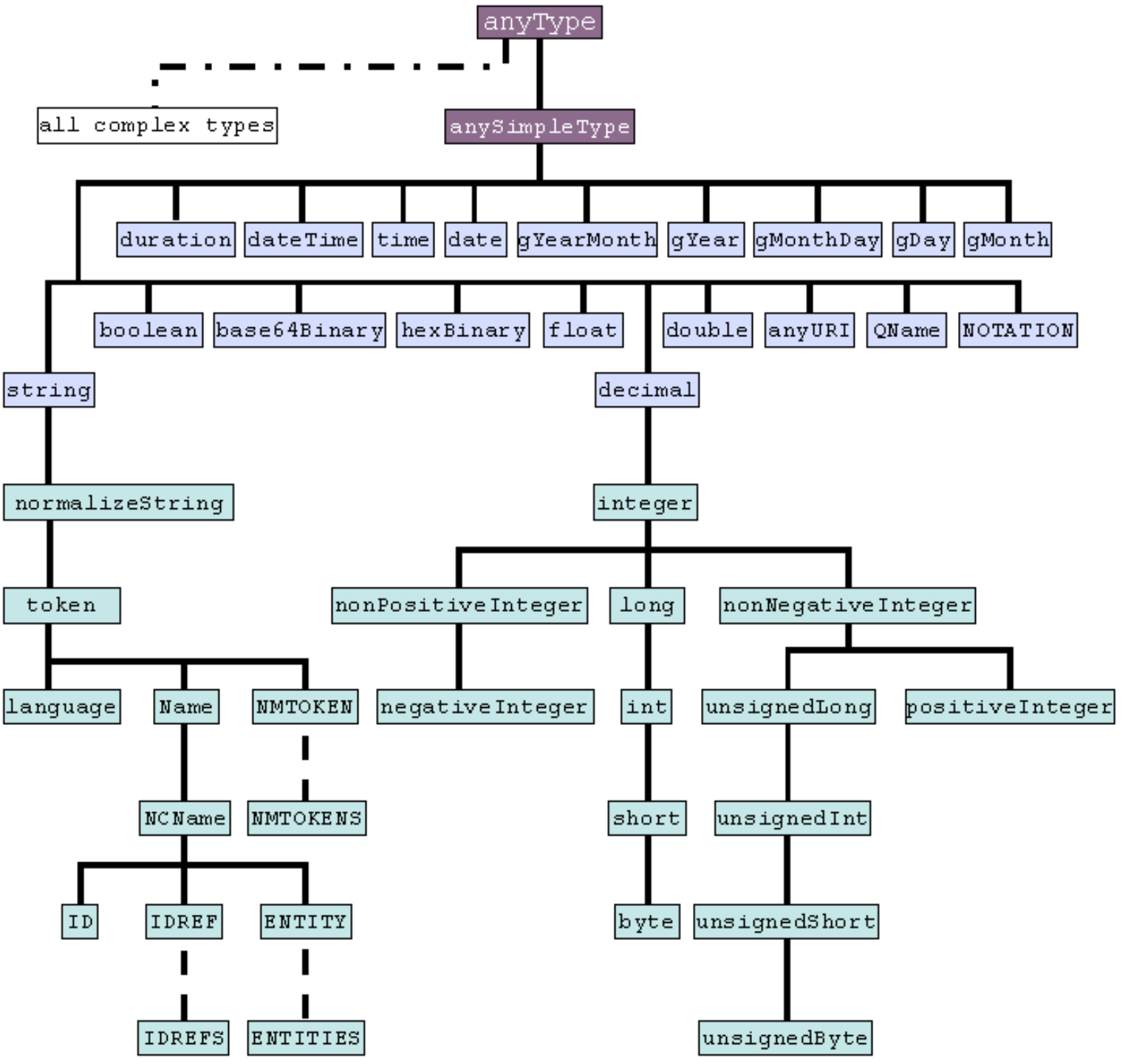
Jim Barnette, Defense Information Systems Agency (DISA); Paul V. Biron, Health Level Seven; Don Box, DevelopMentor; Allen Brown, Microsoft; Lee Buck, TIBCO Extensibility; Charles E. Campbell, Informix; Wayne Carr, Intel; Peter Chen, Bootstrap Alliance and LSU; David Cleary, Progress Software; Dan Connolly, W3C (staff contact); Ugo Corda, Xerox; Roger L. Costello, MITRE; Haavard Danielson, Progress Software; Josef Dietl, Mozquito Technologies; David Ezell, Hewlett-Packard Company; Alexander Falk, Altova GmbH; David Fallside, IBM; Dan Fox, Defense Logistics Information Service (DLIS); Matthew Fuchs, Commerce One; Andrew Goodchild, Distributed Systems Technology Centre (DSTC Pty Ltd); Paul Grosso, Arbortext, Inc; Martin Gudgin,

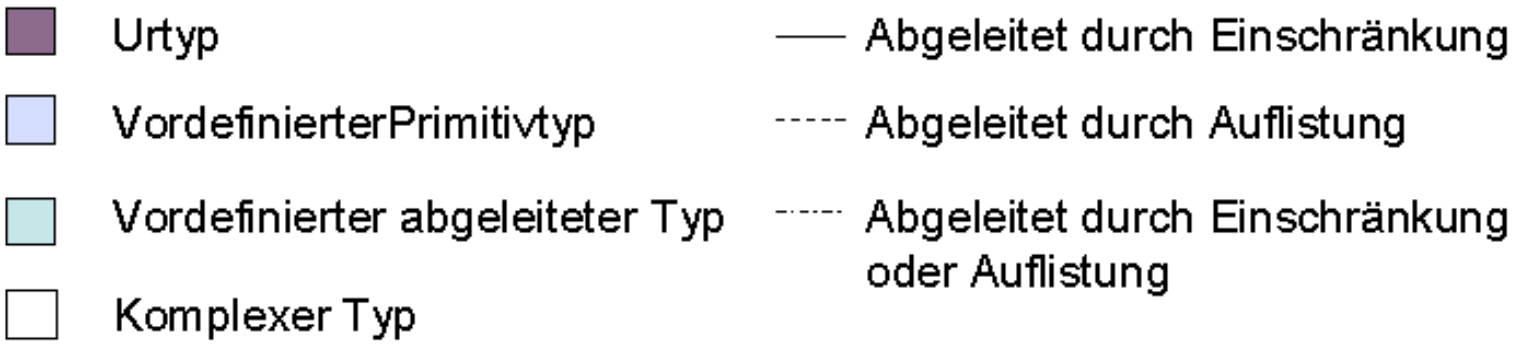
DevelopMentor; Dave Hollander, Contivo, Inc (co-chair); Mary Holstege, Invited Expert; Jane Hunter, Distributed Systems Technology Centre (DSTC Pty Ltd); Rick Jelliffe, Academia Sinica; Simon Johnston, Rational Software; Bob Lojek, Mozquito Technologies; Ashok Malhotra, Microsoft; Lisa Martin, IBM; Noah Mendelsohn, Lotus Development Corporation; Adrian Michel, Commerce One; Alex Milowski, Invited Expert; Don Mullen, TIBCO Extensibility; Dave Peterson, Graphic Communications Association; Jonathan Robie, Software AG; Eric Sedlar, Oracle Corp.; C. M. Sperberg-McQueen, W3C (co-chair); Bob Streich, Calico Commerce; William K. Stumbo, Xerox; Henry S. Thompson, University of Edinburgh; Mark Tucker, Health Level Seven; Asir S. Vedamuthu, webMethods, Inc; Priscilla Walmsley, XMLSolutions; Norm Walsh, Sun Microsystems; Aki Yoshida, SAP AG; Kongyi Zhou, Oracle Corp.

The XML Schema Working Group has benefited in its work from the participation and contributions of a number of people not currently members of the Working Group, including in particular those named below. Affiliations given are those current at the time of their work with the WG.

Paula Angerstein, Vignette Corporation; David Beech, Oracle Corp.; Gabe Beged-Dov, Rogue Wave Software; Greg Bumgardner, Rogue Wave Software; Dean Burson, Lotus Development Corporation; Mike Cokus, MITRE; Andrew Eisenberg, Progress Software; Rob Ellman, Calico Commerce; George Feinberg, Object Design; Charles Frankston, Microsoft; Ernesto Guerrieri, Inso; Michael Hyman, Microsoft; Renato Iannella, Distributed Systems Technology Centre (DSTC Pty Ltd); Dianne Kennedy, Graphic Communications Association; Janet Koenig, Sun Microsystems; Setrag Khoshafian, Technology Deployment International (TDI); Ara Kullukian, Technology Deployment International (TDI); Andrew Layman, Microsoft; Dmitry Lenkov, Hewlett-Packard Company; John McCarthy, Lawrence Berkeley National Laboratory; Murata Makoto, Xerox; Eve Maler, Sun Microsystems; Murray Maloney, Muzmo Communication, acting for Commerce One; Chris Olds, Wall Data; Frank Olken, Lawrence Berkeley National Laboratory; Shriram Revankar, Xerox; Mark Reinhold, Sun Microsystems; John C. Schneider, MITRE; Lew Shannon, NCR; William Shea, Merrill Lynch; Ralph Swick, W3C; Tony Stewart, Rivcom; Matt Timmermans, Microstar; Jim Trezzo, Oracle Corp.; Steph Tryphonas, Microstar


schema.xsd
validiert
XML-Schema
für Schema
validiert
projektverwaltung.xsd
XML-Schema
dieser Sprache
validiert
projektverwaltung.xml
XML Dokument
in einer konkreten XML-Sprache


anyType
all complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizeString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte


Urtyp
VordefinierterPrimitivtyp
Vordefinierter abgeleiteter Typ
Komplexer Typ
Abgeleitet durch Einschränkung
Abgeleitet durch Auflistung
Abgeleitet durch Einschränkung oder Auflistung

IEEE Std 754-1985 IEEE Standard for Binary Floating-Point Arithmetic - Description

Content

- 7.3 Overflow
- 7.4 Underflow
- 7.5 Inexact

- **8. Traps**
  - 8.1 Trap Handler
  - 8.2 Precedence
- **Annex A Recommended Functions and Predicates**

---

links: [Ordering Information] - [Catalog] - [Standard Status]

(m.v.rodriguez@ieee.org)
URL:
(Modified:Tue Nov 4 03:48:17 2003)

Network Working Group
Request for Comments: 2045
Obsoletes: 1521, 1522, 1590
Category: Standards Track

N. Freed
Innosoft
N. Borenstein
First Virtual

November 1996

Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

STD 11, RFC 822, defines a message representation protocol specifying considerable detail about US-ASCII message headers, and leaves the message content, or message body, as flat US-ASCII text. This set of documents, collectively called the Multipurpose Internet Mail Extensions, or MIME, redefines the format of messages to allow for

(1) textual message bodies in character sets other than US-ASCII,

(2) an extensible set of different formats for non-textual message bodies,

(3) multi-part message bodies, and

(4) textual header information in character sets other than US-ASCII.

These documents are based on earlier work documented in RFC 934, STD 11, and RFC 1049, but extends and revises them. Because RFC 822 said

so little about message bodies, these documents are largely orthogonal to (rather than a revision of) RFC 822.

This initial document specifies the various headers used to describe the structure of MIME messages. The second document, RFC 2046, defines the general structure of the MIME media typing system and defines an initial set of media types. The third document, RFC 2047, describes extensions to RFC 822 to allow non-US-ASCII text data in

Internet mail header fields. The fourth document, RFC 2048, specifies various IANA registration procedures for MIME-related facilities. The fifth and final document, RFC 2049, describes MIME conformance criteria as well as providing some illustrative examples of MIME message formats, acknowledgements, and the bibliography.

These documents are revisions of RFCs 1521, 1522, and 1590, which themselves were revisions of RFCs 1341 and 1342. An appendix in RFC 2049 describes differences and changes from previous versions.

Table of Contents

1. Introduction

Since its publication in 1982, RFC 822 has defined the standard format of textual mail messages on the Internet. Its success has been such that the RFC 822 format has been adopted, wholly or partially, well beyond the confines of the Internet and the Internet SMTP transport defined by RFC 821. As the format has seen wider use, a number of limitations have proven increasingly restrictive for the user community.

RFC 822 was intended to specify a format for text messages. As such, non-text messages, such as multimedia messages that might include audio or images, are simply not mentioned. Even in the case of text, however, RFC 822 is inadequate for the needs of mail users whose languages require the use of character sets richer than US-ASCII. Since RFC 822 does not specify mechanisms for mail containing audio, video, Asian language text, or even text in most European languages, additional specifications are needed.

One of the notable limitations of RFC 821/822 based mail systems is

the fact that they limit the contents of electronic mail messages to relatively short lines (e.g. 1000 characters or less [RFC-821]) of 7bit US-ASCII. This forces users to convert any non-textual data that they may wish to send into seven-bit bytes representable as printable US-ASCII characters before invoking a local mail UA (User Agent, a program with which human users send and receive mail). Examples of such encodings currently used in the Internet include pure hexadecimal, uuencode, the 3-in-4 base 64 scheme specified in RFC 1421, the Andrew Toolkit Representation [ATK], and many others.

The limitations of RFC 822 mail become even more apparent as gateways are designed to allow for the exchange of mail messages between RFC 822 hosts and X.400 hosts. X.400 [X400] specifies mechanisms for the inclusion of non-textual material within electronic mail messages. The current standards for the mapping of X.400 messages to RFC 822 messages specify either that X.400 non-textual material must be converted to (not encoded in) IA5Text format, or that they must be discarded, notifying the RFC 822 user that discarding has occurred. This is clearly undesirable, as information that a user may wish to receive is lost. Even though a user agent may not have the capability of dealing with the non-textual material, the user might have some mechanism external to the UA that can extract useful information from the material. Moreover, it does not allow for the fact that the message may eventually be gatewayed back into an X.400 message handling system (i.e., the X.400 message is "tunneled" through Internet mail), where the non-textual information would definitely become useful again.

This document describes several mechanisms that combine to solve most of these problems without introducing any serious incompatibilities with the existing world of RFC 822 mail. In particular, it describes:

(1) A MIME-Version header field, which uses a version number to declare a message to be conformant with MIME and allows mail processing agents to distinguish between such messages and those generated by older or non-conformant software, which are presumed to lack

such a field.

(2) A Content-Type header field, generalized from RFC 1049, which can be used to specify the media type and subtype of data in the body of a message and to fully specify the native representation (canonical form) of such data.

(3) A Content-Transfer-Encoding header field, which can be used to specify both the encoding transformation that was applied to the body and the domain of the result. Encoding transformations other than the identity transformation are usually applied to data in order to allow it to pass through mail transport mechanisms which may have data or character set limitations.

(4) Two additional header fields that can be used to further describe the data in a body, the Content-ID and Content-Description header fields.

All of the header fields defined in this document are subject to the general syntactic rules for header fields specified in RFC 822. In particular, all of these header fields except for Content-Disposition can include RFC 822 comments, which have no semantic content and should be ignored during MIME processing.

Finally, to specify and promote interoperability, RFC 2049 provides a basic applicability statement for a subset of the above mechanisms that defines a minimal level of "conformance" with this document.

HISTORICAL NOTE: Several of the mechanisms described in this set of documents may seem somewhat strange or even baroque at first reading. It is important to note that compatibility with existing standards AND robustness across existing practice were two of the highest priorities of the working group that developed this set of documents. In particular, compatibility was always favored over elegance.

Please refer to the current edition of the "Internet Official Protocol Standards" for the standardization state and status of this protocol. RFC 822 and STD 3, RFC 1123 also provide essential background for MIME since no conforming implementation of MIME can violate them. In addition, several other informational RFC documents will be of interest to the MIME implementor, in particular RFC 1344, RFC 1345, and RFC 1524.

2. Definitions, Conventions, and Generic BNF Grammar

Although the mechanisms specified in this set of documents are all described in prose, most are also described formally in the augmented BNF notation of RFC 822. Implementors will need to be familiar with this notation in order to understand this set of documents, and are referred to RFC 822 for a complete explanation of the augmented BNF notation.

Some of the augmented BNF in this set of documents makes named references to syntax rules defined in RFC 822. A complete formal grammar, then, is obtained by combining the collected grammar appendices in each document in this set with the BNF of RFC 822 plus the modifications to RFC 822 defined in RFC 1123 (which specifically changes the syntax for `return', `date' and `mailbox').

All numeric and octet values are given in decimal notation in this set of documents. All media type values, subtype values, and parameter names as defined are case-insensitive. However, parameter values are case-sensitive unless otherwise specified for the specific parameter.

FORMATTING NOTE: Notes, such at this one, provide additional nonessential information which may be skipped by the reader without missing anything essential. The primary purpose of these non-essential notes is to convey information about the rationale of this set of documents, or to place these documents in the proper historical or evolutionary context. Such information may in particular be skipped by those who are focused entirely on building a conformant implementation, but may be of use to those who wish to understand why certain design choices were made.

2.1. CRLF

The term CRLF, in this set of documents, refers to the sequence of octets corresponding to the two US-ASCII characters CR (decimal value 13) and LF (decimal value 10) which, taken together, in this order, denote a line break in RFC 822 mail.

2.2. Character Set

The term "character set" is used in MIME to refer to a method of converting a sequence of octets into a sequence of characters. Note that unconditional and unambiguous conversion in the other direction is not required, in that not all characters may be representable by a given character set and a character set may provide more than one sequence of octets to represent a particular sequence of characters.

This definition is intended to allow various kinds of character encodings, from simple single-table mappings such as US-ASCII to complex table switching methods such as those that use ISO 2022's techniques, to be used as character sets. However, the definition associated with a MIME character set name must fully specify the mapping to be performed. In particular, use of external profiling information to determine the exact mapping is not permitted.

NOTE: The term "character set" was originally to describe such straightforward schemes as US-ASCII and ISO-8859-1 which have a simple one-to-one mapping from single octets to single characters. Multi-octet coded character sets and switching techniques make the situation more complex. For example, some communities use the term "character encoding" for what MIME calls a "character set", while using the phrase "coded character set" to denote an abstract mapping from integers (not octets) to characters.

2.3. Message

The term "message", when not further qualified, means either a (complete or "top-level") RFC 822 message being transferred on a network, or a message encapsulated in a body of type "message/rfc822" or "message/partial".

2.4. Entity

The term "entity", refers specifically to the MIME-defined header

fields and contents of either a message or one of the parts in the body of a multipart entity. The specification of such entities is the essence of MIME. Since the contents of an entity are often called the "body", it makes sense to speak about the body of an entity. Any sort of field may be present in the header of an entity, but only those fields whose names begin with "content-" actually have any MIME-related meaning. Note that this does NOT imply thay they have no meaning at all -- an entity that is also a message has non-MIME header fields whose meanings are defined by RFC 822.

2.5. Body Part

The term "body part" refers to an entity inside of a multipart entity.

2.6. Body

The term "body", when not further qualified, means the body of an entity, that is, the body of either a message or of a body part.

NOTE: The previous four definitions are clearly circular. This is unavoidable, since the overall structure of a MIME message is indeed recursive.

2.7. 7bit Data

"7bit data" refers to data that is all represented as relatively short lines with 998 octets or less between CRLF line separation sequences [RFC-821]. No octets with decimal values greater than 127 are allowed and neither are NULs (octets with decimal value 0). CR (decimal value 13) and LF (decimal value 10) octets only occur as part of CRLF line separation sequences.

2.8. 8bit Data

"8bit data" refers to data that is all represented as relatively

short lines with 998 octets or less between CRLF line separation sequences [RFC-821]), but octets with decimal values greater than 127 may be used. As with "7bit data" CR and LF octets only occur as part of CRLF line separation sequences and no NULs are allowed.

2.9. Binary Data

"Binary data" refers to data where any sequence of octets whatsoever is allowed.

2.10. Lines

"Lines" are defined as sequences of octets separated by a CRLF sequences. This is consistent with both RFC 821 and RFC 822. "Lines" only refers to a unit of data in a message, which may or may not correspond to something that is actually displayed by a user agent.

3. MIME Header Fields

MIME defines a number of new RFC 822 header fields that are used to describe the content of a MIME entity. These header fields occur in at least two contexts:

(1) As part of a regular RFC 822 message header.

(2) In a MIME body part header within a multipart construct.

The formal definition of these header fields is as follows:

```
entity-headers := [ content CRLF ]
                  [ encoding CRLF ]
                  [ id CRLF ]
```

```
                [ description CRLF ]
                *( MIME-extension-field CRLF )

  MIME-message-headers := entity-headers
                          fields
                          version CRLF
                          ; The ordering of the header
                          ; fields implied by this BNF
                          ; definition should be ignored.

  MIME-part-headers := entity-headers
                       [ fields ]
                       ; Any field not beginning with
                       ; "content-" can have no defined
                       ; meaning and may be ignored.
                       ; The ordering of the header
                       ; fields implied by this BNF
                       ; definition should be ignored.
```

The syntax of the various specific MIME header fields will be described in the following sections.

4. MIME-Version Header Field

Since RFC 822 was published in 1982, there has really been only one format standard for Internet messages, and there has been little perceived need to declare the format standard in use. This document is an independent specification that complements RFC 822. Although the extensions in this document have been defined in such a way as to be compatible with RFC 822, there are still circumstances in which it might be desirable for a mail-processing agent to know whether a message was composed with the new standard in mind.

Therefore, this document defines a new header field, "MIME-Version", which is to be used to declare the version of the Internet message body format standard in use.

Messages composed in accordance with this document MUST include such a header field, with the following verbatim text:

MIME-Version: 1.0

The presence of this header field is an assertion that the message has been composed in compliance with this document.

Since it is possible that a future document might extend the message format standard again, a formal BNF is given for the content of the MIME-Version field:

version := "MIME-Version" ":" 1*DIGIT "." 1*DIGIT

Thus, future format specifiers, which might replace or extend "1.0", are constrained to be two integer fields, separated by a period. If a message is received with a MIME-version value other than "1.0", it cannot be assumed to conform with this document.

Note that the MIME-Version header field is required at the top level of a message. It is not required for each body part of a multipart entity. It is required for the embedded headers of a body of type "message/rfc822" or "message/partial" if and only if the embedded message is itself claimed to be MIME-conformant.

It is not possible to fully specify how a mail reader that conforms with MIME as defined in this document should treat a message that might arrive in the future with some value of MIME-Version other than "1.0".

It is also worth noting that version control for specific media types is not accomplished using the MIME-Version mechanism. In particular, some formats (such as application/postscript) have version numbering conventions that are internal to the media format. Where such conventions exist, MIME does nothing to supersede them. Where no such conventions exist, a MIME media type might use a "version" parameter in the content-type field if necessary.

NOTE TO IMPLEMENTORS: When checking MIME-Version values any RFC 822 comment strings that are present must be ignored. In particular, the following four MIME-Version fields are equivalent:

MIME-Version: 1.0

MIME-Version: 1.0 (produced by MetaSend Vx.x)

MIME-Version: (produced by MetaSend Vx.x) 1.0

MIME-Version: 1.(produced by MetaSend Vx.x)0

In the absence of a MIME-Version field, a receiving mail user agent (whether conforming to MIME requirements or not) may optionally choose to interpret the body of the message according to local conventions. Many such conventions are currently in use and it should be noted that in practice non-MIME messages can contain just about anything.

It is impossible to be certain that a non-MIME mail message is actually plain text in the US-ASCII character set since it might well be a message that, using some set of nonstandard local conventions that predate MIME, includes text in another character set or non-textual data presented in a manner that cannot be automatically recognized (e.g., a uuencoded compressed UNIX tar file).

5. Content-Type Header Field

The purpose of the Content-Type field is to describe the data contained in the body fully enough that the receiving user agent can pick an appropriate agent or mechanism to present the data to the user, or otherwise deal with the data in an appropriate manner. The value in this field is called a media type.

HISTORICAL NOTE: The Content-Type header field was first defined in RFC 1049. RFC 1049 used a simpler and less powerful syntax, but one that is largely compatible with the mechanism given here.

The Content-Type header field specifies the nature of the data in the body of an entity by giving media type and subtype identifiers, and by providing auxiliary information that may be required for certain media types. After the media type and subtype names, the remainder

of the header field is simply a set of parameters, specified in an attribute=value notation. The ordering of parameters is not significant.

In general, the top-level media type is used to declare the general type of data, while the subtype specifies a specific format for that type of data. Thus, a media type of "image/xyz" is enough to tell a user agent that the data is an image, even if the user agent has no knowledge of the specific image format "xyz". Such information can be used, for example, to decide whether or not to show a user the raw data from an unrecognized subtype -- such an action might be reasonable for unrecognized subtypes of text, but not for unrecognized subtypes of image or audio. For this reason, registered subtypes of text, image, audio, and video should not contain embedded information that is really of a different type. Such compound formats should be represented using the "multipart" or "application" types.

Parameters are modifiers of the media subtype, and as such do not fundamentally affect the nature of the content. The set of meaningful parameters depends on the media type and subtype. Most parameters are associated with a single specific subtype. However, a given top-level media type may define parameters which are applicable to any subtype of that type. Parameters may be required by their defining content type or subtype or they may be optional. MIME implementations must ignore any parameters whose names they do not recognize.

For example, the "charset" parameter is applicable to any subtype of "text", while the "boundary" parameter is required for any subtype of the "multipart" media type.

There are NO globally-meaningful parameters that apply to all media types. Truly global mechanisms are best addressed, in the MIME model, by the definition of additional Content-* header fields.

An initial set of seven top-level media types is defined in RFC 2046. Five of these are discrete types whose content is essentially opaque as far as MIME processing is concerned. The remaining two are composite types whose contents require additional handling by MIME processors.

This set of top-level media types is intended to be substantially complete. It is expected that additions to the larger set of supported types can generally be accomplished by the creation of new subtypes of these initial types. In the future, more top-level types may be defined only by a standards-track extension to this standard. If another top-level type is to be used for any reason, it must be given a name starting with "X-" to indicate its non-standard status and to avoid a potential conflict with a future official name.

5.1. Syntax of the Content-Type Header Field

In the Augmented BNF notation of RFC 822, a Content-Type header field value is defined as follows:

```
content := "Content-Type" ":" type "/" subtype
           *(";" parameter)
           ; Matching of media type and subtype
           ; is ALWAYS case-insensitive.

type := discrete-type / composite-type

discrete-type := "text" / "image" / "audio" / "video" /
                 "application" / extension-token

composite-type := "message" / "multipart" / extension-token

extension-token := ietf-token / x-token

ietf-token := <An extension token defined by a
               standards-track RFC and registered
               with IANA.>
```

```
x-token := <The two characters "X-" or "x-" followed, with
           no intervening white space, by any token>

subtype := extension-token / iana-token

iana-token := <A publicly-defined extension token. Tokens
              of this form must be registered with IANA
              as specified in RFC 2048.>

parameter := attribute "=" value

attribute := token
             ; Matching of attributes
             ; is ALWAYS case-insensitive.

value := token / quoted-string

token := 1*<any (US-ASCII) CHAR except SPACE, CTLs,
           or tspecials>

tspecials :=  "(" / ")" / "<" / ">" / "@" /
              "," / ";" / ":" / "\" / <">
              "/" / "[" / "]" / "?" / "="
              ; Must be in quoted-string,
              ; to use within parameter values
```


Note that the definition of "tspecials" is the same as the RFC 822 definition of "specials" with the addition of the three characters "/", "?", and "=", and the removal of ".".

Note also that a subtype specification is MANDATORY -- it may not be omitted from a Content-Type header field. As such, there are no default subtypes.

The type, subtype, and parameter names are not case sensitive. For example, TEXT, Text, and TeXt are all equivalent top-level media types. Parameter values are normally case sensitive, but sometimes are interpreted in a case-insensitive fashion, depending on the

intended use. (For example, multipart boundaries are case-sensitive, but the "access-type" parameter for message/External-body is not case-sensitive.)

Note that the value of a quoted string parameter does not include the quotes. That is, the quotation marks in a quoted-string are not a part of the value of the parameter, but are merely used to delimit that parameter value. In addition, comments are allowed in accordance with RFC 822 rules for structured header fields. Thus the following two forms

Content-type: text/plain; charset=us-ascii (Plain text)

Content-type: text/plain; charset="us-ascii"

are completely equivalent.

Beyond this syntax, the only syntactic constraint on the definition of subtype names is the desire that their uses must not conflict. That is, it would be undesirable to have two different communities using "Content-Type: application/foobar" to mean two different things. The process of defining new media subtypes, then, is not intended to be a mechanism for imposing restrictions, but simply a mechanism for publicizing their definition and usage. There are, therefore, two acceptable mechanisms for defining new media subtypes:

(1) Private values (starting with "X-") may be defined bilaterally between two cooperating agents without outside registration or standardization. Such values cannot be registered or standardized.

(2) New standard values should be registered with IANA as described in RFC 2048.

The second document in this set, RFC 2046, defines the initial set of media types for MIME.

5.2. Content-Type Defaults

Default RFC 822 messages without a MIME Content-Type header are taken by this protocol to be plain text in the US-ASCII character set, which can be explicitly specified as:

```
Content-type: text/plain; charset=us-ascii
```

This default is assumed if no Content-Type header field is specified. It is also recommend that this default be assumed when a syntactically invalid Content-Type header field is encountered. In the presence of a MIME-Version header field and the absence of any Content-Type header field, a receiving User Agent can also assume that plain US-ASCII text was the sender's intent. Plain US-ASCII text may still be assumed in the absence of a MIME-Version or the presence of an syntactically invalid Content-Type header field, but the sender's intent might have been otherwise.

6. Content-Transfer-Encoding Header Field

Many media types which could be usefully transported via email are represented, in their "natural" format, as 8bit character or binary data. Such data cannot be transmitted over some transfer protocols. For example, RFC 821 (SMTP) restricts mail messages to 7bit US-ASCII data with lines no longer than 1000 characters including any trailing CRLF line separator.

It is necessary, therefore, to define a standard mechanism for encoding such data into a 7bit short line format. Proper labelling of unencoded material in less restrictive formats for direct use over less restrictive transports is also desireable. This document specifies that such encodings will be indicated by a new "Content-Transfer-Encoding" header field. This field has not been defined by any previous standard.

6.1. Content-Transfer-Encoding Syntax

The Content-Transfer-Encoding field's value is a single token specifying the type of encoding, as enumerated below. Formally:

```
encoding := "Content-Transfer-Encoding" ":" mechanism

mechanism := "7bit" / "8bit" / "binary" /
             "quoted-printable" / "base64" /
             ietf-token / x-token
```

These values are not case sensitive -- Base64 and BASE64 and bAsE64 are all equivalent. An encoding type of 7BIT requires that the body

is already in a 7bit mail-ready representation. This is the default value -- that is, "Content-Transfer-Encoding: 7BIT" is assumed if the Content-Transfer-Encoding header field is not present.

6.2. Content-Transfer-Encodings Semantics

This single Content-Transfer-Encoding token actually provides two pieces of information. It specifies what sort of encoding transformation the body was subjected to and hence what decoding operation must be used to restore it to its original form, and it specifies what the domain of the result is.

The transformation part of any Content-Transfer-Encodings specifies, either explicitly or implicitly, a single, well-defined decoding algorithm, which for any sequence of encoded octets either transforms it to the original sequence of octets which was encoded, or shows that it is illegal as an encoded sequence. Content-Transfer-Encodings transformations never depend on any additional external profile information for proper operation. Note that while decoders must produce a single, well-defined output for a valid encoding no such restrictions exist for encoders: Encoding a given sequence of octets to different, equivalent encoded sequences is perfectly legal.

Three transformations are currently defined: identity, the "quoted-printable" encoding, and the "base64" encoding. The domains are "binary", "8bit" and "7bit".

The Content-Transfer-Encoding values "7bit", "8bit", and "binary" all mean that the identity (i.e. NO) encoding transformation has been performed. As such, they serve simply as indicators of the domain of the body data, and provide useful information about the sort of encoding that might be needed for transmission in a given transport system. The terms "7bit data", "8bit data", and "binary data" are all defined in Section 2.

The quoted-printable and base64 encodings transform their input from an arbitrary domain into material in the "7bit" range, thus making it safe to carry over restricted transports. The specific definition of

the transformations are given below.

The proper Content-Transfer-Encoding label must always be used. Labelling unencoded data containing 8bit characters as "7bit" is not allowed, nor is labelling unencoded non-line-oriented data as anything other than "binary" allowed.

Unlike media subtypes, a proliferation of Content-Transfer-Encoding values is both undesirable and unnecessary. However, establishing only a single transformation into the "7bit" domain does not seem

possible. There is a tradeoff between the desire for a compact and efficient encoding of largely- binary data and the desire for a somewhat readable encoding of data that is mostly, but not entirely, 7bit. For this reason, at least two encoding mechanisms are necessary: a more or less readable encoding (quoted-printable) and a "dense" or "uniform" encoding (base64).

Mail transport for unencoded 8bit data is defined in RFC 1652. As of the initial publication of this document, there are no standardized Internet mail transports for which it is legitimate to include unencoded binary data in mail bodies. Thus there are no circumstances in which the "binary" Content-Transfer-Encoding is actually valid in Internet mail. However, in the event that binary mail transport becomes a reality in Internet mail, or when MIME is used in conjunction with any other binary-capable mail transport mechanism, binary bodies must be labelled as such using this mechanism.

NOTE: The five values defined for the Content-Transfer-Encoding field imply nothing about the media type other than the algorithm by which it was encoded or the transport system requirements if unencoded.

6.3. New Content-Transfer-Encodings

Implementors may, if necessary, define private Content-Transfer-Encoding values, but must use an x-token, which is a name prefixed by "X-", to indicate its non-standard status, e.g., "Content-Transfer-Encoding: x-my-new-encoding". Additional standardized Content-

Transfer-Encoding values must be specified by a standards-track RFC. The requirements such specifications must meet are given in RFC 2048. As such, all content-transfer-encoding namespace except that beginning with "X-" is explicitly reserved to the IETF for future use.

Unlike media types and subtypes, the creation of new Content-Transfer-Encoding values is STRONGLY discouraged, as it seems likely to hinder interoperability with little potential benefit

6.4. Interpretation and Use

If a Content-Transfer-Encoding header field appears as part of a message header, it applies to the entire body of that message. If a Content-Transfer-Encoding header field appears as part of an entity's headers, it applies only to the body of that entity. If an entity is of type "multipart" the Content-Transfer-Encoding is not permitted to have any value other than "7bit", "8bit" or "binary". Even more severe restrictions apply to some subtypes of the "message" type.

It should be noted that most media types are defined in terms of octets rather than bits, so that the mechanisms described here are mechanisms for encoding arbitrary octet streams, not bit streams. If a bit stream is to be encoded via one of these mechanisms, it must first be converted to an 8bit byte stream using the network standard bit order ("big-endian"), in which the earlier bits in a stream become the higher-order bits in a 8bit byte. A bit stream not ending at an 8bit boundary must be padded with zeroes. RFC 2046 provides a mechanism for noting the addition of such padding in the case of the application/octet-stream media type, which has a "padding" parameter.

The encoding mechanisms defined here explicitly encode all data in US-ASCII. Thus, for example, suppose an entity has header fields such as:

```
Content-Type: text/plain; charset=ISO-8859-1
Content-transfer-encoding: base64
```

This must be interpreted to mean that the body is a base64 US-ASCII encoding of data that was originally in ISO-8859-1, and will be in that character set again after decoding.

Certain Content-Transfer-Encoding values may only be used on certain media types. In particular, it is EXPRESSLY FORBIDDEN to use any encodings other than "7bit", "8bit", or "binary" with any composite media type, i.e. one that recursively includes other Content-Type fields. Currently the only composite media types are "multipart" and "message". All encodings that are desired for bodies of type multipart or message must be done at the innermost level, by encoding the actual body that needs to be encoded.

It should also be noted that, by definition, if a composite entity has a transfer-encoding value such as "7bit", but one of the enclosed entities has a less restrictive value such as "8bit", then either the outer "7bit" labelling is in error, because 8bit data are included, or the inner "8bit" labelling placed an unnecessarily high demand on the transport system because the actual included data were actually 7bit-safe.

NOTE ON ENCODING RESTRICTIONS: Though the prohibition against using content-transfer-encodings on composite body data may seem overly restrictive, it is necessary to prevent nested encodings, in which data are passed through an encoding algorithm multiple times, and must be decoded multiple times in order to be properly viewed. Nested encodings add considerable complexity to user agents: Aside from the obvious efficiency problems with such multiple encodings, they can obscure the basic structure of a message. In particular, they can imply that several decoding operations are necessary simply

to find out what types of bodies a message contains. Banning nested encodings may complicate the job of certain mail gateways, but this seems less of a problem than the effect of nested encodings on user agents.

Any entity with an unrecognized Content-Transfer-Encoding must be treated as if it has a Content-Type of "application/octet-stream", regardless of what the Content-Type header field actually says.

NOTE ON THE RELATIONSHIP BETWEEN CONTENT-TYPE AND CONTENT-TRANSFER-ENCODING: It may seem that the Content-Transfer-Encoding could be inferred from the characteristics of the media that is to be encoded, or, at the very least, that certain Content-Transfer-Encodings could be mandated for use with specific media types. There are several reasons why this is not the case. First, given the varying types of transports used for mail, some encodings may be appropriate for some combinations of media types and transports but not for others. (For example, in an 8bit transport, no encoding would be required for text in certain character sets, while such encodings are clearly required for 7bit SMTP.)

Second, certain media types may require different types of transfer encoding under different circumstances. For example, many PostScript bodies might consist entirely of short lines of 7bit data and hence require no encoding at all. Other PostScript bodies (especially those using Level 2 PostScript's binary encoding mechanism) may only be reasonably represented using a binary transport encoding. Finally, since the Content-Type field is intended to be an open-ended specification mechanism, strict specification of an association between media types and encodings effectively couples the specification of an application protocol with a specific lower-level transport. This is not desirable since the developers of a media type should not have to be aware of all the transports in use and what their limitations are.

6.5. Translating Encodings

The quoted-printable and base64 encodings are designed so that conversion between them is possible. The only issue that arises in such a conversion is the handling of hard line breaks in quoted-printable encoding output. When converting from quoted-printable to base64 a hard line break in the quoted-printable form represents a CRLF sequence in the canonical form of the data. It must therefore be converted to a corresponding encoded CRLF in the base64 form of the data. Similarly, a CRLF sequence in the canonical form of the data obtained after base64 decoding must be converted to a quoted-printable hard line break, but ONLY when converting text data.

6.6. Canonical Encoding Model

There was some confusion, in the previous versions of this RFC, regarding the model for when email data was to be converted to canonical form and encoded, and in particular how this process would affect the treatment of CRLFs, given that the representation of newlines varies greatly from system to system, and the relationship between content-transfer-encodings and character sets. A canonical model for encoding is presented in RFC 2049 for this reason.

6.7. Quoted-Printable Content-Transfer-Encoding

The Quoted-Printable encoding is intended to represent data that largely consists of octets that correspond to printable characters in the US-ASCII character set. It encodes the data in such a way that the resulting octets are unlikely to be modified by mail transport. If the data being encoded are mostly US-ASCII text, the encoded form of the data remains largely recognizable by humans. A body which is entirely US-ASCII may also be encoded in Quoted-Printable to ensure the integrity of the data should the message pass through a character-translating, and/or line-wrapping gateway.

In this encoding, octets are to be represented as determined by the following rules:

(1) (General 8bit representation) Any octet, except a CR or LF that is part of a CRLF line break of the canonical (standard) form of the data being encoded, may be represented by an "=" followed by a two digit hexadecimal representation of the octet's value. The digits of the hexadecimal alphabet, for this purpose, are "0123456789ABCDEF". Uppercase letters must be used; lowercase letters are not allowed. Thus, for example, the decimal value 12 (US-ASCII form feed) can be represented by "=0C", and the decimal value 61 (US-ASCII EQUAL SIGN) can be represented by "=3D". This rule must be followed except when the following rules allow an alternative encoding.

(2) (Literal representation) Octets with decimal values of 33 through 60 inclusive, and 62 through 126, inclusive, MAY be represented as the US-ASCII characters which correspond to those octets (EXCLAMATION POINT through LESS THAN, and GREATER THAN through TILDE,

respectively).

(3) (White Space) Octets with values of 9 and 32 MAY be represented as US-ASCII TAB (HT) and SPACE characters,

respectively, but MUST NOT be so represented at the end of an encoded line. Any TAB (HT) or SPACE characters on an encoded line MUST thus be followed on that line by a printable character. In particular, an "=" at the end of an encoded line, indicating a soft line break (see rule #5) may follow one or more TAB (HT) or SPACE characters. It follows that an octet with decimal value 9 or 32 appearing at the end of an encoded line must be represented according to Rule #1. This rule is necessary because some MTAs (Message Transport Agents, programs which transport messages from one user to another, or perform a portion of such transfers) are known to pad lines of text with SPACEs, and others are known to remove "white space" characters from the end of a line. Therefore, when decoding a Quoted-Printable body, any trailing white space on a line must be deleted, as it will necessarily have been added by intermediate transport agents.

(4) (Line Breaks) A line break in a text body, represented as a CRLF sequence in the text canonical form, must be represented by a (RFC 822) line break, which is also a CRLF sequence, in the Quoted-Printable encoding. Since the canonical representation of media types other than text do not generally include the representation of line breaks as CRLF sequences, no hard line breaks (i.e. line breaks that are intended to be meaningful and to be displayed to the user) can occur in the quoted-printable encoding of such types. Sequences like "=0D", "=0A", "=0A=0D" and "=0D=0A" will routinely appear in non-text data represented in quoted-printable, of course.

Note that many implementations may elect to encode the

local representation of various content types directly rather than converting to canonical form first, encoding, and then converting back to local representation. In particular, this may apply to plain text material on systems that use newline conventions other than a CRLF terminator sequence. Such an implementation optimization is permissible, but only when the combined canonicalization-encoding step is equivalent to performing the three steps separately.

(5) (Soft Line Breaks) The Quoted-Printable encoding REQUIRES that encoded lines be no more than 76 characters long. If longer lines are to be encoded with the Quoted-Printable encoding, "soft" line breaks

must be used. An equal sign as the last character on a encoded line indicates such a non-significant ("soft") line break in the encoded text.

Thus if the "raw" form of the line is a single unencoded line that says:

Now's the time for all folk to come to the aid of their country.

This can be represented, in the Quoted-Printable encoding, as:

```
Now's the time =
for all folk to come=
 to the aid of their country.
```

This provides a mechanism with which long lines are encoded in such a way as to be restored by the user agent. The 76 character limit does not count the trailing CRLF, but counts all other characters, including any equal signs.

Since the hyphen character ("-") may be represented as itself in the Quoted-Printable encoding, care must be taken, when encapsulating a quoted-printable encoded body inside one or more multipart entities, to ensure that the boundary delimiter does not appear anywhere in the

encoded body. (A good strategy is to choose a boundary that includes a character sequence such as "=_" which can never appear in a quoted-printable body. See the definition of multipart messages in RFC 2046.)

NOTE: The quoted-printable encoding represents something of a compromise between readability and reliability in transport. Bodies encoded with the quoted-printable encoding will work reliably over most mail gateways, but may not work perfectly over a few gateways, notably those involving translation into EBCDIC. A higher level of confidence is offered by the base64 Content-Transfer-Encoding. A way to get reasonably reliable transport through EBCDIC gateways is to also quote the US-ASCII characters

!"#$@[\]^`{|}~

according to rule #1.

Because quoted-printable data is generally assumed to be line-oriented, it is to be expected that the representation of the breaks between the lines of quoted-printable data may be altered in transport, in the same manner that plain text mail has always been altered in Internet mail when passing between systems with differing newline conventions. If such alterations are likely to constitute a

corruption of the data, it is probably more sensible to use the base64 encoding rather than the quoted-printable encoding.

NOTE: Several kinds of substrings cannot be generated according to the encoding rules for the quoted-printable content-transfer-encoding, and hence are formally illegal if they appear in the output of a quoted-printable encoder. This note enumerates these cases and suggests ways to handle such illegal substrings if any are encountered in quoted-printable data that is to be decoded.

(1) An "=" followed by two hexadecimal digits, one or both of which are lowercase letters in "abcdef", is formally illegal. A robust implementation might choose to recognize them as the corresponding uppercase letters.

(2) An "=" followed by a character that is neither a hexadecimal digit (including "abcdef") nor the CR character of a CRLF pair is illegal. This case can be the result of US-ASCII text having been included in a quoted-printable part of a message without itself having been subjected to quoted-printable encoding. A reasonable approach by a robust implementation might be to include the "=" character and the following character in the decoded data without any transformation and, if possible, indicate to the user that proper decoding was not possible at this point in the data.

(3) An "=" cannot be the ultimate or penultimate character in an encoded object. This could be handled as in case (2) above.

(4) Control characters other than TAB, or CR and LF as parts of CRLF pairs, must not appear. The same is true for octets with decimal values greater than 126. If found in incoming quoted-printable data by a decoder, a robust implementation might exclude them from the decoded data and warn the user that illegal characters were discovered.

(5) Encoded lines must not be longer than 76 characters, not counting the trailing CRLF. If longer lines are found in incoming, encoded data, a robust implementation might nevertheless decode the lines, and might report the erroneous encoding to the user.

WARNING TO IMPLEMENTORS: If binary data is encoded in quoted-printable, care must be taken to encode CR and LF characters as "=0D" and "=0A", respectively. In particular, a CRLF sequence in binary data should be encoded as "=0D=0A". Otherwise, if CRLF were

represented as a hard line break, it might be incorrectly decoded on platforms with different line break conventions.

For formalists, the syntax of quoted-printable data is described by the following grammar:

```
quoted-printable := qp-line *(CRLF qp-line)

qp-line := *(qp-segment transport-padding CRLF)
        qp-part transport-padding

qp-part := qp-section
        ; Maximum length of 76 characters

qp-segment := qp-section *(SPACE / TAB) "="
           ; Maximum length of 76 characters

qp-section := [*(ptext / SPACE / TAB) ptext]

ptext := hex-octet / safe-char

safe-char := <any octet with decimal value of 33 through
          60 inclusive, and 62 through 126>
          ; Characters not listed as "mail-safe" in
          ; RFC 2049 are also not recommended.

hex-octet := "=" 2(DIGIT / "A" / "B" / "C" / "D" / "E" / "F")
          ; Octet must be used for characters > 127, =,
          ; SPACEs or TABs at the ends of lines, and is
          ; recommended for any character not listed in
          ; RFC 2049 as "mail-safe".

transport-padding := *LWSP-char
                  ; Composers MUST NOT generate
                  ; non-zero length transport
                  ; padding, but receivers MUST
                  ; be able to handle padding
                  ; added by message transports.
```

IMPORTANT: The addition of LWSP between the elements shown in this BNF is NOT allowed since this BNF does not specify a structured header field.

6.8. Base64 Content-Transfer-Encoding

The Base64 Content-Transfer-Encoding is designed to represent arbitrary sequences of octets in a form that need not be humanly readable. The encoding and decoding algorithms are simple, but the encoded data are consistently only about 33 percent larger than the unencoded data. This encoding is virtually identical to the one used in Privacy Enhanced Mail (PEM) applications, as defined in RFC 1421.

A 65-character subset of US-ASCII is used, enabling 6 bits to be represented per printable character. (The extra 65th character, "=", is used to signify a special processing function.)

NOTE: This subset has the important property that it is represented identically in all versions of ISO 646, including US-ASCII, and all characters in the subset are also represented identically in all versions of EBCDIC. Other popular encodings, such as the encoding used by the uuencode utility, Macintosh binhex 4.0 [RFC-1741], and the base85 encoding specified as part of Level 2 PostScript, do not share these properties, and thus do not fulfill the portability requirements a binary transport encoding for mail must meet.

The encoding process represents 24-bit groups of input bits as output strings of 4 encoded characters. Proceeding from left to right, a 24-bit input group is formed by concatenating 3 8bit input groups. These 24 bits are then treated as 4 concatenated 6-bit groups, each of which is translated into a single digit in the base64 alphabet. When encoding a bit stream via the base64 encoding, the bit stream must be presumed to be ordered with the most-significant-bit first. That is, the first bit in the stream will be the high-order bit in the first 8bit byte, and the eighth bit will be the low-order bit in the first 8bit byte, and so on.

Each 6-bit group is used as an index into an array of 64 printable characters. The character referenced by the index is placed in the output string. These characters, identified in Table 1, below, are selected so as to be universally representable, and the set excludes characters with particular significance to SMTP (e.g., ".", CR, LF) and to the multipart boundary delimiters defined in RFC 2046 (e.g., "-").

Table 1: The Base64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

The encoded output stream must be represented in lines of no more than 76 characters each. All line breaks or other characters not found in Table 1 must be ignored by decoding software. In base64 data, characters other than those in Table 1, line breaks, and other white space probably indicate a transmission error, about which a warning message or even a message rejection might be appropriate under some circumstances.

Special processing is performed if fewer than 24 bits are available

at the end of the data being encoded. A full encoding quantum is always completed at the end of a body. When fewer than 24 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 6-bit groups. Padding at the end of the data is performed using the "=" character. Since all base64 input is an integral number of octets, only the following cases can arise: (1) the final quantum of encoding input is an integral multiple of 24 bits; here, the final unit of encoded output will be an integral multiple of 4 characters with no "=" padding, (2) the final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by two "=" padding characters, or (3) the final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be three characters followed by one "=" padding character.

Because it is used only for padding at the end of the data, the occurrence of any "=" characters may be taken as evidence that the end of the data has been reached (without truncation in transit). No

such assurance is possible, however, when the number of octets transmitted was a multiple of three and no "=" characters are present.

Any characters outside of the base64 alphabet are to be ignored in base64-encoded data.

Care must be taken to use the proper octets for line breaks if base64 encoding is applied directly to text material that has not been converted to canonical form. In particular, text line breaks must be converted into CRLF sequences prior to base64 encoding. The important thing to note is that this may be done directly by the encoder rather than in a prior canonicalization step in some implementations.

NOTE: There is no need to worry about quoting potential boundary delimiters within base64-encoded bodies within multipart entities because no hyphen characters are used in the base64 encoding.

7. Content-ID Header Field

In constructing a high-level user agent, it may be desirable to allow one body to make reference to another. Accordingly, bodies may be labelled using the "Content-ID" header field, which is syntactically identical to the "Message-ID" header field:

id := "Content-ID" ":" msg-id

Like the Message-ID values, Content-ID values must be generated to be world-unique.

The Content-ID value may be used for uniquely identifying MIME entities in several contexts, particularly for caching data referenced by the message/external-body mechanism. Although the Content-ID header is generally optional, its use is MANDATORY in implementations which generate data of the optional MIME media type "message/external-body". That is, each message/external-body entity must have a Content-ID field to permit caching of such data.

It is also worth noting that the Content-ID value has special semantics in the case of the multipart/alternative media type. This is explained in the section of RFC 2046 dealing with multipart/alternative.

8. Content-Description Header Field

The ability to associate some descriptive information with a given body is often desirable. For example, it may be useful to mark an "image" body as "a picture of the Space Shuttle Endeavor." Such text may be placed in the Content-Description header field. This header field is always optional.

description := "Content-Description" ":" *text

The description is presumed to be given in the US-ASCII character set, although the mechanism specified in RFC 2047 may be used for non-US-ASCII Content-Description values.

9. Additional MIME Header Fields

Future documents may elect to define additional MIME header fields for various purposes. Any new header field that further describes the content of a message should begin with the string "Content-" to allow such fields which appear in a message header to be distinguished from ordinary RFC 822 message header fields.

MIME-extension-field := <Any RFC 822 header field which
begins with the string
"Content-">

10. Summary

Using the MIME-Version, Content-Type, and Content-Transfer-Encoding header fields, it is possible to include, in a standardized way, arbitrary types of data with RFC 822 conformant mail messages. No restrictions imposed by either RFC 821 or RFC 822 are violated, and care has been taken to avoid problems caused by additional restrictions imposed by the characteristics of some Internet mail transport mechanisms (see RFC 2049).

The next document in this set, RFC 2046, specifies the initial set of media types that can be labelled and transported using these headers.

11. Security Considerations

Security issues are discussed in the second document in this set, RFC 2046.

12. Authors' Addresses

For more information, the authors of this document are best contacted via Internet mail:

Ned Freed
Innosoft International, Inc.
1050 East Garvey Avenue South
West Covina, CA 91790
USA

Phone: +1 818 919 3600
Fax: +1 818 919 3614
EMail: ned@innosoft.com

Nathaniel S. Borenstein
First Virtual Holdings
25 Washington Avenue
Morristown, NJ 07960
USA

Phone: +1 201 540 8967
Fax: +1 201 993 3032
EMail: nsb@nsb.fv.com

MIME is a result of the work of the Internet Engineering Task Force Working Group on RFC 822 Extensions. The chairman of that group, Greg Vaudreuil, may be reached at:

Gregory M. Vaudreuil
Octel Network Services
17080 Dallas Parkway
Dallas, TX 75248-1905
USA

EMail: Greg.Vaudreuil@Octel.Com

Appendix A -- Collected Grammar

This appendix contains the complete BNF grammar for all the syntax specified by this document.

By itself, however, this grammar is incomplete. It refers by name to several syntax rules that are defined by RFC 822. Rather than reproduce those definitions here, and risk unintentional differences between the two, this document simply refers the reader to RFC 822 for the remaining definitions. Wherever a term is undefined, it refers to the RFC 822 definition.

```
attribute := token
             ; Matching of attributes
             ; is ALWAYS case-insensitive.

composite-type := "message" / "multipart" / extension-token

content := "Content-Type" ":" type "/" subtype
           *(";" parameter)
           ; Matching of media type and subtype
           ; is ALWAYS case-insensitive.

description := "Content-Description" ":" *text

discrete-type := "text" / "image" / "audio" / "video" /
                 "application" / extension-token

encoding := "Content-Transfer-Encoding" ":" mechanism

entity-headers := [ content CRLF ]
                  [ encoding CRLF ]
                  [ id CRLF ]
                  [ description CRLF ]
                  *( MIME-extension-field CRLF )
```

```
extension-token := ietf-token / x-token

hex-octet := "=" 2(DIGIT / "A" / "B" / "C" / "D" / "E" / "F")
          ; Octet must be used for characters > 127, =,
          ; SPACEs or TABs at the ends of lines, and is
          ; recommended for any character not listed in
          ; RFC 2049 as "mail-safe".

iana-token := <A publicly-defined extension token. Tokens
           of this form must be registered with IANA
           as specified in RFC 2048.>
```



```
ietf-token := <An extension token defined by a
           standards-track RFC and registered
           with IANA.>

id := "Content-ID" ":" msg-id

mechanism := "7bit" / "8bit" / "binary" /
          "quoted-printable" / "base64" /
          ietf-token / x-token

MIME-extension-field := <Any RFC 822 header field which
                    begins with the string
                    "Content-">

MIME-message-headers := entity-headers
                    fields
                    version CRLF
                    ; The ordering of the header
                    ; fields implied by this BNF
                    ; definition should be ignored.

MIME-part-headers := entity-headers
                  [fields]
                  ; Any field not beginning with
                  ; "content-" can have no defined
                  ; meaning and may be ignored.
```

```
                ; The ordering of the header
                ; fields implied by this BNF
                ; definition should be ignored.

  parameter := attribute "=" value

  ptext := hex-octet / safe-char

  qp-line := *(qp-segment transport-padding CRLF)
          qp-part transport-padding

  qp-part := qp-section
          ; Maximum length of 76 characters

  qp-section := [*(ptext / SPACE / TAB) ptext]

  qp-segment := qp-section *(SPACE / TAB) "="
              ; Maximum length of 76 characters

  quoted-printable := qp-line *(CRLF qp-line)
```



```
  safe-char := <any octet with decimal value of 33 through
             60 inclusive, and 62 through 126>
             ; Characters not listed as "mail-safe" in
             ; RFC 2049 are also not recommended.

  subtype := extension-token / iana-token

  token := 1*<any (US-ASCII) CHAR except SPACE, CTLs,
          or tspecials>

  transport-padding := *LWSP-char
                ; Composers MUST NOT generate
                ; non-zero length transport
                ; padding, but receivers MUST
                ; be able to handle padding
                ; added by message transports.
```

```
tspecials :=  "(" / ")" / "<" / ">" / "@" /
              "," / ";" / ":" / "\" / <">
              "/" / "[" / "]" / "?" / "="
              ; Must be in quoted-string,
              ; to use within parameter values

type := discrete-type / composite-type

value := token / quoted-string

version := "MIME-Version" ":" 1*DIGIT "." 1*DIGIT

x-token := <The two characters "X-" or "x-" followed, with
           no  intervening white space, by any token>
```

Network Working Group H. Alvestrand
Request for Comments: 1766 UNINETT
Category: Standards Track March 1995

Tags for the Identification of Languages

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document describes a language tag for use in cases where it is desired to indicate the language used in an information object.

It also defines a Content-language: header, for use in the case where one desires to indicate the language of something that has RFC-822-like headers, like MIME body parts or Web documents, and a new parameter to the Multipart/Alternative type, to aid in the usage of the Content-Language: header.

1. Introduction

There are a number of languages spoken by human beings in this world.

A great number of these people would prefer to have information presented in a language that they understand.

In some contexts, it is possible to have information in more than one language, or it might be possible to provide tools for assisting in the understanding of a language (like dictionaries).

A prerequisite for any such function is a means of labelling the information content with an identifier for the language in which is is written.

In the tradition of solving only problems that we think we understand, this document specifies an identifier mechanism, and one possible use for it.

2. The Language tag

The language tag is composed of 1 or more parts: A primary language tag and a (possibly empty) series of subtags.

The syntax of this tag in RFC-822 EBNF is:

```
Language-Tag = Primary-tag *( "-" Subtag )
Primary-tag = 1*8ALPHA
Subtag = 1*8ALPHA
```

Whitespace is not allowed within the tag.

All tags are to be treated as case insensitive; there exist conventions for capitalization of some of them, but these should not be taken to carry meaning.

The namespace of language tags is administered by the IANA according to the rules in section 5 of this document.

The following registrations are predefined:

In the primary language tag:

- All 2-letter tags are interpreted according to ISO standard 639, "Code for the representation of names of languages" [ISO 639].

- The value "i" is reserved for IANA-defined registrations

- The value "x" is reserved for private use. Subtags of "x" will not be registered by the IANA.

- Other values cannot be assigned except by updating this standard.

The reason for reserving all other tags is to be open towards new revisions of ISO 639; the use of "i" and "x" is the minimum we can do here to be able to extend the mechanism to meet our requirements.

In the first subtag:

- All 2-letter codes are interpreted as ISO 3166 alpha-2 country codes denoting the area in which the language is used.

- Codes of 3 to 8 letters may be registered with the IANA by anyone who feels a need for it, according to the rules in

chapter 5 of this document.

The information in the subtag may for instance be:

- Country identification, such as en-US (this usage is described in ISO 639)

- Dialect or variant information, such as no-nynorsk or en-cockney

- Languages not listed in ISO 639 that are not variants of any listed language, which can be registered with the i-prefix, such as i-cherokee

- Script variations, such as az-arabic and az-cyrillic

In the second and subsequent subtag, any value can be registered.

NOTE: The ISO 639/ISO 3166 convention is that language names are written in lower case, while country codes are written in upper case.

This convention is recommended, but not enforced; the tags are case insensitive.

NOTE: ISO 639 defines a registration authority for additions to and changes in the list of languages in ISO 639. This authority is:

International Information Centre for Terminology (Infoterm)
P.O. Box 130
A-1021 Wien
Austria
Phone: +43 1 26 75 35 Ext. 312
Fax: +43 1 216 32 72

The following codes have been added in 1989 (nothing later): ug (Uigur), iu (Inuktitut, also called Eskimo), za (Zhuang), he (Hebrew, replacing iw), yi (Yiddish, replacing ji), and id (Indonesian, replacing in).

NOTE: The registration agency for ISO 3166 (country codes) is:

ISO 3166 Maintenance Agency Secretariat
c/o DIN Deutches Institut fuer Normung
Burggrafenstrasse 6
Postfach 1107
D-10787 Berlin
Germany
Phone: +49 30 26 01 320
Fax: +49 30 26 01 231

The country codes AA, QM-QZ, XA-XZ and ZZ are reserved by ISO 3166 as user-assigned codes.

2.1. Meaning of the language tag

The language tag always defines a language as spoken (or written) by human beings for communication of information to other human beings. Computer languages are explicitly excluded.

There is no guaranteed relationship between languages whose tags

start out with the same series of subtags; especially, they are NOT gurauranteed to be mutually comprehensible, although this will sometimes be the case.

Applications should always treat language tags as a single token; the division into main tag and subtags is an administrative mechanism, not a navigation aid.

The relationship between the tag and the information it relates to is defined by the standard describing the context in which it appears. So, this section can only give possible examples of its usage.

- For a single information object, it should be taken as the set of languages that is required for a complete comprehension of the complete object. Example: Simple text.

- For an aggregation of information objects, it should be taken as the set of languages used inside components of that aggregation. Examples: Document stores and libraries.

- For information objects whose purpose in life is providing alternatives, it should be regarded as a hint that the material inside is provided in several languages, and that one has to inspect each of the alternatives in order to find its language or languages. In this case, multiple languages need not mean that one needs to be multilingual to get complete understanding of the document. Example: MIME multipart/alternative.

- It would be possible to define (for instance) an SGML DTD that defines a <LANG xx> tag for indicating that following or contained text is written in this language, such that one could write "<LANG FR>C'est la vie</LANG>"; the Norwegian-speaking user could then access a French-Norwegian dictionary to find out what the quote meant.

3. The Content-language header

The Language header is intended for use in the case where one desires to indicate the language(s) of something that has RFC-822-like headers, like MIME body parts or Web documents.

The RFC-822 EBNF of the Language header is:

Language-Header = "Content-Language" ":" 1#Language-tag

Note that the Language-Header is allowed to list several languages in a comma-separated list.

Whitespace is allowed, which means also that one can place parenthesized comments anywhere in the language sequence.

3.1. Examples of Content-language values

NOTE: NONE of the subtags shown in this document have actually been assigned; they are used for illustration purposes only.

Norwegian official document, with parallel text in both official versions of Norwegian. (Both versions are readable by all Norwegians).

```
Content-Type: multipart/alternative;
      differences=content-language
Content-Language: no-nynorsk, no-bokmaal
```

Voice recording from the London docks

```
Content-type: audio/basic
Content-Language: en-cockney
```

Document in Sami, which does not have an ISO 639 code, and is spoken in several countries, but with about half the speakers in Norway, with six different, mutually incomprehensible dialects:

```
Content-type: text/plain; charset=iso-8859-10
Content-Language: i-sami-no (North Sami)
```

An English-French dictionary

```
Content-type: application/dictionary
Content-Language: en, fr (This is a dictionary)
```

An official EC document (in a few of its official languages)

Content-type: multipart/alternative
Content-Language: en, fr, de, da, el, it

An excerpt from Star Trek

Content-type: video/mpeg
Content-Language: x-klingon

4. Use of Content-Language with Multipart/Alternative

When using the Multipart/Alternative body part of MIME, it is possible to have the body parts giving the same information content in different languages. In this case, one should put a Content-Language header on each of the body parts, and a summary Content-Language header onto the Multipart/Alternative itself.

4.1. The differences parameter to multipart/alternative

As defined in RFC 1541, Multipart/Alternative only has one parameter: boundary.

The common usage of Multipart/Alternative is to have more than one format of the same message (f.ex. PostScript and ASCII).

The use of language tags to differentiate between different alternatives will certainly not lead all MIME UAs to present the most sensible body part as default.

Therefore, a new parameter is defined, to allow the configuration of MIME readers to handle language differences in a sensible manner.

Name: Differences
Value: One or more of
Content-Type
Content-Language

Further values can be registered with IANA; it must be the name of a header for which a definition exists in a published RFC. If not present, Differences=Content-Type is assumed.

The intent is that the MIME reader can look at these headers of the message component to do an intelligent choice of what to present to the user, based on knowledge about the user preferences and capabilities.

(The intent of having registration with IANA of the fields used in this context is to maintain a list of usages that a mail UA may expect to see, not to reject usages.)

(NOTE: The MIME specification [RFC 1521], section 7.2, states that headers not beginning with "Content-" are generally to be ignored in body parts. People defining a header for use with "differences=" should take note of this.)

The mechanism for deciding which body part to present is outside the scope of this document.

MIME EXAMPLE:

```
Content-Type: multipart/alternative; differences=Content-Language;
        boundary="limit"
Content-Language: en, fr, de

--limit
Content-Language: fr

Le renard brun et agile saute par dessus le chien paresseux
--limit
Content-Language: de
Content-Type: text/plain; charset=iso-8859-1
Content-Transfer-encoding: quoted-printable

Der schnelle braune Fuchs h=FCpft =FCber den faulen Hund
--limit
Content-Language: en
```

The quick brown fox jumps over the lazy dog
--limit--

When composing a message, the choice of sequence may be somewhat arbitrary. However, non-MIME mail readers will show the first body part first, meaning that this should most likely be the language understood by most of the recipients.

5. IANA registration procedure for language tags

Any language tag must start with an existing tag, and extend it.

This registration form should be used by anyone who wants to use a language tag not defined by ISO or IANA.

LANGUAGE TAG REGISTRATION FORM

Name of requester :
E-mail address of requester:
Tag to be registered :

English name of language :

Native name of language (transcribed into ASCII):

Reference to published description of the language (book or article):

The language form must be sent to <ietf-types@uninett.no> for a 2-week review period before submitting it to IANA. (This is an open

list. Requests to be added should be sent to <ietf-types-request@uninett.no>.)

When the two week period has passed, the language tag reviewer, who is appointed by the IETF Applications Area Director, either forwards the request to IANA@ISI.EDU, or rejects it because of significant objections raised on the list.

Decisions made by the reviewer may be appealed to the IESG.

All registered forms are available online in the directory ftp://ftp.isi.edu/in-notes/iana/assignments/languages/

6. Security Considerations

Security issues are not discussed in this memo.

7. Character set considerations

Codes may always be expressed using the US-ASCII character repertoire (a-z), which is present in most character sets.

The issue of deciding upon the rendering of a character set based on the language tag is not addressed in this memo; however, it is thought impossible to make such a decision correctly for all cases unless means of switching language in the middle of a text are defined (for example, a rendering engine that decides font based on Japanese or Chinese language will fail to work when a mixed Japanese-Chinese text is encountered)

8. Acknowledgements

This document has benefited from innumberable rounds of review and comments in various fora of the IETF and the Internet working groups. As so, any list of contributors is bound to be incomplete; please regard the following as only a selection from the group of people who

have contributed to make this document what it is today.

In alphabetical order:

Tim Berners-Lee, Nathaniel Borenstein, Jim Conklin, Dave Crocker, Ned Freed, Tim Goodwin, Olle Jarnefors, John Klensin, Keith Moore, Masataka Ohta, Keld Jorn Simonsen, Rhys Weatherley, and many, many others.

9. Author's Address

Harald Tveit Alvestrand
UNINETT
Pb. 6883 Elgeseter
N-7002 TRONDHEIM
NORWAY

EMail: Harald.T.Alvestrand@uninett.no
Phone: +47 73 59 70 94

10. References

[ISO 639]
ISO 639:1988 (E/F) - Code for the representation of names of languages - The International Organization for Standardization, 1st edition, 1988 17 pages Prepared by ISO/TC 37 - Terminology (principles and coordination).

[ISO 3166]
ISO 3166:1988 (E/F) - Codes for the representation of names of countries - The International Organization for Standardization, 3rd edition, 1988-08-15.

[RFC 1521]
Borenstein, N., and N. Freed, "MIME Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, Bellcore, Innosoft, September 1993.

[RFC 1327]
Kille, S., "Mapping between X.400(1988) / ISO 10021 and RFC 822", RFC 1327, University College London, May 1992.

ISO 639.2 Registration Authority

ISO 639 Joint Advisory Committee Home - ISO 639-1 Registration Authority Home
ISO 639-2 Registration Authority Home - Other Standards Maintained by the Library
Library of Congress Home

Codes for the Representation of Names of Languages

Alpha-3 codes arranged alphabetically by English name of language

Table of Contents

ISO 639-2 is the alpha-3 code. Where two codes are provided (22 languages total) the bibliographic code is given first and the terminology code is given second.

ISO 639-1 is the alpha-2 code. Multiple codes for the same language are to be considered synonyms.

A-B

Language Name (English)	Language Name (French)	639-2	639-1
Abkhazian	abkhaze	abk	ab
Achinese	aceh	ace	
Acoli	acoli	ach	
Adangme	adangme	ada	

Adygei; Adyghe	adyghé	ady	
Adyghe; Adygei	adygh	ady	
Afar	afar	aar	aa
Afrihili	afrihili	afh	
Afrikaans	afrikaans	afr	af
Afro-Asiatic (Other)	afro-asiatiques, autres langues	afa	
Akan	akan	aka	ak
Akkadian	akkadien	akk	
Albanian	albanais	alb/sqi*	sq
Aleut	aléoute	ale	
Algonquian languages	algonquines, langues	alg	
Altaic (Other)	altaïques, autres langues	tut	
Amharic	amharique	amh	am
Apache languages	apache	apa	
Arabic	arabe	ara	ar
Aragonese	aragonais	arg	an
Aramaic	araméen	arc	
Arapaho	arapaho	arp	
Araucanian	araucan	arn	
Arawak	arawak	arw	
Armenian	arménien	arm/hye*	hy
Artificial (Other)	artificielles, autres langues	art	
Assamese	assamais	asm	as
Asturian; Bable	asturien; bable	ast	
Athapascan languages	athapascanes, langues	ath	
Australian languages	australiennes, langues	aus	
Austronesian (Other)	malayo-polynésiennes, autres langues	map	
Avaric	avar	ava	av
Avestan	avestique	ave	ae
Awadhi	awadhi	awa	
Aymara	aymara	aym	ay
Azerbaijani	azéri	aze	az
Bable; Asturian	bable; asturien	ast	
Balinese	balinais	ban	
Baltic (Other)	baltiques, autres langues	bat	
Baluchi	baloutchi	bal	
Bambara	bambara	bam	bm
Bamileke languages	bamilékés, langues	bai	
Banda	banda	bad	
Bantu (Other)	bantoues, autres langues	bnt	
Basa	basa	bas	
Bashkir	bachkir	bak	ba
Basque	basque	baq/eus*	eu
Batak (Indonesia)	batak (Indonésie)	btk	
Beja	bedja	bej	

Belarusian	biélorusse	bel	be
Bemba	bemba	bem	
Bengali	bengali	ben	bn
Berber (Other)	berbères, autres langues	ber	
Bhojpuri	bhojpuri	bho	
Bihari	bihari	bih	bh
Bikol	bikol	bik	
Bilin; Blin	bilen; blin	byn	
Bini	bini	bin	
Bislama	bichlamar	bis	bi
Blin; Bilin	blin; bilen	byn	
Bokmål, Norwegian; Norwegian Bokmål	bokmål, norvégien; bokmål norvégien	nob	nb
Bosnian	bosniaque	bos	bs
Braj	braj	bra	
Breton	breton	bre	br
Buginese	bugi	bug	
Bulgarian	bulgare	bul	bg
Buriat	bouriate	bua	
Burmese	birman	bur/mya*	my

Go to the top

C-D

Language Name (English)	Language Name (French)	639-2	639-1
Caddo	caddo	cad	
Carib	caribe	car	
Castilian; Spanish	castillan; espagnol	spa	es
Catalan; Valencian	catalan; valencien	cat	ca
Caucasian (Other)	caucasiennes, autres langues	cau	
Cebuano	cebuano	ceb	
Celtic (Other)	celtiques, autres langues	cel	
Central American Indian (Other)	indiennes d'Amérique centrale, autres langues	cai	
Chagatai	djaghataï	chg	
Chamic languages	chames, langues	cmc	
Chamorro	chamorro	cha	ch
Chechen	tchétchène	che	ce
Cherokee	cherokee	chr	
Chewa; Chichewa; Nyanja	chewa, chichewa, nyanja	nya	ny
Cheyenne	cheyenne	chy	
Chibcha	chibcha	chb	
Chichewa; Chewa; Nyanja	chichewa; chewa; nyanja	nya	ny
Chinese	chinois	chi/zho*	zh
Chinook jargon	chinook, jargon	chn	

Chipewyan	chipewyan	chp	
Choctaw	choctaw	cho	
Chuang; Zhuang	chuang; zhuang	zha	za
Church Slavic; Slavonic; Church Slavonic; Old Bulgarian; Old Church Slavonic	slavon d'église; vieux slave; slavon liturgique; vieux bulgare	chu	cu
Church Slavonic; Church Slavic; Old Slavonic; Old Bulgarian; Old Church Slavonic	slavon liturgique; slavon d'église; vieux slave; vieux bulgare	chu	cu
Chuukese	chuuk	chk	
Chuvash	tchouvache	chv	cv
Classical Newari; Old Newari	newari classique	nwc	
Coptic	copte	cop	
Cornish	cornique	cor	kw
Corsican	corse	cos	co
Cree	cree	cre	cr
Creek	muskogee	mus	
Creoles and pidgins (Other)	créoles et pidgins divers	crp	
Creoles and pidgins, English-based (Other)	créoles et pidgins anglais, autres	cpe	
Creoles and pidgins, French-based (Other)	créoles et pidgins français, autres	cpf	
Creoles and pidgins, Portuguese-based (Other)	créoles et pidgins portugais, autres	cpp	
Crimean Tatar; Crimean Turkish	tatar de Crimé	crh	
Crimean Turkish; Crimean Tatar	tatar de Crimé	crh	
Croatian	croate	scr/hrv*	hr
Cushitic (Other)	couchitiques, autres langues	cus	
Czech	tchèque	cze/ces*	cs
Dakota	dakota	dak	
Danish	danois	dan	da
Dargwa	dargwa	dar	
Dayak	dayak	day	
Delaware	delaware	del	
Dinka	dinka	din	
Divehi	maldivien	div	dv
Dogri	dogri	doi	
Dogrib	dogrib	dgr	
Dravidian (Other)	dravidiennes, autres langues	dra	
Duala	douala	dua	
Dutch; Flemish	néerlandais; flamand	dut/nld*	nl
Dutch, Middle (ca. 1050-1350)	néerlandais moyen (ca. 1050-1350)	dum	
Dyula	dioula	dyu	
Dzongkha	dzongkha	dzo	dz

Go to the top

E-F

Language Name (English)	Language Name (French)	639-2	639-1
Efik	efik	efi	
Egyptian (Ancient)	égyptien	egy	
Ekajuk	ekajuk	eka	
Elamite	élamite	elx	
English	anglais	eng	en
English, Middle (1100-1500)	anglais moyen (1100-1500)	enm	
English, Old (ca.450-1100)	anglo-saxon (ca.450-1100)	ang	
Erzya	erza	myv	
Esperanto	espéranto	epo	eo
Estonian	estonien	est	et
Ewe	éwé	ewe	ee
Ewondo	éwondo	ewo	
Fang	fang	fan	
Fanti	fanti	fat	
Faroese	féroïen	fao	fo
Fijian	fidjien	fij	fj
Finnish	finnois	fin	fi
Finno-Ugrian (Other)	finno-ougriennes, autres langues	fiu	
Flemish; Dutch	flamand; néerlandais	dut/nld*	nl
Fon	fon	fon	
French	français	fre/fra*	fr
French, Middle (ca.1400-1600)	français moyen (1400-1600)	frm	
French, Old (842-ca.1400)	français ancien (842-ca.1400)	fro	
Frisian	frison	fry	fy
Friulian	frioulan	fur	
Fulah	peul	ful	ff

Go to the top

G-H

Language Name (English)	Language Name (French)	639-2	639-1
Ga	ga	gaa	
Gaelic; Scottish Gaelic	gaélique; gaélique écossais	gla	gd
Gallegan	galicien	glg	gl
Ganda	ganda	lug	lg
Gayo	gayo	gay	
Gbaya	gbaya	gba	
Geez	guèze	gez	
Georgian	géorgien	geo/kat*	ka

German	allemand	ger/deu*	de
German, Low; Saxon, Low; Low German; Low Saxon	allemand, bas; saxon, bas; bas allemand; bas saxon	nds	
German, Middle High (ca.1050-1500)	allemand, moyen haut (ca. 1050-1500)	gmh	
German, Old High (ca.750-1050)	allemand, vieux haut (ca. 750-1050)	goh	
Germanic (Other)	germaniques, autres langues	gem	
Gikuyu; Kikuyu	kikuyu	kik	ki
Gilbertese	kiribati	gil	
Gondi	gond	gon	
Gorontalo	gorontalo	gor	
Gothic	gothique	got	
Grebo	grebo	grb	
Greek, Ancient (to 1453)	grec ancien (jusqu'à 1453)	grc	
Greek, Modern (1453-)	grec moderne (après 1453)	gre/ell*	el
Greenlandic; Kalaallisut	groenlandais	kal	kl
Guarani	guarani	grn	gn
Gujarati	goudjrati	guj	gu
Gwich´in	gwich´in	gwi	
Haida	haida	hai	
Haitian; Haitian Creole	haïtien; créole haïtien	hat	ht
Haitian Creole; Haitian	créole haïtien; haïtien	hat	ht
Hausa	haoussa	hau	ha
Hawaiian	hawaïen	haw	
Hebrew	hébreu	heb	he
Herero	herero	her	hz
Hiligaynon	hiligaynon	hil	
Himachali	himachali	him	
Hindi	hindi	hin	hi
Hiri Motu	hiri motu	hmo	ho
Hittite	hittite	hit	
Hmong	hmong	hmn	
Hungarian	hongrois	hun	hu
Hupa	hupa	hup	

Go to the top

I-J

Language Name (English)	**Language Name (French)**	**639-2**	**639-1**
Iban	iban	iba	
Icelandic	islandais	ice/isl*	is
Ido	ido	ido	io
Igbo	igbo	ibo	ig
Ijo	ijo	ijo	

Iloko	ilocano	ilo	
Inari Sami	sami d'Inari	smn	
Indic (Other)	indo-aryennes, autres langues	inc	
Indo-European (Other)	indo-européennes, autres langues	ine	
Indonesian	indonésien	ind	id
Ingush	ingouche	inh	
Interlingua (International Auxiliary Language Association)	interlingua (langue auxiliaire internationale)	ina	ia
Interlingue	interlingue	ile	ie
Inuktitut	inuktitut	iku	iu
Inupiaq	inupiaq	ipk	ik
Iranian (Other)	iraniennes, autres langues	ira	
Irish	irlandais	gle	ga
Irish, Middle (900-1200)	irlandais moyen (900-1200)	mga	
Irish, Old (to 900)	irlandais ancien (jusqu'à 900)	sga	
Iroquoian languages	iroquoises, langues (famille)	iro	
Italian	italien	ita	it
Japanese	japonais	jpn	ja
Javanese	javanais	jav	jv
Judeo-Arabic	judéo-arabe	jrb	
Judeo-Persian	judéo-persan	jpr	

Go to the top

K-L

Language Name (English)	Language Name (French)	639-2	639-1
Kabardian	kabardien	kbd	
Kabyle	kabyle	kab	
Kachin	kachin	kac	
Kalaallisut; Greenlandic	groenlandais	kal	kl
Kalmyk	kalmouk	xal	
Kamba	kamba	kam	
Kannada	kannada	kan	kn
Kanuri	kanouri	kau	kr
Karachay-Balkar	karatchaï balkar	krc	
Kara-Kalpak	karakalpak	kaa	
Karen	karen	kar	
Kashmiri	kashmiri	kas	ks
Kashubian	kachoube	csb	
Kawi	kawi	kaw	
Kazakh	kazakh	kaz	kk
Khasi	khasi	kha	
Khmer	khmer	khm	km
Khoisan (Other)	khoisan, autres langues	khi	

Khotanese	khotanais	kho	
Kikuyu; Gikuyu	kikuyu	kik	ki
Kimbundu	kimbundu	kmb	
Kinyarwanda	rwanda	kin	rw
Kirghiz	kirghize	kir	ky
Klingon; tlhlngan-Hol	klingon	tlh	
Komi	komi	kom	kv
Kongo	kongo	kon	kg
Konkani	konkani	kok	
Korean	coréen	kor	ko
Kosraean	kosrae	kos	
Kpelle	kpellé	kpe	
Kru	krou	kro	
Kuanyama; Kwanyama	kuanyama; kwanyama	kua	kj
Kumyk	koumyk	kum	
Kurdish	kurde	kur	ku
Kurukh	kurukh	kru	
Kutenai	kutenai	kut	
Kwanyama, Kuanyama	kwanyama; kuanyama	kua	kj
Ladino	judéo-espagnol	lad	
Lahnda	lahnda	lah	
Lamba	lamba	lam	
Lao	lao	lao	lo
Latin	latin	lat	la
Latvian	letton	lav	lv
Letzeburgesch; Luxembourgish	luxembourgeois	ltz	lb
Lezghian	lezghien	lez	
Limburgan; Limburger; Limburgish	limbourgeois	lim	li
Limburger; Limburgan; Limburgish;	limbourgeois	lim	li
Limburgish; Limburger; Limburgan	limbourgeois	lim	li
Lingala	lingala	lin	ln
Lithuanian	lituanien	lit	lt
Lojban	lojban	jbo	
Low German; Low Saxon; German, Low; Saxon, Low	bas allemand; bas saxon; allemand, bas; saxon, bas	nds	
Low Saxon; Low German; Saxon, Low; German, Low	bas saxon; bas allemand; saxon, bas; allemand, bas	nds	
Lower Sorbian	bas-sorabe	dsb	
Lozi	lozi	loz	
Luba-Katanga	luba-katanga	lub	lu
Luba-Lulua	luba-lulua	lua	
Luiseno	luiseno	lui	
Lule Sami	sami de Lule	smj	
Lunda	lunda	lun	
Luo (Kenya and Tanzania)	luo (Kenya et Tanzanie)	luo	
Lushai	lushai	lus	

Luxembourgish; Letzeburgesch	luxembourgeois	ltz	lb

Go to the top

M-N

Language Name (English)	Language Name (French)	639-2	639-1
Macedonian	macédonien	mac/ mkd*	mk
Madurese	madourais	mad	
Magahi	magahi	mag	
Maithili	maithili	mai	
Makasar	makassar	mak	
Malagasy	malgache	mlg	mg
Malay	malais	may/ msa*	ms
Malayalam	malayalam	mal	ml
Maltese	maltais	mlt	mt
Manchu	mandchou	mnc	
Mandar	mandar	mdr	
Mandingo	mandingue	man	
Manipuri	manipuri	mni	
Manobo languages	manobo, langues	mno	
Manx	manx; mannois	glv	gv
Maori	maori	mao/mri*	mi
Marathi	marathe	mar	mr
Mari	mari	chm	
Marshallese	marshall	mah	mh
Marwari	marvari	mwr	
Masai	massaï	mas	
Mayan languages	maya, langues	myn	
Mende	mendé	men	
Micmac	micmac	mic	
Minangkabau	minangkabau	min	
Miscellaneous languages	diverses, langues	mis	
Mohawk	mohawk	moh	
Moksha	moksa	mdf	
Moldavian	moldave	mol	mo
Mon-Khmer (Other)	môn-khmer, autres langues	mkh	
Mongo	mongo	lol	
Mongolian	mongol	mon	mn
Mossi	moré	mos	
Multiple languages	multilingue	mul	
Munda languages	mounda, langues	mun	
Nahuatl	nahuatl	nah	

Nauru	nauruan	nau	na
Navaho, Navajo	navaho	nav	nv
Navajo; Navaho	navaho	nav	nv
Ndebele, North	ndébélé du Nord	nde	nd
Ndebele, South	ndébélé du Sud	nbl	nr
Ndonga	ndonga	ndo	ng
Neapolitan	napolitain	nap	
Nepali	népalais	nep	ne
Newari	newari	new	
Nias	nias	nia	
Niger-Kordofanian (Other)	nigéro-congolaises, autres langues	nic	
Nilo-Saharan (Other)	nilo-sahariennes, autres langues	ssa	
Niuean	niué	niu	
Nogai	nogaï; nogay	nog	
Norse, Old	norrois, vieux	non	
North American Indian (Other)	indiennes d'Amérique du Nord, autres langues	nai	
Northern Sami	sami du Nord	sme	se
North Ndebele	ndébélé du Nord	nde	nd
Norwegian	norvégien	nor	no
Norwegian Bokmål; Bokmål, Norwegian	norvégien bokmål; bokmål, norvégien	nob	nb
Norwegian Nynorsk; Nynorsk, Norwegian	norvégien nynorsk; nynorsk, norvégien	nno	nn
Nubian languages	nubiennes, langues	nub	
Nyamwezi	nyamwezi	nym	
Nyanja; Chichewa; Chewa	nyanja; chichewa; chewa	nya	ny
Nyankole	nyankolé	nyn	
Nynorsk, Norwegian; Norwegian Nynorsk	nynorsk, norvégien; norvégien nynorsk	nno	nn
Nyoro	nyoro	nyo	
Nzima	nzema	nzi	

Go to the top

O-P

Language Name (English)	**Language Name (French)**	**639-2**	**639-1**
Occitan (post 1500); Provençal	occitan (après 1500); provençal	oci	oc
Ojibwa	ojibwa	oji	oj
Old Bulgarian; Old Slavonic; Church Slavonic; Church Slavic; Old Church Slavonic	vieux bulgare; vieux slave; slavon liturgique; slavon d'église	chu	cu
Old Church Slavonic; Old Slavonic; Church Slavonic; Old Bulgarian; Church Slavic	vieux slave; slavon liturgique; vieux bulgare; slavon d'église	chu	cu
Old Newari; Classical Newari	newari classique	nwc	
Old Slavonic; Church Slavonic; Old Bulgarian; Church Slavic; Old Church Slavonic	vieux slave; slavon liturgique; vieux bulgare; slavon d'église	chu	cu
Oriya	oriya	ori	or
Oromo	galla	orm	om

Osage	osage	osa	
Ossetian; Ossetic	ossète	oss	os
Ossetic; Ossetian	ossète	oss	os
Otomian languages	otomangue, langues	oto	
Pahlavi	pahlavi	pal	
Palauan	palau	pau	
Pali	pali	pli	pi
Pampanga	pampangan	pam	
Pangasinan	pangasinan	pag	
Panjabi; Punjabi	pendjabi	pan	pa
Papiamento	papiamento	pap	
Papuan (Other)	papoues, autres langues	paa	
Persian	persan	per/fas*	fa
Persian, Old (ca.600-400 B.C.)	perse, vieux (ca. 600-400 av. J.-C.)	peo	
Philippine (Other)	philippines, autres langues	phi	
Phoenician	phénicien	phn	
Pohnpeian	pohnpei	pon	
Polish	polonais	pol	pl
Portuguese	portugais	por	pt
Prakrit languages	prâkrit	pra	
Provençal; Occitan (post 1500)	provençal; occitan (après 1500)	oci	oc
Provençal, Old (to 1500)	provençal ancien (jusqu'à 1500)	pro	
Punjabi; Panjabi	pendjabi	pan	pa
Pushto	pachto	pus	ps

Go to the top

Q-R

Language Name (English)	**Language Name (French)**	**639-2**	**639-1**
Quechua	quechua	que	qu
Raeto-Romance	rhéto-roman	roh	rm
Rajasthani	rajasthani	raj	
Rapanui	rapanui	rap	
Rarotongan	rarotonga	rar	
Reserved for local use	réservée à l'usage local	qaa-qtz	
Romance (Other)	romanes, autres langues	roa	
Romanian	roumain	rum/ron*	ro
Romany	tsigane	rom	
Rundi	rundi	run	rn
Russian	russe	rus	ru

Go to the top

S-T

Language Name (English)	Language Name (French)	639-2	639-1
Salishan languages	salish, langues	sal	
Samaritan Aramaic	samaritain	sam	
Sami languages (Other)	sami, autres langues	smi	
Samoan	samoan	smo	sm
Sandawe	sandawe	sad	
Sango	sango	sag	sg
Sanskrit	sanskrit	san	sa
Santali	santal	sat	
Sardinian	sarde	srd	sc
Sasak	sasak	sas	
Saxon, Low; German, Low; Low Saxon; Low German	saxon, bas; allemand, bas; bas saxon; bas allemand	nds	
Scots	écossais	sco	
Scottish Gaelic; Gaelic	gaélique écossais; gaélique	gla	gd
Selkup	selkoupe	sel	
Semitic (Other)	sémitiques, autres langues	sem	
Serbian	serbe	scc/srp*	sr
Serer	sérère	srr	
Shan	chan	shn	
Shona	shona	sna	sn
Sichuan Yi	yi de Sichuan	iii	ii
Sidamo	sidamo	sid	
Sign languages	langues des signes	sgn	
Siksika	blackfoot	bla	
Sindhi	sindhi	snd	sd
Sinhalese	singhalais	sin	si
Sino-Tibetan (Other)	sino-tibétaines, autres langues	sit	
Siouan languages	sioux, langues	sio	
Skolt Sami	sami skolt	sms	
Slave (Athapascan)	esclave (athapascan)	den	
Slavic (Other)	slaves, autres langues	sla	
Slovak	slovaque	slo/slk*	sk
Slovenian	slovène	slv	sl
Sogdian	sogdien	sog	
Somali	somali	som	so
Songhai	songhai	son	
Soninke	soninké	snk	
Sorbian languages	sorabes, langues	wen	
Sotho, Northern	sotho du Nord	nso	
Sotho, Southern	sotho du Sud	sot	st
South American Indian (Other)	indiennes d'Amérique du Sud, autres langues	sai	

Southern Sami	sami du Sud	sma	
South Ndebele	ndébélé du Sud	nbl	nr
Spanish; Castilian	espagnol; castillan	spa	es
Sukuma	sukuma	suk	
Sumerian	sumérien	sux	
Sundanese	soundanais	sun	su
Susu	soussou	sus	
Swahili	swahili	swa	sw
Swati	swati	ssw	ss
Swedish	suédois	swe	sv
Syriac	syriaque	syr	
Tagalog	tagalog	tgl	tl
Tahitian	tahitien	tah	ty
Tai (Other)	thaïes, autres langues	tai	
Tajik	tadjik	tgk	tg
Tamashek	tamacheq	tmh	
Tamil	tamoul	tam	ta
Tatar	tatar	tat	tt
Telugu	télougou	tel	te
Tereno	tereno	ter	
Tetum	tetum	tet	
Thai	thaï	tha	th
Tibetan	tibétain	tib/bod*	bo
Tigre	tigré	tig	
Tigrinya	tigrigna	tir	ti
Timne	temne	tem	
Tiv	tiv	tiv	
tlhlngan-Hol; Klingon	klingon	tlh	
Tlingit	tlingit	tli	
Tok Pisin	tok pisin	tpi	
Tokelau	tokelau	tkl	
Tonga (Nyasa)	tonga (Nyasa)	tog	
Tonga (Tonga Islands)	tongan (Îles Tonga)	ton	to
Tsimshian	tsimshian	tsi	
Tsonga	tsonga	tso	ts
Tswana	tswana	tsn	tn
Tumbuka	tumbuka	tum	
Tupi languages	tupi, langues	tup	
Turkish	turc	tur	tr
Turkish, Ottoman (1500-1928)	turc ottoman (1500-1928)	ota	
Turkmen	turkmène	tuk	tk
Tuvalu	tuvalu	tvl	
Tuvinian	touva	tyv	
Twi	twi	twi	tw

Go to the top

U-Z

Language Name (English)	Language Name (French)	639-2	639-1
Udmurt	oudmourte	udm	
Ugaritic	ougaritique	uga	
Uighur	ouïgour	uig	ug
Ukrainian	ukrainien	ukr	uk
Umbundu	umbundu	umb	
Undetermined	indéterminée	und	
Upper Sorbian	haut-sorabe	hsb	
Urdu	ourdou	urd	ur
Uzbek	ouszbek	uzb	uz
Vai	vaï	vai	
Valencian; Catalan	valencien; catalan	cat	ca
Venda	venda	ven	ve
Vietnamese	vietnamien	vie	vi
Volapük	volapük	vol	vo
Votic	vote	vot	
Wakashan languages	wakashennes, langues	wak	
Walamo	walamo	wal	
Walloon	wallon	wln	wa
Waray	waray	war	
Washo	washo	was	
Welsh	gallois	wel/cym*	cy
Wolof	wolof	wol	wo
Xhosa	xhosa	xho	xh
Yakut	iakoute	sah	
Yao	yao	yao	
Yapese	yapois	yap	
Yiddish	yiddish	yid	yi
Yoruba	yoruba	yor	yo
Yupik languages	yupik, langues	ypk	
Zande	zandé	znd	
Zapotec	zapotèque	zap	
Zenaga	zenaga	zen	
Zhuang; Chuang	zhuang; chuang	zha	za
Zulu	zoulou	zul	zu
Zuni	zuni	zun	

Top of Document

Comments on this document: iso639-2@loc.gov

ISO 639 Joint Advisory Committee Home - ISO 639-1 Registration Authority Home
ISO 639-2 Registration Authority Home - Other Standards Maintained by the Library
Library of Congress Home

Library of Congress >> Standards

Contact Us
March 22, 2004

Normenausschüsse

- Normenausschuss Gebrauchstauglichkeit und Dienstleistungen (NAGD)
- Ausschuss Gestaltung von Normen (AGN)
- Normenausschuss Gießereiwesen (GINA)
- Normenausschuss Gleitlager (NGL)
- Normenausschuss Grundlagen des Umweltschutzes (NAGUS)
- Normenausschuss Heiz-, Koch- und Wärmgerät (FNH)
- Normenausschuss Heiz- und Raumlufttechnik (NHRS)
- Normenausschuss Holzwirtschaft und Möbel (NHM)
- Normenausschuss Informationstechnik (NI)
- Normenausschuss Kältetechnik (FNKä)
- Normenausschuss Kautschuktechnik (FAKAU)
- Normenausschuss Kommunale Technik (NKT)
- Normenausschuss Kraftfahrzeuge (FAKRA)
- Normenausschuss Kunststoffe (FNK)
- Normenausschuss Laborgeräte und Laboreinrichtungen (FNLa)
- Normenausschuss Lebensmittel und landwirtschaftliche Produkte (NAL)
- Normenausschuss Lichttechnik (FNL)
- Normenausschuss Luft- und Raumfahrt (NL)
- Normenausschuss Maschinenbau (NAM)
- Normenausschuss Materialprüfung (NMP)
- Normenausschuss Mechanische Verbindungselemente (FMV)
- Normenausschuss Medizin (NAMed)
- Normenausschuss Nichteisenmetalle (FNNE)
- Ausschuss Normenpraxis (ANP)
- Ausschuss Normungsgrundsätze (ANG)
- Normenausschuss Papier und Pappe (NPa)
- Normenausschuss Persönliche Schutzausrüstung (NPS)
- Normenausschuss Pigmente und Füllstoffe (NPF)
- Normenausschuss Qualitätsmanagement, Statistik und Zertifizierungsgrundlagen (NQSZ)
- Normenausschuss Radiologie (NAR)
- Kommission Reinhaltung der Luft (KRdL) im VDI und DIN - Normenausschuss
- Normenausschuss Rettungsdienst und Krankenhaus (NARK)
- Normenausschuss Rohrleitungen und Dampfkesselanlagen (NARD)
- Normenausschuss Rundstahlketten (NRK)
- Normenausschuss Sachmerkmale (NSM)

- Normenausschuss Schienenfahrzeuge (FSF)
- Normenstelle Schiffs- und Meerestechnik (NSMT)
- Normenausschuss Schweißtechnik (NAS)
- Normenausschuss Sicherheitstechnische Grundsätze (NASG)
- Normenausschuss Sport- und Freizeitgerät (NASport)
- Normenausschuss Stahldraht und Stahldrahterzeugnisse (NAD)
- Normenausschuss Tankanlagen (NATank)
- Normenausschuss Technische Grundlagen (NATG)
- Normenausschuss Terminologie (NAT)
- Normenausschuss Textil und Textilmaschinen (Textilnorm)
- Normenausschuss Veranstaltungstechnik - Bühne, Beleuchtung und Ton (NVT)
- Normenausschuss Verpackungswesen (NAVp)
- Ausschuss Wälzlager (AWL)
- Normenausschuss Wasserwesen (NAW)
- Normenausschuss Werkstofftechnologie (NWT)
- Normenausschuss Werkzeuge und Spannzeuge (FWS)
- Normenausschuss Werkzeugmaschinen (NWM)

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
	<xsd:element name = "ProjektVerwaltung">
		<xsd:complexType>
			<xsd:sequence>
				<xsd:element ref = "Person" maxOccurs = "unbounded"/>
				<xsd:element ref = "Projekt" maxOccurs = "unbounded"/>
			</xsd:sequence>
		</xsd:complexType>
	</xsd:element>
	<xsd:element name = "Person">
		<xsd:complexType>
			<xsd:sequence>
				<xsd:element name = "Vorname" type = "xsd:token" maxOccurs = "unbounded"/
>
				<xsd:element name = "Nachname" type = "xsd:token"/>
				<xsd:element ref = "Qualifikationsprofil" minOccurs = "0"/>
			</xsd:sequence>
		</xsd:complexType>
	</xsd:element>
	<xsd:element name = "Projekt">
		<xsd:complexType/>
	</xsd:element>
	<xsd:element name = "Qualifikationsprofil">
		<xsd:complexType mixed = "true">
			<xsd:sequence>
				<xsd:element name = "Qualifikation" type = "xsd:string" minOccurs = "0"
maxOccurs = "unbounded"/>
				<xsd:element name = "Leistungsstufe" type = "xsd:string" minOccurs = "0"
maxOccurs = "unbounded"/>
			</xsd:sequence>
		</xsd:complexType>
	</xsd:element>
</xsd:schema>
```

```
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="PersonType">
        <xsd:sequence>
        <xsd:element name = "Vorname" type = "xsd:string"
                    maxOccurs = "unbounded"/>
        <xsd:element name = "Nachname" type = "xsd:string"/>
        <xsd:element ref = "Qualifikationsprofil" minOccurs = "0"/>
  </xsd:sequence>
  </xsd:complexType>

  <xsd:element name = "ProjektVerwaltung">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Person" type="PersonType" maxOccurs = "unbounded"/>
        <xsd:element ref = "Projekt" maxOccurs = "unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name = "Projekt">
    <xsd:complexType/>
  </xsd:element>

  <xsd:element name = "Qualifikationsprofil">
    <xsd:complexType mixed = "true">
      <xsd:sequence>
        <xsd:element name = "Qualifikation" type = "xsd:string"
                    minOccurs = "0" maxOccurs = "unbounded"/>
        <xsd:element name = "Leistungsstufe" type = "xsd:string"
                    minOccurs = "0" maxOccurs = "unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="parentType">
	<xsd:sequence>
		<xsd:element name="elementA" type="xsd:int"/>
	</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="childType">
<xsd:complexContent>
	<xsd:restriction base="parentType">
		<xsd:sequence>
			<xsd:element name="elementA" type="xsd:short"/>
		</xsd:sequence>
	</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="usage1" type="parentType"/>
<xsd:element name="usage2" type="childType"/>

</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<xsd:complexType name="parentElement">
		<xsd:sequence>
			<xsd:element name="elementA"/>
		</xsd:sequence>
	</xsd:complexType>

	<xsd:complexType name="childElement">
		<xsd:complexContent>
			<xsd:extension base="parentElement">
				<xsd:sequence>
					<xsd:element name="elementB"/>
				</xsd:sequence>
			</xsd:extension>
		</xsd:complexContent>
	</xsd:complexType>
</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">
      <xs:element name="Vorname">
            <xs:complexType>
                  <xs:simpleContent>
                        <xs:extension base="xs:string">
                              <xs:attribute
                                    name="rufname"
                                    type="xs:boolean"/>
                        </xs:extension>
                  </xs:simpleContent>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

Unicode Character Database

Unicode Data File Format

Revision	3.1.0
Authors	Mark Davis and Ken Whistler
Date	2001-02-28
This Version	http://www.unicode.org/Public/3.1-Update/UnicodeData-3.1.0.html
Previous Version	http://www.unicode.org/Public/3.0-Update1/UnicodeData-3.0.1.html
Latest Version	http://www.unicode.org/Public/UNIDATA/UnicodeData.html

Summary

This document describes the format and content of the UnicodeData.txt file in the Unicode Character Database (UCD).

Status

The file and the files described herein are part of the Unicode Character Database and governed by the UCD Terms of Use given below.

For general information on file formats and table formats, and the implications of normative vs informative properties, see UnicodeCharacterDatabase.html.

Warning: the information in this file does not completely describe the use and interpretation of Unicode character properties and behavior. It must be used in conjunction with the data in the other files in the UCD, and relies on the notation and definitions supplied in The Unicode Standard. All chapter references are to Version 3.1.0 of the standard.

Introduction

This document describes the format of the UnicodeData.txt file, which is one of the files in the Unicode Character Database. The document is divided into the following sections:

Field Formats

Each line represents the data for one encoded character in the Unicode Standard. (For information on the file format, see UCD File Format in UnicodeCharacterDatabase.html).

Every encoded character has a data entry, with the exception of certain special ranges, as detailed below.

- These ranges represented only by their start and end characters, since the properties in the file are uniform, except for code values (which are all sequential and assigned).
- The names of CJK ideograph characters and the names and decompositions of Hangul syllable characters are algorithmically derivable. (See the Unicode Standard and Unicode Standard Annex #15 for more information).
- Surrogate code values and private use characters have no names.
- The supplementary Private Use characters (U+F0000 .. U+FFFFD, U +100000 .. U+10FFFD) are listed as distinct ranges. These correspond to surrogate pairs where the first surrogate is in the High Surrogate Private Use section.

The exact ranges represented by start and end characters are:

- CJK Ideographs Extension A (U+3400 .. U+4DB5)
- CJK Ideographs (U+4E00 .. U+9FA5)
- Hangul Syllables (U+AC00 .. U+D7A3)
- Non-Private Use High Surrogates (U+D800 .. U+DB7F)
- Private Use High Surrogates (U+DB80 .. U+DBFF)
- Low Surrogates (U+DC00 .. U+DFFF)
- The Private Use Area (U+E000 .. U+F8FF)
- CJK Ideographs Extension B (U+20000 .. U+2A6D6)
- Plane 15 Private Use Area (U+F0000 .. U+FFFFD)
- Plane 16 Private Use Area (U+100000 .. U+10FFFD)

The following table describes the format and meaning of each field in a data entry

in the UnicodeData file.

Field	Name	N/I	Explanation
0	Code value	N	Code value.
1	Character name	N	These names match exactly the names published in Chapter 14 of the Unicode Standard, Version 3.0.
2	General Category	N	This is a useful breakdown into various "character types" which can be used as a default categorization in implementations. See below for a brief explanation.
3	Canonical Combining Classes	N	The classes used for the Canonical Ordering Algorithm in the Unicode Standard. These classes are also printed in Chapter 4 of the Unicode Standard.
4	Bidirectional Category	N	See the list below for an explanation of the abbreviations used in this field. These are the categories required by the Bidirectional Behavior Algorithm in the Unicode Standard. These categories are summarized in Chapter 3 of the Unicode Standard.
5	Character Decomposition Mapping	N	In the Unicode Standard, not all of the mappings are full (maximal) decompositions. Recursive application of look-up for decompositions will, in all cases, lead to a maximal decomposition. The decomposition mappings match exactly the decomposition mappings published with the character names in the Unicode Standard.
6	Decimal digit value	N	This is a numeric field. If the character has the decimal digit property, as specified in Chapter 4 of the Unicode Standard, the value of that digit is represented with an integer value in this field

7	Digit value	N	This is a numeric field. If the character represents a digit, not necessarily a decimal digit, the value is here. This covers digits which do not form decimal radix forms, such as the compatibility superscript digits
8	Numeric value	N	This is a numeric field. If the character has the numeric property, as specified in Chapter 4 of the Unicode Standard, the value of that character is represented with an integer or rational number in this field. This includes fractions as, e.g., "1/5" for U+2155 VULGAR FRACTION ONE FIFTH Also included are numerical values for compatibility characters such as circled numbers.
9	Mirrored	N	If the character has been identified as a "mirrored" character in bidirectional text, this field has the value "Y"; otherwise "N". The list of mirrored characters is also printed in Chapter 4 of the Unicode Standard.
10	Unicode 1.0 Name	I	This is the old name as published in Unicode 1.0. This name is only provided when it is significantly different from the current name for the character.
11	10646 comment field	I	This is the ISO 10646 comment field. It appears in parentheses in the 10646 names list, or contains an asterisk to mark an Annex P note.
12	Uppercase Mapping	N	Upper case equivalent mapping. If a character is part of an alphabet with case distinctions, and has a simple upper case equivalent, then the upper case equivalent is in this field. See the explanation below on case distinctions. These mappings are always one-to-one, not one-to-many or many-to-one. For full case mappings, see UTR #21

			and SpecialCasing.txt.
13	Lowercase Mapping	N	Similar to Uppercase mapping
14	Titlecase Mapping	N	Similar to Uppercase mapping

General Category

The values in this field are abbreviations for the following values. For more information, see the Unicode Standard.

> Note: the standard does not assign information to control characters (except for certain cases in the Bidirectional Algorithm). Implementations will generally also assign categories to certain control characters, notably CR and LF, according to platform conventions. See UAX #13: Unicode Newline Guidelines for more information.

Abbr.	Description
Lu	Letter, Uppercase
Ll	Letter, Lowercase
Lt	Letter, Titlecase
Lm	Letter, Modifier
Lo	Letter, Other
Mn	Mark, Non-Spacing
Mc	Mark, Spacing Combining
Me	Mark, Enclosing
Nd	Number, Decimal Digit
Nl	Number, Letter
No	Number, Other
Pc	Punctuation, Connector
Pd	Punctuation, Dash
Ps	Punctuation, Open
Pe	Punctuation, Close
Pi	Punctuation, Initial quote (may behave like Ps or Pe depending on usage)
Pf	Punctuation, Final quote (may behave like Ps or Pe depending on usage)

Po	Punctuation, Other
Sm	Symbol, Math
Sc	Symbol, Currency
Sk	Symbol, Modifier
So	Symbol, Other
Zs	Separator, Space
Zl	Separator, Line
Zp	Separator, Paragraph
Cc	Other, Control
Cf	Other, Format
Cs	Other, Surrogate
Co	Other, Private Use
Cn	Other, Not Assigned (no characters in the file have this property)

Note: The term "L&" is sometimes used to stand for Uppercase, Lowercase or Titlecase letters (Lu, Ll, or Lt).

Bidirectional Category

Please refer to Chapter 3 for an explanation of the algorithm for Bidirectional Behavior and an explanation of the significance of these categories. An up-to-date version can be found on Unicode Standard Annex #9: The Bidirectional Algorithm.

Type	Description
L	Left-to-Right
LRE	Left-to-Right Embedding
LRO	Left-to-Right Override
R	Right-to-Left
AL	Right-to-Left Arabic
RLE	Right-to-Left Embedding
RLO	Right-to-Left Override
PDF	Pop Directional Format
EN	European Number

ES	European Number Separator
ET	European Number Terminator
AN	Arabic Number
CS	Common Number Separator
NSM	Non-Spacing Mark
BN	Boundary Neutral
B	Paragraph Separator
S	Segment Separator
WS	Whitespace
ON	Other Neutrals

Character Decomposition Mapping

The tags supplied with certain decomposition mappings generally indicate formatting information. Where no such tag is given, the mapping is designated as canonical. Conversely, the presence of a formatting tag also indicates that the mapping is a compatibility mapping and not a canonical mapping. In the absence of other formatting information in a compatibility mapping, the tag is used to distinguish it from canonical mappings.

In some instances a canonical mapping or a compatibility mapping may consist of a single character. For a canonical mapping, this indicates that the character is a canonical equivalent of another single character. For a compatibility mapping, this indicates that the character is a compatibility equivalent of another single character. The compatibility formatting tags used are:

Tag	Description
<font>	A font variant (e.g. a blackletter form).
<noBreak>	A no-break version of a space or hyphen.
<initial>	An initial presentation form (Arabic).
<medial>	A medial presentation form (Arabic).
<final>	A final presentation form (Arabic).
<isolated>	An isolated presentation form (Arabic).
<circle>	An encircled form.
<super>	A superscript form.

<sub>	A subscript form.
<vertical>	A vertical layout presentation form.
<wide>	A wide (or zenkaku) compatibility character.
<narrow>	A narrow (or hankaku) compatibility character.
<small>	A small variant form (CNS compatibility).
<square>	A CJK squared font variant.
<fraction>	A vulgar fraction form.
<compat>	Otherwise unspecified compatibility character.

Reminder: There is a difference between decomposition and decomposition mapping. The decomposition mappings are defined in the UnicodeData, while the decomposition (also termed "full decomposition") is defined in Chapter 3 to use those mappings recursively.

- The canonical decomposition is formed by recursively applying the canonical mappings, then applying the canonical reordering algorithm.
- The compatibility decomposition is formed by recursively applying the canonical and compatibility mappings, then applying the canonical reordering algorithm.

Canonical Combining Classes

Value	Description
0:	Spacing, split, enclosing, reordrant, and Tibetan subjoined
1:	Overlays and interior
7:	Nuktas
8:	Hiragana/Katakana voicing marks
9:	Viramas
10:	Start of fixed position classes
199:	End of fixed position classes
200:	Below left attached
202:	Below attached
204:	Below right attached
208:	Left attached (reordrant around single base character)

210: Right attached

212: Above left attached

214: Above attached

216: Above right attached

218: Below left

220: Below

222: Below right

224: Left (reordrant around single base character)

226: Right

228: Above left

230: Above

232: Above right

233: Double below

234: Double above

240: Below (iota subscript)

Note: some of the combining classes in this list do not currently have members but are specified here for completeness.

Decompositions and Normalization

Decomposition is specified in Chapter 3. Unicode Standard Annex #15: Unicode Normalization Forms specifies the interaction between decomposition and normalization. That report specifies how the decompositions defined in UnicodeData.txt are used to derive normalized forms of Unicode text.

Note that as of the 2.1.9 update of the Unicode Character Database, the decompositions in the UnicodeData.txt file can be used to recursively derive the full decomposition in canonical order, without the need to separately apply canonical reordering. However, canonical reordering of combining character sequences must still be applied in decomposition when normalizing source text which contains any combining marks.

Case Mappings

There are a number of complications to case mappings that occur once the repertoire of characters is expanded beyond ASCII. For more information, see UTR #21: Case Mappings.

For compatibility with existing parsers, UnicodeData.txt only contains case mappings for characters where they are one-to-one mappings; it also omits information about context-sensitive case mappings. Information about these special cases can be found in a separate data file, SpecialCasing.txt.

Property Invariants

Values in UnicodeData.txt are subject to correction as errors are found; however, some characteristics of the categories themselves can be considered invariants. Applications may wish to take these invariants into account when choosing how to implement character properties. For more information, see Unicode Policies.

The following is a partial list of known invariants for the Unicode Character Database.

Database Fields

- The number of fields in UnicodeData.txt is fixed.
- The order of the fields is also fixed.
    - Any additional information about character properties to be added in the future will appear in separate data tables, rather than being added on to the existing table or by subdivision or reinterpretation of existing fields.

General Category

- There will never be more than 32 General Category values.
    - It is very unlikely that the Unicode Technical Committee will subdivide the General Category partition any further, since that can cause implementations to misbehave. Because the General Category is limited to 32 values, 5 bits can be used to represent the information, and a 32-bit integer can be used as a bitmask to represent arbitrary sets of categories.

Combining Classes

- Combining classes are limited to the values 0 to 255.
    - In practice, there are far fewer than 256 values used. Implementations may take advantage of this fact for compression, since only the ordering of the non-zero values matters for the Canonical Reordering Algorithm. It is possible for up to 256 values to be used in the future; however, UTC decisions in the future may restrict the number of values to 128, since this has implementation advantages. [Signed bytes can be used without widening to ints in Java, for example.]

- All characters other than those of General Category M* have the combining class 0.
    - Currently, all characters other than those of General Category Mn have the value 0. However, some characters of General Category Me or Mc may be given non-zero values in the future.
    - The precise values above the value 0 are not invariant--only the relative ordering is considered normative. For example, it is not guaranteed in future versions that the class of U+05B4 will be precisely 14.

Canonical Decomposition

- Canonical mappings are always in canonical order.
- Canonical mappings have only the first of a pair possibly further decomposing.
- Canonical decompositions are "transparent" to other character data:
    - `BIDI(a) = BIDI(principal(canonicalDecomposition(a))`
    - `Category(a) = Category(principal(canonicalDecomposition(a))`
    - `CombiningClass(a) = CombiningClass(principal(canonicalDecomposition(a))`
      where principal(a) is the first character not of type Mn, or the first character if all characters are of type Mn.
- However, because there are sometimes missing case pairs, and because of some legacy characters, it is only generally true that:
    - `upper(canonicalDecomposition(a)) = canonicalDecomposition(upper(a))`
    - `lower(canonicalDecomposition(a)) = canonicalDecomposition(lower(a))`
    - `title(canonicalDecomposition(a)) = canonicalDecomposition(title(a))`

Modification History

This section provides a summary of the changes between update versions of the Unicode Standard.

Unicode 3.1

Modifications made for Version 3.0.1 of UnicodeData.txt include:

- Addition of 2237 new entries, to cover new characters and new ranges of unified Han characters encoded in Unicode 3.1.
- Changed General Category value of 16EE..16F0 (Runic golden numbers) from No to Nl.

Unicode 3.0.1

Modifications made for Version 3.0.1 of UnicodeData.txt include:

- Added 5- and 6-digit representation of code points past U+FFFF.
- Added Private Use range definitions for Planes 15 and 16.
- Minor additions for the 10646 comment field.

Unicode 3.0.0

Modifications made for Version 3.0.0 of UnicodeData.txt include many new characters and a number of property changes. These are summarized in Appendex D of The Unicode Standard, Version 3.0.

Unicode 2.1.9

Modifications made for Version 2.1.9 of UnicodeData.txt include:

- Corrected combining class for U+05AE HEBREW ACCENT ZINOR.
- Corrected combining class for U+20E1 COMBINING LEFT RIGHT ARROW ABOVE
- Corrected combining class for U+0F35 and U+0F37 to 220.
- Corrected combining class for U+0F71 to 129.
- Added a decomposition for U+0F0C TIBETAN MARK DELIMITER TSHEG BSTAR.
- Added decompositions for several Greek symbol letters: U+03D0..U+03D2, U+03D5, U+03D6, U+03F0..U+03F2.
- Removed decompositions from the conjoining jamo block: U+1100..U+11F8.
- Changes to decomposition mappings for some Tibetan vowels for consistency in normalization. (U+0F71, U+0F73, U+0F77, U+0F79, U+0F81)
- Updated the decomposition mappings for several Vietnamese characters with two diacritics (U+1EAC, U+1EAD, U+1EB6, U+1EB7, U+1EC6, U+1EC7, U+1ED8, U+1ED9), so that the recursive decomposition can be generated directly in canonically reordered form (not a normative change).
- Updated the decomposition mappings for several Arabic compatibility characters involving shadda (U+FC5E..U+FC62, U+FCF2..U+FCF4), and two Latin characters (U+1E1C, U+1E1D), so that the decompositions are generated directly in canonically reordered form (not a normative change).
- Changed BIDI category for: U+00A0 NO-BREAK SPACE, U+2007 FIGURE SPACE, U+2028 LINE SEPARATOR.
- Changed BIDI category for extenders of General Category Lm: U+3005, U+3021..U+3035, U+FF9E, U+FF9F.
- Changed General Category and BIDI category for the Greek numeral signs: U+0374, U+0375.
- Corrected General Category for U+FFE8 HALFWIDTH FORMS LIGHT VERTICAL.
- Added Unicode 1.0 names for many Tibetan characters (informative).

Unicode 2.1.8

Modifications made for Version 2.1.8 of UnicodeData.txt include:

- Added combining class 240 for U+0345 COMBINING GREEK YPOGEGRAMMENI so that decompositions involving iota subscript are derivable directly in canonically reordered form; this also has a bearing on simplification of casing of polytonic Greek.
- Changes in decompositions related to Greek tonos. These result from the clarification that monotonic Greek "tonos" should be equated with U+0301 COMBINING ACUTE, rather than with U+030D COMBINING VERTICAL LINE ABOVE. (All Greek characters in the Greek block involving "tonos"; some Greek characters in the polytonic Greek in the 1FXX block.)
- Changed decompositions involving dialytika tonos. (U+0390, U+03B0)
- Changed ternary decompositions to binary. (U+0CCB, U+FB2C, U+FB2D) These changes simplify normalization.
- Removed canonical decomposition for Latin Candrabindu. (U+0310)
- Corrected error in canonical decomposition for U+1FF4.
- Added compatibility decompositions to clarify collation tables. (U+2100, U+2101, U+2105, U+2106, U+1E9A)
- A series of general category changes to assist the convergence of of Unicode definition of identifier with ISO TR 10176:
  - So > Lo: U+0950, U+0AD0, U+0F00, U+0F88..U+0F8B
  - Po > Lo: U+0E2F, U+0EAF, U+3006
  - Lm > Sk: U+309B, U+309C
  - Po > Pc: U+30FB, U+FF65
  - Ps/Pe > Mn: U+0F3E, U+0F3F
- A series of bidi property changes for consistency.
  - L > ET: U+09F2, U+09F3
  - ON > L: U+3007
  - L > ON: U+0F3A..U+0F3D, U+037E, U+0387
- Add case mapping: U+01A6 <-> U+0280
- Updated symmetric swapping value for guillemets: U+00AB, U+00BB, U+2039, U+203A.
- Changes to combining class values. Most Indic fixed position class non-spacing marks were changed to combining class 0. This fixes some inconsistencies in how canonical reordering would apply to Indic scripts, including Tibetan. Indic interacting top/bottom fixed position classes were merged into single (non-zero) classes as part of this change. Tibetan subjoined consonants are changed from combining class 6 to combining class 0. Thai pinthu (U+0E3A) moved to combining class 9. Moved two Devanagari stress marks into generic above and below combining classes (U+0951, U+0952).
- Corrected placement of semicolon near symmetric swapping field. (U+FA0E, etc., scattered positions to U+FA29)

Version 2.1.7

This version was for internal change tracking only, and never publicly released.

Version 2.1.6

This version was for internal change tracking only, and never publicly released.

Unicode 2.1.5

Modifications made for Version 2.1.5 of UnicodeData.txt include:

- Changed decomposition for U+FF9E and U+FF9F so that correct collation weighting will automatically result from the canonical equivalences.
- Removed canonical decompositions for U+04D4, U+04D5, U+04D8, U+04D9, U+04E0, U+04E1, U+04E8, U+04E9 (the implication being that no canonical equivalence is claimed between these 8 characters and similar Latin letters), and updated 4 canonical decompositions for U+04DB, U+04DC, U+04EA, U+04EB to reflect the implied difference in the base character.
- Added Pi, and Pf categories and assigned the relevant quotation marks to those categories, based on the Unicode Technical Corrigendum on Quotation Characters.
- Updating of many bidi properties, following the advice of the ad hoc committee on bidi, and to make the bidi properties of compatibility characters more consistent.
- Changed category of several Tibetan characters: U+0F3E, U+0F3F, U+0F88..U+0F8B to make them non-combining, reflecting the combined opinion of Tibetan experts.
- Added case mapping for U+03F2.
- Corrected case mapping for U+0275.
- Added titlecase mappings for U+03D0, U+03D1, U+03D5, U+03D6, U+03F0.. U+03F2.
- Corrected compatibility label for U+2121.
- Add specific entries for all the CJK compatibility ideographs, U+F900..U+FA2D, so the canonical decomposition for each (the URO character it is equivalent to) can be carried in the database.

Version 2.1.4

This version was for internal change tracking only, and never publicly released.

Version 2.1.3

This version was for internal change tracking only, and never publicly released.

Unicode 2.1.2

Modifications made in updating UnicodeData.txt to Version 2.1.2 for the Unicode Standard, Version 2.1 (from Version 2.0) include:

- Added two characters (U+20AC and U+FFFC).
- Amended bidi properties for U+0026, U+002E, U+0040, U+2007.
- Corrected case mappings for U+018E, U+019F, U+01DD, U+0258, U+0275, U+03C2, U+1E9B.
- Changed combining order class for U+0F71.
- Corrected canonical decompositions for U+0F73, U+1FBE.
- Changed decomposition for U+FB1F from compatibility to canonical.
- Added compatibility decompositions for U+FBE8, U+FBE9, U+FBF9..U+FBFB.
- Corrected compatibility decompositions for U+2469, U+246A, U+3358.

Version 2.1.1

This version was for internal change tracking only, and never publicly released.

Unicode 2.0.0

The modifications made in updating UnicodeData.txt for the Unicode Standard, Version 2.0 include:

- Fixed decompositions with TONOS to use correct NSM: 030D.
- Removed old Hangul Syllables; mapping to new characters are in a separate table.
- Marked compatibility decompositions with additional tags.
- Changed old tag names for clarity.
- Revision of decompositions to use first-level decomposition, instead of maximal decomposition.
- Correction of all known errors in decompositions from earlier versions.
- Added control code names (as old Unicode names).
- Added Hangul Jamo decompositions.
- Added Number category to match properties list in book.
- Fixed categories of Koranic Arabic marks.
- Fixed categories of precomposed characters to match decomposition where possible.
- Added Hebrew cantillation marks and the Tibetan script.
- Added place holders for ranges such as CJK Ideographic Area and the Private Use Area.
- Added categories Me, Sk, Pc, Nl, Cs, Cf, and rectified a number of mistakes in the database.

UCD Terms of Use

Disclaimer

The Unicode Character Database is provided as is by Unicode, Inc. No claims are made as to fitness for any particular purpose. No warranties of any kind are expressed or implied. The recipient agrees to determine applicability of information provided. If this file has been purchased on magnetic or optical media from Unicode, Inc., the sole remedy for any claim will be exchange of defective media within 90 days of receipt.

This disclaimer is applicable for all other data files accompanying the Unicode Character Database, some of which have been compiled by the Unicode Consortium, and some of which have been supplied by other sources.

Limitations on Rights to Redistribute This Data

Recipient is granted the right to make copies in any form for internal distribution and to freely use the information supplied in the creation of products supporting the Unicode™ Standard. The files in the Unicode Character Database can be redistributed to third parties or other organizations (whether for profit or not) as long as this notice and the disclaimer notice are retained. Information can be extracted from these files and used in documentation or programs, as long as there is an accompanying notice indicating the source.

Related Links

Technical Work

Roadmaps

Superseded Technical Report

UAX#21: Case Mappings has been incorporated into the Unicode Standard Version 4.0, and is thus now superseded. The last version of that document before it was superseded can be found at http://www.unicode.org/reports/tr21/tr21-5.html

The text has been incorporated into the following sections:

3.13 Default Case Operations
4.2 Case Normative
5.18 Case Mappings

Last updated: - Invalid Date

```
# DerivedGeneralCategory-3.1.0.txt
#
# Unicode Character Database: Derived Property Data
# Generated algorithmically from the Unicode Character Database
# For documentation, see DerivedProperties.html
# Date: 2001-03-02 00:03:25.1 GMT [MD]
# Note: Unassigned and Noncharacter codepoints are omitted,
#       except when listing Noncharacter or Cn.
# ================================================


# ================================================
# General Category (listing UnicodeData.txt, field 2: see UnicodeData.html)
# ================================================

0220..0221    ; Cn #   [2]
0234..024F    ; Cn #  [28]
02AE..02AF    ; Cn #   [2]
02EF..02FF    ; Cn #  [17]
034F..035F    ; Cn #  [17]
0363..0373    ; Cn #  [17]
0376..0379    ; Cn #   [4]
037B..037D    ; Cn #   [3]
037F..0383    ; Cn #   [5]
038B          ; Cn #
038D          ; Cn #
03A2          ; Cn #
03CF          ; Cn #
03D8..03D9    ; Cn #   [2]
03F6..03FF    ; Cn #  [10]
0487          ; Cn #
048A..048B    ; Cn #   [2]
04C5..04C6    ; Cn #   [2]
04C9..04CA    ; Cn #   [2]
04CD..04CF    ; Cn #   [3]
04F6..04F7    ; Cn #   [2]
04FA..0530    ; Cn #  [55]
0557..0558    ; Cn #   [2]
0560          ; Cn #
0588          ; Cn #
058B..0590    ; Cn #   [6]
05A2          ; Cn #
05BA          ; Cn #
05C5..05CF    ; Cn #  [11]
05EB..05EF    ; Cn #   [5]
```

05F5..060B ; Cn # [23]
060D..061A ; Cn # [14]
061C..061E ; Cn # [3]
0620 ; Cn #
063B..063F ; Cn # [5]
0656..065F ; Cn # [10]
066E..066F ; Cn # [2]
06EE..06EF ; Cn # [2]
06FF ; Cn #
070E ; Cn #
072D..072F ; Cn # [3]
074B..077F ; Cn # [53]
07B1..0900 ; Cn # [336]
0904 ; Cn #
093A..093B ; Cn # [2]
094E..094F ; Cn # [2]
0955..0957 ; Cn # [3]
0971..0980 ; Cn # [16]
0984 ; Cn #
098D..098E ; Cn # [2]
0991..0992 ; Cn # [2]
09A9 ; Cn #
09B1 ; Cn #
09B3..09B5 ; Cn # [3]
09BA..09BB ; Cn # [2]
09BD ; Cn #
09C5..09C6 ; Cn # [2]
09C9..09CA ; Cn # [2]
09CE..09D6 ; Cn # [9]
09D8..09DB ; Cn # [4]
09DE ; Cn #
09E4..09E5 ; Cn # [2]
09FB..0A01 ; Cn # [7]
0A03..0A04 ; Cn # [2]
0A0B..0A0E ; Cn # [4]
0A11..0A12 ; Cn # [2]
0A29 ; Cn #
0A31 ; Cn #
0A34 ; Cn #
0A37 ; Cn #
0A3A..0A3B ; Cn # [2]
0A3D ; Cn #
0A43..0A46 ; Cn # [4]
0A49..0A4A ; Cn # [2]
0A4E..0A58 ; Cn # [11]
0A5D ; Cn #

0A5F..0A65 ; Cn # [7]
0A75..0A80 ; Cn # [12]
0A84 ; Cn #
0A8C ; Cn #
0A8E ; Cn #
0A92 ; Cn #
0AA9 ; Cn #
0AB1 ; Cn #
0AB4 ; Cn #
0ABA..0ABB ; Cn # [2]
0AC6 ; Cn #
0ACA ; Cn #
0ACE..0ACF ; Cn # [2]
0AD1..0ADF ; Cn # [15]
0AE1..0AE5 ; Cn # [5]
0AF0..0B00 ; Cn # [17]
0B04 ; Cn #
0B0D..0B0E ; Cn # [2]
0B11..0B12 ; Cn # [2]
0B29 ; Cn #
0B31 ; Cn #
0B34..0B35 ; Cn # [2]
0B3A..0B3B ; Cn # [2]
0B44..0B46 ; Cn # [3]
0B49..0B4A ; Cn # [2]
0B4E..0B55 ; Cn # [8]
0B58..0B5B ; Cn # [4]
0B5E ; Cn #
0B62..0B65 ; Cn # [4]
0B71..0B81 ; Cn # [17]
0B84 ; Cn #
0B8B..0B8D ; Cn # [3]
0B91 ; Cn #
0B96..0B98 ; Cn # [3]
0B9B ; Cn #
0B9D ; Cn #
0BA0..0BA2 ; Cn # [3]
0BA5..0BA7 ; Cn # [3]
0BAB..0BAD ; Cn # [3]
0BB6 ; Cn #
0BBA..0BBD ; Cn # [4]
0BC3..0BC5 ; Cn # [3]
0BC9 ; Cn #
0BCE..0BD6 ; Cn # [9]
0BD8..0BE6 ; Cn # [15]
0BF3..0C00 ; Cn # [14]

0C04 ; Cn #
0C0D ; Cn #
0C11 ; Cn #
0C29 ; Cn #
0C34 ; Cn #
0C3A..0C3D ; Cn # [4]
0C45 ; Cn #
0C49 ; Cn #
0C4E..0C54 ; Cn # [7]
0C57..0C5F ; Cn # [9]
0C62..0C65 ; Cn # [4]
0C70..0C81 ; Cn # [18]
0C84 ; Cn #
0C8D ; Cn #
0C91 ; Cn #
0CA9 ; Cn #
0CB4 ; Cn #
0CBA..0CBD ; Cn # [4]
0CC5 ; Cn #
0CC9 ; Cn #
0CCE..0CD4 ; Cn # [7]
0CD7..0CDD ; Cn # [7]
0CDF ; Cn #
0CE2..0CE5 ; Cn # [4]
0CF0..0D01 ; Cn # [18]
0D04 ; Cn #
0D0D ; Cn #
0D11 ; Cn #
0D29 ; Cn #
0D3A..0D3D ; Cn # [4]
0D44..0D45 ; Cn # [2]
0D49 ; Cn #
0D4E..0D56 ; Cn # [9]
0D58..0D5F ; Cn # [8]
0D62..0D65 ; Cn # [4]
0D70..0D81 ; Cn # [18]
0D84 ; Cn #
0D97..0D99 ; Cn # [3]
0DB2 ; Cn #
0DBC ; Cn #
0DBE..0DBF ; Cn # [2]
0DC7..0DC9 ; Cn # [3]
0DCB..0DCE ; Cn # [4]
0DD5 ; Cn #
0DD7 ; Cn #
0DE0..0DF1 ; Cn # [18]

0DF5..0E00 ; Cn # [12]
0E3B..0E3E ; Cn # [4]
0E5C..0E80 ; Cn # [37]
0E83 ; Cn #
0E85..0E86 ; Cn # [2]
0E89 ; Cn #
0E8B..0E8C ; Cn # [2]
0E8E..0E93 ; Cn # [6]
0E98 ; Cn #
0EA0 ; Cn #
0EA4 ; Cn #
0EA6 ; Cn #
0EA8..0EA9 ; Cn # [2]
0EAC ; Cn #
0EBA ; Cn #
0EBE..0EBF ; Cn # [2]
0EC5 ; Cn #
0EC7 ; Cn #
0ECE..0ECF ; Cn # [2]
0EDA..0EDB ; Cn # [2]
0EDE..0EFF ; Cn # [34]
0F48 ; Cn #
0F6B..0F70 ; Cn # [6]
0F8C..0F8F ; Cn # [4]
0F98 ; Cn #
0FBD ; Cn #
0FCD..0FCE ; Cn # [2]
0FD0..0FFF ; Cn # [48]
1022 ; Cn #
1028 ; Cn #
102B ; Cn #
1033..1035 ; Cn # [3]
103A..103F ; Cn # [6]
105A..109F ; Cn # [70]
10C6..10CF ; Cn # [10]
10F7..10FA ; Cn # [4]
10FC..10FF ; Cn # [4]
115A..115E ; Cn # [5]
11A3..11A7 ; Cn # [5]
11FA..11FF ; Cn # [6]
1207 ; Cn #
1247 ; Cn #
1249 ; Cn #
124E..124F ; Cn # [2]
1257 ; Cn #
1259 ; Cn #

125E..125F ; Cn # [2]
1287 ; Cn #
1289 ; Cn #
128E..128F ; Cn # [2]
12AF ; Cn #
12B1 ; Cn #
12B6..12B7 ; Cn # [2]
12BF ; Cn #
12C1 ; Cn #
12C6..12C7 ; Cn # [2]
12CF ; Cn #
12D7 ; Cn #
12EF ; Cn #
130F ; Cn #
1311 ; Cn #
1316..1317 ; Cn # [2]
131F ; Cn #
1347 ; Cn #
135B..1360 ; Cn # [6]
137D..139F ; Cn # [35]
13F5..1400 ; Cn # [12]
1677..167F ; Cn # [9]
169D..169F ; Cn # [3]
16F1..177F ; Cn # [143]
17DD..17DF ; Cn # [3]
17EA..17FF ; Cn # [22]
180F ; Cn #
181A..181F ; Cn # [6]
1878..187F ; Cn # [8]
18AA..1DFF ; Cn # [1366]
1E9C..1E9F ; Cn # [4]
1EFA..1EFF ; Cn # [6]
1F16..1F17 ; Cn # [2]
1F1E..1F1F ; Cn # [2]
1F46..1F47 ; Cn # [2]
1F4E..1F4F ; Cn # [2]
1F58 ; Cn #
1F5A ; Cn #
1F5C ; Cn #
1F5E ; Cn #
1F7E..1F7F ; Cn # [2]
1FB5 ; Cn #
1FC5 ; Cn #
1FD4..1FD5 ; Cn # [2]
1FDC ; Cn #
1FF0..1FF1 ; Cn # [2]

1FF5 ; Cn #
1FFF ; Cn #
2047 ; Cn #
204E..2069 ; Cn # [28]
2071..2073 ; Cn # [3]
208F..209F ; Cn # [17]
20B0..20CF ; Cn # [32]
20E4..20FF ; Cn # [28]
213B..2152 ; Cn # [24]
2184..218F ; Cn # [12]
21F4..21FF ; Cn # [12]
22F2..22FF ; Cn # [14]
237C ; Cn #
239B..23FF ; Cn # [101]
2427..243F ; Cn # [25]
244B..245F ; Cn # [21]
24EB..24FF ; Cn # [21]
2596..259F ; Cn # [10]
25F8..25FF ; Cn # [8]
2614..2618 ; Cn # [5]
2672..2700 ; Cn # [143]
2705 ; Cn #
270A..270B ; Cn # [2]
2728 ; Cn #
274C ; Cn #
274E ; Cn #
2753..2755 ; Cn # [3]
2757 ; Cn #
275F..2760 ; Cn # [2]
2768..2775 ; Cn # [14]
2795..2797 ; Cn # [3]
27B0 ; Cn #
27BF..27FF ; Cn # [65]
2900..2E7F ; Cn # [1408]
2E9A ; Cn #
2EF4..2EFF ; Cn # [12]
2FD6..2FEF ; Cn # [26]
2FFC..2FFF ; Cn # [4]
303B..303D ; Cn # [3]
3040 ; Cn #
3095..3098 ; Cn # [4]
309F..30A0 ; Cn # [2]
30FF..3104 ; Cn # [6]
312D..3130 ; Cn # [4]
318F ; Cn #
31B8..31FF ; Cn # [72]

321D..321F ; Cn # [3]
3244..325F ; Cn # [28]
327C..327E ; Cn # [3]
32B1..32BF ; Cn # [15]
32CC..32CF ; Cn # [4]
32FF ; Cn #
3377..337A ; Cn # [4]
33DE..33DF ; Cn # [2]
33FF ; Cn #
4DB6..4DFF ; Cn # [74]
9FA6..9FFF ; Cn # [90]
A48D..A48F ; Cn # [3]
A4A2..A4A3 ; Cn # [2]
A4B4 ; Cn #
A4C1 ; Cn #
A4C5 ; Cn #
A4C7..ABFF ; Cn # [1849]
D7A4..D7FF ; Cn # [92]
FA2E..FAFF ; Cn # [210]
FB07..FB12 ; Cn # [12]
FB18..FB1C ; Cn # [5]
FB37 ; Cn #
FB3D ; Cn #
FB3F ; Cn #
FB42 ; Cn #
FB45 ; Cn #
FBB2..FBD2 ; Cn # [33]
FD40..FD4F ; Cn # [16]
FD90..FD91 ; Cn # [2]
FDC8..FDEF ; Cn # [40]
FDFC..FE1F ; Cn # [36]
FE24..FE2F ; Cn # [12]
FE45..FE48 ; Cn # [4]
FE53 ; Cn #
FE67 ; Cn #
FE6C..FE6F ; Cn # [4]
FE73 ; Cn #
FE75 ; Cn #
FEFD..FEFE ; Cn # [2]
FF00 ; Cn #
FF5F..FF60 ; Cn # [2]
FFBF..FFC1 ; Cn # [3]
FFC8..FFC9 ; Cn # [2]
FFD0..FFD1 ; Cn # [2]
FFD8..FFD9 ; Cn # [2]
FFDD..FFDF ; Cn # [3]

FFE7 ; Cn #
FFEF..FFF8 ; Cn # [10]
FFFE..102FF ; Cn # [770]
1031F ; Cn #
10324..1032F ; Cn # [12]
1034B..103FF ; Cn # [181]
10426..10427 ; Cn # [2]
1044E..1CFFF ; Cn # [52146]
1D0F6..1D0FF ; Cn # [10]
1D127..1D129 ; Cn # [3]
1D1DE..1D3FF ; Cn # [546]
1D455 ; Cn #
1D49D ; Cn #
1D4A0..1D4A1 ; Cn # [2]
1D4A3..1D4A4 ; Cn # [2]
1D4A7..1D4A8 ; Cn # [2]
1D4AD ; Cn #
1D4BA ; Cn #
1D4BC ; Cn #
1D4C1 ; Cn #
1D4C4 ; Cn #
1D506 ; Cn #
1D50B..1D50C ; Cn # [2]
1D515 ; Cn #
1D51D ; Cn #
1D53A ; Cn #
1D53F ; Cn #
1D545 ; Cn #
1D547..1D549 ; Cn # [3]
1D551 ; Cn #
1D6A4..1D6A7 ; Cn # [4]
1D7CA..1D7CD ; Cn # [4]
1D800..1FFFF ; Cn # [10240]
2A6D7..2F7FF ; Cn # [20777]
2FA1E..E0000 ; Cn # [722403]
E0002..E001F ; Cn # [30]
E0080..EFFFF ; Cn # [65408]
FFFFE..FFFFF ; Cn # [2]
10FFFE..10FFFF; Cn # [2]

Total code points: 880391

==

0041..005A ; Lu # [26] LATIN CAPITAL LETTER A..LATIN CAPITAL LETTER Z
00C0..00D6 ; Lu # [23] LATIN CAPITAL LETTER A WITH GRAVE..LATIN CAPITAL

LETTER O WITH DIAERESIS
00D8..00DE ; Lu # [7] LATIN CAPITAL LETTER O WITH STROKE..LATIN CAPITAL LETTER THORN
0100 ; Lu # LATIN CAPITAL LETTER A WITH MACRON
0102 ; Lu # LATIN CAPITAL LETTER A WITH BREVE
0104 ; Lu # LATIN CAPITAL LETTER A WITH OGONEK
0106 ; Lu # LATIN CAPITAL LETTER C WITH ACUTE
0108 ; Lu # LATIN CAPITAL LETTER C WITH CIRCUMFLEX
010A ; Lu # LATIN CAPITAL LETTER C WITH DOT ABOVE
010C ; Lu # LATIN CAPITAL LETTER C WITH CARON
010E ; Lu # LATIN CAPITAL LETTER D WITH CARON
0110 ; Lu # LATIN CAPITAL LETTER D WITH STROKE
0112 ; Lu # LATIN CAPITAL LETTER E WITH MACRON
0114 ; Lu # LATIN CAPITAL LETTER E WITH BREVE
0116 ; Lu # LATIN CAPITAL LETTER E WITH DOT ABOVE
0118 ; Lu # LATIN CAPITAL LETTER E WITH OGONEK
011A ; Lu # LATIN CAPITAL LETTER E WITH CARON
011C ; Lu # LATIN CAPITAL LETTER G WITH CIRCUMFLEX
011E ; Lu # LATIN CAPITAL LETTER G WITH BREVE
0120 ; Lu # LATIN CAPITAL LETTER G WITH DOT ABOVE
0122 ; Lu # LATIN CAPITAL LETTER G WITH CEDILLA
0124 ; Lu # LATIN CAPITAL LETTER H WITH CIRCUMFLEX
0126 ; Lu # LATIN CAPITAL LETTER H WITH STROKE
0128 ; Lu # LATIN CAPITAL LETTER I WITH TILDE
012A ; Lu # LATIN CAPITAL LETTER I WITH MACRON
012C ; Lu # LATIN CAPITAL LETTER I WITH BREVE
012E ; Lu # LATIN CAPITAL LETTER I WITH OGONEK
0130 ; Lu # LATIN CAPITAL LETTER I WITH DOT ABOVE
0132 ; Lu # LATIN CAPITAL LIGATURE IJ
0134 ; Lu # LATIN CAPITAL LETTER J WITH CIRCUMFLEX
0136 ; Lu # LATIN CAPITAL LETTER K WITH CEDILLA
0139 ; Lu # LATIN CAPITAL LETTER L WITH ACUTE
013B ; Lu # LATIN CAPITAL LETTER L WITH CEDILLA
013D ; Lu # LATIN CAPITAL LETTER L WITH CARON
013F ; Lu # LATIN CAPITAL LETTER L WITH MIDDLE DOT
0141 ; Lu # LATIN CAPITAL LETTER L WITH STROKE
0143 ; Lu # LATIN CAPITAL LETTER N WITH ACUTE
0145 ; Lu # LATIN CAPITAL LETTER N WITH CEDILLA
0147 ; Lu # LATIN CAPITAL LETTER N WITH CARON
014A ; Lu # LATIN CAPITAL LETTER ENG
014C ; Lu # LATIN CAPITAL LETTER O WITH MACRON
014E ; Lu # LATIN CAPITAL LETTER O WITH BREVE
0150 ; Lu # LATIN CAPITAL LETTER O WITH DOUBLE ACUTE
0152 ; Lu # LATIN CAPITAL LIGATURE OE
0154 ; Lu # LATIN CAPITAL LETTER R WITH ACUTE
0156 ; Lu # LATIN CAPITAL LETTER R WITH CEDILLA

0158 ; Lu # LATIN CAPITAL LETTER R WITH CARON
015A ; Lu # LATIN CAPITAL LETTER S WITH ACUTE
015C ; Lu # LATIN CAPITAL LETTER S WITH CIRCUMFLEX
015E ; Lu # LATIN CAPITAL LETTER S WITH CEDILLA
0160 ; Lu # LATIN CAPITAL LETTER S WITH CARON
0162 ; Lu # LATIN CAPITAL LETTER T WITH CEDILLA
0164 ; Lu # LATIN CAPITAL LETTER T WITH CARON
0166 ; Lu # LATIN CAPITAL LETTER T WITH STROKE
0168 ; Lu # LATIN CAPITAL LETTER U WITH TILDE
016A ; Lu # LATIN CAPITAL LETTER U WITH MACRON
016C ; Lu # LATIN CAPITAL LETTER U WITH BREVE
016E ; Lu # LATIN CAPITAL LETTER U WITH RING ABOVE
0170 ; Lu # LATIN CAPITAL LETTER U WITH DOUBLE ACUTE
0172 ; Lu # LATIN CAPITAL LETTER U WITH OGONEK
0174 ; Lu # LATIN CAPITAL LETTER W WITH CIRCUMFLEX
0176 ; Lu # LATIN CAPITAL LETTER Y WITH CIRCUMFLEX
0178..0179 ; Lu # [2] LATIN CAPITAL LETTER Y WITH DIAERESIS..LATIN CAPITAL LETTER Z WITH ACUTE
017B ; Lu # LATIN CAPITAL LETTER Z WITH DOT ABOVE
017D ; Lu # LATIN CAPITAL LETTER Z WITH CARON
0181..0182 ; Lu # [2] LATIN CAPITAL LETTER B WITH HOOK..LATIN CAPITAL LETTER B WITH TOPBAR
0184 ; Lu # LATIN CAPITAL LETTER TONE SIX
0186..0187 ; Lu # [2] LATIN CAPITAL LETTER OPEN O..LATIN CAPITAL LETTER C WITH HOOK
0189..018B ; Lu # [3] LATIN CAPITAL LETTER AFRICAN D..LATIN CAPITAL LETTER D WITH TOPBAR
018E..0191 ; Lu # [4] LATIN CAPITAL LETTER REVERSED E..LATIN CAPITAL LETTER F WITH HOOK
0193..0194 ; Lu # [2] LATIN CAPITAL LETTER G WITH HOOK..LATIN CAPITAL LETTER GAMMA
0196..0198 ; Lu # [3] LATIN CAPITAL LETTER IOTA..LATIN CAPITAL LETTER K WITH HOOK
019C..019D ; Lu # [2] LATIN CAPITAL LETTER TURNED M..LATIN CAPITAL LETTER N WITH LEFT HOOK
019F..01A0 ; Lu # [2] LATIN CAPITAL LETTER O WITH MIDDLE TILDE..LATIN CAPITAL LETTER O WITH HORN
01A2 ; Lu # LATIN CAPITAL LETTER OI
01A4 ; Lu # LATIN CAPITAL LETTER P WITH HOOK
01A6..01A7 ; Lu # [2] LATIN LETTER YR..LATIN CAPITAL LETTER TONE TWO
01A9 ; Lu # LATIN CAPITAL LETTER ESH
01AC ; Lu # LATIN CAPITAL LETTER T WITH HOOK
01AE..01AF ; Lu # [2] LATIN CAPITAL LETTER T WITH RETROFLEX HOOK..LATIN CAPITAL LETTER U WITH HORN
01B1..01B3 ; Lu # [3] LATIN CAPITAL LETTER UPSILON..LATIN CAPITAL LETTER Y WITH HOOK

01B5 ; Lu # LATIN CAPITAL LETTER Z WITH STROKE
01B7..01B8 ; Lu # [2] LATIN CAPITAL LETTER EZH..LATIN CAPITAL LETTER EZH REVERSED
01BC ; Lu # LATIN CAPITAL LETTER TONE FIVE
01C4 ; Lu # LATIN CAPITAL LETTER DZ WITH CARON
01C7 ; Lu # LATIN CAPITAL LETTER LJ
01CA ; Lu # LATIN CAPITAL LETTER NJ
01CD ; Lu # LATIN CAPITAL LETTER A WITH CARON
01CF ; Lu # LATIN CAPITAL LETTER I WITH CARON
01D1 ; Lu # LATIN CAPITAL LETTER O WITH CARON
01D3 ; Lu # LATIN CAPITAL LETTER U WITH CARON
01D5 ; Lu # LATIN CAPITAL LETTER U WITH DIAERESIS AND MACRON
01D7 ; Lu # LATIN CAPITAL LETTER U WITH DIAERESIS AND ACUTE
01D9 ; Lu # LATIN CAPITAL LETTER U WITH DIAERESIS AND CARON
01DB ; Lu # LATIN CAPITAL LETTER U WITH DIAERESIS AND GRAVE
01DE ; Lu # LATIN CAPITAL LETTER A WITH DIAERESIS AND MACRON
01E0 ; Lu # LATIN CAPITAL LETTER A WITH DOT ABOVE AND MACRON
01E2 ; Lu # LATIN CAPITAL LETTER AE WITH MACRON
01E4 ; Lu # LATIN CAPITAL LETTER G WITH STROKE
01E6 ; Lu # LATIN CAPITAL LETTER G WITH CARON
01E8 ; Lu # LATIN CAPITAL LETTER K WITH CARON
01EA ; Lu # LATIN CAPITAL LETTER O WITH OGONEK
01EC ; Lu # LATIN CAPITAL LETTER O WITH OGONEK AND MACRON
01EE ; Lu # LATIN CAPITAL LETTER EZH WITH CARON
01F1 ; Lu # LATIN CAPITAL LETTER DZ
01F4 ; Lu # LATIN CAPITAL LETTER G WITH ACUTE
01F6..01F8 ; Lu # [3] LATIN CAPITAL LETTER HWAIR..LATIN CAPITAL LETTER N WITH GRAVE
01FA ; Lu # LATIN CAPITAL LETTER A WITH RING ABOVE AND ACUTE
01FC ; Lu # LATIN CAPITAL LETTER AE WITH ACUTE
01FE ; Lu # LATIN CAPITAL LETTER O WITH STROKE AND ACUTE
0200 ; Lu # LATIN CAPITAL LETTER A WITH DOUBLE GRAVE
0202 ; Lu # LATIN CAPITAL LETTER A WITH INVERTED BREVE
0204 ; Lu # LATIN CAPITAL LETTER E WITH DOUBLE GRAVE
0206 ; Lu # LATIN CAPITAL LETTER E WITH INVERTED BREVE
0208 ; Lu # LATIN CAPITAL LETTER I WITH DOUBLE GRAVE
020A ; Lu # LATIN CAPITAL LETTER I WITH INVERTED BREVE
020C ; Lu # LATIN CAPITAL LETTER O WITH DOUBLE GRAVE
020E ; Lu # LATIN CAPITAL LETTER O WITH INVERTED BREVE
0210 ; Lu # LATIN CAPITAL LETTER R WITH DOUBLE GRAVE
0212 ; Lu # LATIN CAPITAL LETTER R WITH INVERTED BREVE
0214 ; Lu # LATIN CAPITAL LETTER U WITH DOUBLE GRAVE
0216 ; Lu # LATIN CAPITAL LETTER U WITH INVERTED BREVE
0218 ; Lu # LATIN CAPITAL LETTER S WITH COMMA BELOW
021A ; Lu # LATIN CAPITAL LETTER T WITH COMMA BELOW
021C ; Lu # LATIN CAPITAL LETTER YOGH

021E ; Lu # LATIN CAPITAL LETTER H WITH CARON
0222 ; Lu # LATIN CAPITAL LETTER OU
0224 ; Lu # LATIN CAPITAL LETTER Z WITH HOOK
0226 ; Lu # LATIN CAPITAL LETTER A WITH DOT ABOVE
0228 ; Lu # LATIN CAPITAL LETTER E WITH CEDILLA
022A ; Lu # LATIN CAPITAL LETTER O WITH DIAERESIS AND MACRON
022C ; Lu # LATIN CAPITAL LETTER O WITH TILDE AND MACRON
022E ; Lu # LATIN CAPITAL LETTER O WITH DOT ABOVE
0230 ; Lu # LATIN CAPITAL LETTER O WITH DOT ABOVE AND MACRON
0232 ; Lu # LATIN CAPITAL LETTER Y WITH MACRON
0386 ; Lu # GREEK CAPITAL LETTER ALPHA WITH TONOS
0388..038A ; Lu # [3] GREEK CAPITAL LETTER EPSILON WITH TONOS..GREEK CAPITAL LETTER IOTA WITH TONOS
038C ; Lu # GREEK CAPITAL LETTER OMICRON WITH TONOS
038E..038F ; Lu # [2] GREEK CAPITAL LETTER UPSILON WITH TONOS..GREEK CAPITAL LETTER OMEGA WITH TONOS
0391..03A1 ; Lu # [17] GREEK CAPITAL LETTER ALPHA..GREEK CAPITAL LETTER RHO
03A3..03AB ; Lu # [9] GREEK CAPITAL LETTER SIGMA..GREEK CAPITAL LETTER UPSILON WITH DIALYTIKA
03D2..03D4 ; Lu # [3] GREEK UPSILON WITH HOOK SYMBOL..GREEK UPSILON WITH DIAERESIS AND HOOK SYMBOL
03DA ; Lu # GREEK LETTER STIGMA
03DC ; Lu # GREEK LETTER DIGAMMA
03DE ; Lu # GREEK LETTER KOPPA
03E0 ; Lu # GREEK LETTER SAMPI
03E2 ; Lu # COPTIC CAPITAL LETTER SHEI
03E4 ; Lu # COPTIC CAPITAL LETTER FEI
03E6 ; Lu # COPTIC CAPITAL LETTER KHEI
03E8 ; Lu # COPTIC CAPITAL LETTER HORI
03EA ; Lu # COPTIC CAPITAL LETTER GANGIA
03EC ; Lu # COPTIC CAPITAL LETTER SHIMA
03EE ; Lu # COPTIC CAPITAL LETTER DEI
03F4 ; Lu # GREEK CAPITAL THETA SYMBOL
0400..042F ; Lu # [48] CYRILLIC CAPITAL LETTER IE WITH GRAVE..CYRILLIC CAPITAL LETTER YA
0460 ; Lu # CYRILLIC CAPITAL LETTER OMEGA
0462 ; Lu # CYRILLIC CAPITAL LETTER YAT
0464 ; Lu # CYRILLIC CAPITAL LETTER IOTIFIED E
0466 ; Lu # CYRILLIC CAPITAL LETTER LITTLE YUS
0468 ; Lu # CYRILLIC CAPITAL LETTER IOTIFIED LITTLE YUS
046A ; Lu # CYRILLIC CAPITAL LETTER BIG YUS
046C ; Lu # CYRILLIC CAPITAL LETTER IOTIFIED BIG YUS
046E ; Lu # CYRILLIC CAPITAL LETTER KSI
0470 ; Lu # CYRILLIC CAPITAL LETTER PSI
0472 ; Lu # CYRILLIC CAPITAL LETTER FITA
0474 ; Lu # CYRILLIC CAPITAL LETTER IZHITSA

0476 ; Lu # CYRILLIC CAPITAL LETTER IZHITSA WITH DOUBLE GRAVE ACCENT
0478 ; Lu # CYRILLIC CAPITAL LETTER UK
047A ; Lu # CYRILLIC CAPITAL LETTER ROUND OMEGA
047C ; Lu # CYRILLIC CAPITAL LETTER OMEGA WITH TITLO
047E ; Lu # CYRILLIC CAPITAL LETTER OT
0480 ; Lu # CYRILLIC CAPITAL LETTER KOPPA
048C ; Lu # CYRILLIC CAPITAL LETTER SEMISOFT SIGN
048E ; Lu # CYRILLIC CAPITAL LETTER ER WITH TICK
0490 ; Lu # CYRILLIC CAPITAL LETTER GHE WITH UPTURN
0492 ; Lu # CYRILLIC CAPITAL LETTER GHE WITH STROKE
0494 ; Lu # CYRILLIC CAPITAL LETTER GHE WITH MIDDLE HOOK
0496 ; Lu # CYRILLIC CAPITAL LETTER ZHE WITH DESCENDER
0498 ; Lu # CYRILLIC CAPITAL LETTER ZE WITH DESCENDER
049A ; Lu # CYRILLIC CAPITAL LETTER KA WITH DESCENDER
049C ; Lu # CYRILLIC CAPITAL LETTER KA WITH VERTICAL STROKE
049E ; Lu # CYRILLIC CAPITAL LETTER KA WITH STROKE
04A0 ; Lu # CYRILLIC CAPITAL LETTER BASHKIR KA
04A2 ; Lu # CYRILLIC CAPITAL LETTER EN WITH DESCENDER
04A4 ; Lu # CYRILLIC CAPITAL LIGATURE EN GHE
04A6 ; Lu # CYRILLIC CAPITAL LETTER PE WITH MIDDLE HOOK
04A8 ; Lu # CYRILLIC CAPITAL LETTER ABKHASIAN HA
04AA ; Lu # CYRILLIC CAPITAL LETTER ES WITH DESCENDER
04AC ; Lu # CYRILLIC CAPITAL LETTER TE WITH DESCENDER
04AE ; Lu # CYRILLIC CAPITAL LETTER STRAIGHT U
04B0 ; Lu # CYRILLIC CAPITAL LETTER STRAIGHT U WITH STROKE
04B2 ; Lu # CYRILLIC CAPITAL LETTER HA WITH DESCENDER
04B4 ; Lu # CYRILLIC CAPITAL LIGATURE TE TSE
04B6 ; Lu # CYRILLIC CAPITAL LETTER CHE WITH DESCENDER
04B8 ; Lu # CYRILLIC CAPITAL LETTER CHE WITH VERTICAL STROKE
04BA ; Lu # CYRILLIC CAPITAL LETTER SHHA
04BC ; Lu # CYRILLIC CAPITAL LETTER ABKHASIAN CHE
04BE ; Lu # CYRILLIC CAPITAL LETTER ABKHASIAN CHE WITH DESCENDER
04C0..04C1 ; Lu # [2] CYRILLIC LETTER PALOCHKA..CYRILLIC CAPITAL LETTER ZHE WITH BREVE
04C3 ; Lu # CYRILLIC CAPITAL LETTER KA WITH HOOK
04C7 ; Lu # CYRILLIC CAPITAL LETTER EN WITH HOOK
04CB ; Lu # CYRILLIC CAPITAL LETTER KHAKASSIAN CHE
04D0 ; Lu # CYRILLIC CAPITAL LETTER A WITH BREVE
04D2 ; Lu # CYRILLIC CAPITAL LETTER A WITH DIAERESIS
04D4 ; Lu # CYRILLIC CAPITAL LIGATURE A IE
04D6 ; Lu # CYRILLIC CAPITAL LETTER IE WITH BREVE
04D8 ; Lu # CYRILLIC CAPITAL LETTER SCHWA
04DA ; Lu # CYRILLIC CAPITAL LETTER SCHWA WITH DIAERESIS
04DC ; Lu # CYRILLIC CAPITAL LETTER ZHE WITH DIAERESIS
04DE ; Lu # CYRILLIC CAPITAL LETTER ZE WITH DIAERESIS
04E0 ; Lu # CYRILLIC CAPITAL LETTER ABKHASIAN DZE

04E2 ; Lu # CYRILLIC CAPITAL LETTER I WITH MACRON
04E4 ; Lu # CYRILLIC CAPITAL LETTER I WITH DIAERESIS
04E6 ; Lu # CYRILLIC CAPITAL LETTER O WITH DIAERESIS
04E8 ; Lu # CYRILLIC CAPITAL LETTER BARRED O
04EA ; Lu # CYRILLIC CAPITAL LETTER BARRED O WITH DIAERESIS
04EC ; Lu # CYRILLIC CAPITAL LETTER E WITH DIAERESIS
04EE ; Lu # CYRILLIC CAPITAL LETTER U WITH MACRON
04F0 ; Lu # CYRILLIC CAPITAL LETTER U WITH DIAERESIS
04F2 ; Lu # CYRILLIC CAPITAL LETTER U WITH DOUBLE ACUTE
04F4 ; Lu # CYRILLIC CAPITAL LETTER CHE WITH DIAERESIS
04F8 ; Lu # CYRILLIC CAPITAL LETTER YERU WITH DIAERESIS
0531..0556 ; Lu # [38] ARMENIAN CAPITAL LETTER AYB..ARMENIAN CAPITAL LETTER FEH
10A0..10C5 ; Lu # [38] GEORGIAN CAPITAL LETTER AN..GEORGIAN CAPITAL LETTER HOE
1E00 ; Lu # LATIN CAPITAL LETTER A WITH RING BELOW
1E02 ; Lu # LATIN CAPITAL LETTER B WITH DOT ABOVE
1E04 ; Lu # LATIN CAPITAL LETTER B WITH DOT BELOW
1E06 ; Lu # LATIN CAPITAL LETTER B WITH LINE BELOW
1E08 ; Lu # LATIN CAPITAL LETTER C WITH CEDILLA AND ACUTE
1E0A ; Lu # LATIN CAPITAL LETTER D WITH DOT ABOVE
1E0C ; Lu # LATIN CAPITAL LETTER D WITH DOT BELOW
1E0E ; Lu # LATIN CAPITAL LETTER D WITH LINE BELOW
1E10 ; Lu # LATIN CAPITAL LETTER D WITH CEDILLA
1E12 ; Lu # LATIN CAPITAL LETTER D WITH CIRCUMFLEX BELOW
1E14 ; Lu # LATIN CAPITAL LETTER E WITH MACRON AND GRAVE
1E16 ; Lu # LATIN CAPITAL LETTER E WITH MACRON AND ACUTE
1E18 ; Lu # LATIN CAPITAL LETTER E WITH CIRCUMFLEX BELOW
1E1A ; Lu # LATIN CAPITAL LETTER E WITH TILDE BELOW
1E1C ; Lu # LATIN CAPITAL LETTER E WITH CEDILLA AND BREVE
1E1E ; Lu # LATIN CAPITAL LETTER F WITH DOT ABOVE
1E20 ; Lu # LATIN CAPITAL LETTER G WITH MACRON
1E22 ; Lu # LATIN CAPITAL LETTER H WITH DOT ABOVE
1E24 ; Lu # LATIN CAPITAL LETTER H WITH DOT BELOW
1E26 ; Lu # LATIN CAPITAL LETTER H WITH DIAERESIS
1E28 ; Lu # LATIN CAPITAL LETTER H WITH CEDILLA
1E2A ; Lu # LATIN CAPITAL LETTER H WITH BREVE BELOW
1E2C ; Lu # LATIN CAPITAL LETTER I WITH TILDE BELOW
1E2E ; Lu # LATIN CAPITAL LETTER I WITH DIAERESIS AND ACUTE
1E30 ; Lu # LATIN CAPITAL LETTER K WITH ACUTE
1E32 ; Lu # LATIN CAPITAL LETTER K WITH DOT BELOW
1E34 ; Lu # LATIN CAPITAL LETTER K WITH LINE BELOW
1E36 ; Lu # LATIN CAPITAL LETTER L WITH DOT BELOW
1E38 ; Lu # LATIN CAPITAL LETTER L WITH DOT BELOW AND MACRON
1E3A ; Lu # LATIN CAPITAL LETTER L WITH LINE BELOW
1E3C ; Lu # LATIN CAPITAL LETTER L WITH CIRCUMFLEX BELOW

1E3E ; Lu # LATIN CAPITAL LETTER M WITH ACUTE
1E40 ; Lu # LATIN CAPITAL LETTER M WITH DOT ABOVE
1E42 ; Lu # LATIN CAPITAL LETTER M WITH DOT BELOW
1E44 ; Lu # LATIN CAPITAL LETTER N WITH DOT ABOVE
1E46 ; Lu # LATIN CAPITAL LETTER N WITH DOT BELOW
1E48 ; Lu # LATIN CAPITAL LETTER N WITH LINE BELOW
1E4A ; Lu # LATIN CAPITAL LETTER N WITH CIRCUMFLEX BELOW
1E4C ; Lu # LATIN CAPITAL LETTER O WITH TILDE AND ACUTE
1E4E ; Lu # LATIN CAPITAL LETTER O WITH TILDE AND DIAERESIS
1E50 ; Lu # LATIN CAPITAL LETTER O WITH MACRON AND GRAVE
1E52 ; Lu # LATIN CAPITAL LETTER O WITH MACRON AND ACUTE
1E54 ; Lu # LATIN CAPITAL LETTER P WITH ACUTE
1E56 ; Lu # LATIN CAPITAL LETTER P WITH DOT ABOVE
1E58 ; Lu # LATIN CAPITAL LETTER R WITH DOT ABOVE
1E5A ; Lu # LATIN CAPITAL LETTER R WITH DOT BELOW
1E5C ; Lu # LATIN CAPITAL LETTER R WITH DOT BELOW AND MACRON
1E5E ; Lu # LATIN CAPITAL LETTER R WITH LINE BELOW
1E60 ; Lu # LATIN CAPITAL LETTER S WITH DOT ABOVE
1E62 ; Lu # LATIN CAPITAL LETTER S WITH DOT BELOW
1E64 ; Lu # LATIN CAPITAL LETTER S WITH ACUTE AND DOT ABOVE
1E66 ; Lu # LATIN CAPITAL LETTER S WITH CARON AND DOT ABOVE
1E68 ; Lu # LATIN CAPITAL LETTER S WITH DOT BELOW AND DOT ABOVE
1E6A ; Lu # LATIN CAPITAL LETTER T WITH DOT ABOVE
1E6C ; Lu # LATIN CAPITAL LETTER T WITH DOT BELOW
1E6E ; Lu # LATIN CAPITAL LETTER T WITH LINE BELOW
1E70 ; Lu # LATIN CAPITAL LETTER T WITH CIRCUMFLEX BELOW
1E72 ; Lu # LATIN CAPITAL LETTER U WITH DIAERESIS BELOW
1E74 ; Lu # LATIN CAPITAL LETTER U WITH TILDE BELOW
1E76 ; Lu # LATIN CAPITAL LETTER U WITH CIRCUMFLEX BELOW
1E78 ; Lu # LATIN CAPITAL LETTER U WITH TILDE AND ACUTE
1E7A ; Lu # LATIN CAPITAL LETTER U WITH MACRON AND DIAERESIS
1E7C ; Lu # LATIN CAPITAL LETTER V WITH TILDE
1E7E ; Lu # LATIN CAPITAL LETTER V WITH DOT BELOW
1E80 ; Lu # LATIN CAPITAL LETTER W WITH GRAVE
1E82 ; Lu # LATIN CAPITAL LETTER W WITH ACUTE
1E84 ; Lu # LATIN CAPITAL LETTER W WITH DIAERESIS
1E86 ; Lu # LATIN CAPITAL LETTER W WITH DOT ABOVE
1E88 ; Lu # LATIN CAPITAL LETTER W WITH DOT BELOW
1E8A ; Lu # LATIN CAPITAL LETTER X WITH DOT ABOVE
1E8C ; Lu # LATIN CAPITAL LETTER X WITH DIAERESIS
1E8E ; Lu # LATIN CAPITAL LETTER Y WITH DOT ABOVE
1E90 ; Lu # LATIN CAPITAL LETTER Z WITH CIRCUMFLEX
1E92 ; Lu # LATIN CAPITAL LETTER Z WITH DOT BELOW
1E94 ; Lu # LATIN CAPITAL LETTER Z WITH LINE BELOW
1EA0 ; Lu # LATIN CAPITAL LETTER A WITH DOT BELOW
1EA2 ; Lu # LATIN CAPITAL LETTER A WITH HOOK ABOVE

1EA4 ; Lu # LATIN CAPITAL LETTER A WITH CIRCUMFLEX AND ACUTE
1EA6 ; Lu # LATIN CAPITAL LETTER A WITH CIRCUMFLEX AND GRAVE
1EA8 ; Lu # LATIN CAPITAL LETTER A WITH CIRCUMFLEX AND HOOK ABOVE
1EAA ; Lu # LATIN CAPITAL LETTER A WITH CIRCUMFLEX AND TILDE
1EAC ; Lu # LATIN CAPITAL LETTER A WITH CIRCUMFLEX AND DOT BELOW
1EAE ; Lu # LATIN CAPITAL LETTER A WITH BREVE AND ACUTE
1EB0 ; Lu # LATIN CAPITAL LETTER A WITH BREVE AND GRAVE
1EB2 ; Lu # LATIN CAPITAL LETTER A WITH BREVE AND HOOK ABOVE
1EB4 ; Lu # LATIN CAPITAL LETTER A WITH BREVE AND TILDE
1EB6 ; Lu # LATIN CAPITAL LETTER A WITH BREVE AND DOT BELOW
1EB8 ; Lu # LATIN CAPITAL LETTER E WITH DOT BELOW
1EBA ; Lu # LATIN CAPITAL LETTER E WITH HOOK ABOVE
1EBC ; Lu # LATIN CAPITAL LETTER E WITH TILDE
1EBE ; Lu # LATIN CAPITAL LETTER E WITH CIRCUMFLEX AND ACUTE
1EC0 ; Lu # LATIN CAPITAL LETTER E WITH CIRCUMFLEX AND GRAVE
1EC2 ; Lu # LATIN CAPITAL LETTER E WITH CIRCUMFLEX AND HOOK ABOVE
1EC4 ; Lu # LATIN CAPITAL LETTER E WITH CIRCUMFLEX AND TILDE
1EC6 ; Lu # LATIN CAPITAL LETTER E WITH CIRCUMFLEX AND DOT BELOW
1EC8 ; Lu # LATIN CAPITAL LETTER I WITH HOOK ABOVE
1ECA ; Lu # LATIN CAPITAL LETTER I WITH DOT BELOW
1ECC ; Lu # LATIN CAPITAL LETTER O WITH DOT BELOW
1ECE ; Lu # LATIN CAPITAL LETTER O WITH HOOK ABOVE
1ED0 ; Lu # LATIN CAPITAL LETTER O WITH CIRCUMFLEX AND ACUTE
1ED2 ; Lu # LATIN CAPITAL LETTER O WITH CIRCUMFLEX AND GRAVE
1ED4 ; Lu # LATIN CAPITAL LETTER O WITH CIRCUMFLEX AND HOOK ABOVE
1ED6 ; Lu # LATIN CAPITAL LETTER O WITH CIRCUMFLEX AND TILDE
1ED8 ; Lu # LATIN CAPITAL LETTER O WITH CIRCUMFLEX AND DOT BELOW
1EDA ; Lu # LATIN CAPITAL LETTER O WITH HORN AND ACUTE
1EDC ; Lu # LATIN CAPITAL LETTER O WITH HORN AND GRAVE
1EDE ; Lu # LATIN CAPITAL LETTER O WITH HORN AND HOOK ABOVE
1EE0 ; Lu # LATIN CAPITAL LETTER O WITH HORN AND TILDE
1EE2 ; Lu # LATIN CAPITAL LETTER O WITH HORN AND DOT BELOW
1EE4 ; Lu # LATIN CAPITAL LETTER U WITH DOT BELOW
1EE6 ; Lu # LATIN CAPITAL LETTER U WITH HOOK ABOVE
1EE8 ; Lu # LATIN CAPITAL LETTER U WITH HORN AND ACUTE
1EEA ; Lu # LATIN CAPITAL LETTER U WITH HORN AND GRAVE
1EEC ; Lu # LATIN CAPITAL LETTER U WITH HORN AND HOOK ABOVE
1EEE ; Lu # LATIN CAPITAL LETTER U WITH HORN AND TILDE
1EF0 ; Lu # LATIN CAPITAL LETTER U WITH HORN AND DOT BELOW
1EF2 ; Lu # LATIN CAPITAL LETTER Y WITH GRAVE
1EF4 ; Lu # LATIN CAPITAL LETTER Y WITH DOT BELOW
1EF6 ; Lu # LATIN CAPITAL LETTER Y WITH HOOK ABOVE
1EF8 ; Lu # LATIN CAPITAL LETTER Y WITH TILDE
1F08..1F0F ; Lu # [8] GREEK CAPITAL LETTER ALPHA WITH PSILI..GREEK CAPITAL LETTER ALPHA WITH DASIA AND PERISPOMENI
1F18..1F1D ; Lu # [6] GREEK CAPITAL LETTER EPSILON WITH PSILI..GREEK CAPITAL

LETTER EPSILON WITH DASIA AND OXIA
1F28..1F2F ; Lu # [8] GREEK CAPITAL LETTER ETA WITH PSILI..GREEK CAPITAL LETTER ETA WITH DASIA AND PERISPOMENI
1F38..1F3F ; Lu # [8] GREEK CAPITAL LETTER IOTA WITH PSILI..GREEK CAPITAL LETTER IOTA WITH DASIA AND PERISPOMENI
1F48..1F4D ; Lu # [6] GREEK CAPITAL LETTER OMICRON WITH PSILI..GREEK CAPITAL LETTER OMICRON WITH DASIA AND OXIA
1F59 ; Lu # GREEK CAPITAL LETTER UPSILON WITH DASIA
1F5B ; Lu # GREEK CAPITAL LETTER UPSILON WITH DASIA AND VARIA
1F5D ; Lu # GREEK CAPITAL LETTER UPSILON WITH DASIA AND OXIA
1F5F ; Lu # GREEK CAPITAL LETTER UPSILON WITH DASIA AND PERISPOMENI
1F68..1F6F ; Lu # [8] GREEK CAPITAL LETTER OMEGA WITH PSILI..GREEK CAPITAL LETTER OMEGA WITH DASIA AND PERISPOMENI
1FB8..1FBB ; Lu # [4] GREEK CAPITAL LETTER ALPHA WITH VRACHY..GREEK CAPITAL LETTER ALPHA WITH OXIA
1FC8..1FCB ; Lu # [4] GREEK CAPITAL LETTER EPSILON WITH VARIA..GREEK CAPITAL LETTER ETA WITH OXIA
1FD8..1FDB ; Lu # [4] GREEK CAPITAL LETTER IOTA WITH VRACHY..GREEK CAPITAL LETTER IOTA WITH OXIA
1FE8..1FEC ; Lu # [5] GREEK CAPITAL LETTER UPSILON WITH VRACHY..GREEK CAPITAL LETTER RHO WITH DASIA
1FF8..1FFB ; Lu # [4] GREEK CAPITAL LETTER OMICRON WITH VARIA..GREEK CAPITAL LETTER OMEGA WITH OXIA
2102 ; Lu # DOUBLE-STRUCK CAPITAL C
2107 ; Lu # EULER CONSTANT
210B..210D ; Lu # [3] SCRIPT CAPITAL H..DOUBLE-STRUCK CAPITAL H
2110..2112 ; Lu # [3] SCRIPT CAPITAL I..SCRIPT CAPITAL L
2115 ; Lu # DOUBLE-STRUCK CAPITAL N
2119..211D ; Lu # [5] DOUBLE-STRUCK CAPITAL P..DOUBLE-STRUCK CAPITAL R
2124 ; Lu # DOUBLE-STRUCK CAPITAL Z
2126 ; Lu # OHM SIGN
2128 ; Lu # BLACK-LETTER CAPITAL Z
212A..212D ; Lu # [4] KELVIN SIGN..BLACK-LETTER CAPITAL C
2130..2131 ; Lu # [2] SCRIPT CAPITAL E..SCRIPT CAPITAL F
2133 ; Lu # SCRIPT CAPITAL M
FF21..FF3A ; Lu # [26] FULLWIDTH LATIN CAPITAL LETTER A..FULLWIDTH LATIN CAPITAL LETTER Z
10400..10425 ; Lu # [38] DESERET CAPITAL LETTER LONG I..DESERET CAPITAL LETTER ENG
1D400..1D419 ; Lu # [26] MATHEMATICAL BOLD CAPITAL A..MATHEMATICAL BOLD CAPITAL Z
1D434..1D44D ; Lu # [26] MATHEMATICAL ITALIC CAPITAL A..MATHEMATICAL ITALIC CAPITAL Z
1D468..1D481 ; Lu # [26] MATHEMATICAL BOLD ITALIC CAPITAL A..MATHEMATICAL BOLD ITALIC CAPITAL Z
1D49C ; Lu # MATHEMATICAL SCRIPT CAPITAL A

1D49E..1D49F ; Lu # [2] MATHEMATICAL SCRIPT CAPITAL C..MATHEMATICAL SCRIPT CAPITAL D
1D4A2 ; Lu # MATHEMATICAL SCRIPT CAPITAL G
1D4A5..1D4A6 ; Lu # [2] MATHEMATICAL SCRIPT CAPITAL J..MATHEMATICAL SCRIPT CAPITAL K
1D4A9..1D4AC ; Lu # [4] MATHEMATICAL SCRIPT CAPITAL N..MATHEMATICAL SCRIPT CAPITAL Q
1D4AE..1D4B5 ; Lu # [8] MATHEMATICAL SCRIPT CAPITAL S..MATHEMATICAL SCRIPT CAPITAL Z
1D4D0..1D4E9 ; Lu # [26] MATHEMATICAL BOLD SCRIPT CAPITAL A..MATHEMATICAL BOLD SCRIPT CAPITAL Z
1D504..1D505 ; Lu # [2] MATHEMATICAL FRAKTUR CAPITAL A..MATHEMATICAL FRAKTUR CAPITAL B
1D507..1D50A ; Lu # [4] MATHEMATICAL FRAKTUR CAPITAL D..MATHEMATICAL FRAKTUR CAPITAL G
1D50D..1D514 ; Lu # [8] MATHEMATICAL FRAKTUR CAPITAL J..MATHEMATICAL FRAKTUR CAPITAL Q
1D516..1D51C ; Lu # [7] MATHEMATICAL FRAKTUR CAPITAL S..MATHEMATICAL FRAKTUR CAPITAL Y
1D538..1D539 ; Lu # [2] MATHEMATICAL DOUBLE-STRUCK CAPITAL A..MATHEMATICAL DOUBLE-STRUCK CAPITAL B
1D53B..1D53E ; Lu # [4] MATHEMATICAL DOUBLE-STRUCK CAPITAL D..MATHEMATICAL DOUBLE-STRUCK CAPITAL G
1D540..1D544 ; Lu # [5] MATHEMATICAL DOUBLE-STRUCK CAPITAL I..MATHEMATICAL DOUBLE-STRUCK CAPITAL M
1D546 ; Lu # MATHEMATICAL DOUBLE-STRUCK CAPITAL O
1D54A..1D550 ; Lu # [7] MATHEMATICAL DOUBLE-STRUCK CAPITAL S..MATHEMATICAL DOUBLE-STRUCK CAPITAL Y
1D56C..1D585 ; Lu # [26] MATHEMATICAL BOLD FRAKTUR CAPITAL A..MATHEMATICAL BOLD FRAKTUR CAPITAL Z
1D5A0..1D5B9 ; Lu # [26] MATHEMATICAL SANS-SERIF CAPITAL A..MATHEMATICAL SANS-SERIF CAPITAL Z
1D5D4..1D5ED ; Lu # [26] MATHEMATICAL SANS-SERIF BOLD CAPITAL A..MATHEMATICAL SANS-SERIF BOLD CAPITAL Z
1D608..1D621 ; Lu # [26] MATHEMATICAL SANS-SERIF ITALIC CAPITAL A..MATHEMATICAL SANS-SERIF ITALIC CAPITAL Z
1D63C..1D655 ; Lu # [26] MATHEMATICAL SANS-SERIF BOLD ITALIC CAPITAL A..MATHEMATICAL SANS-SERIF BOLD ITALIC CAPITAL Z
1D670..1D689 ; Lu # [26] MATHEMATICAL MONOSPACE CAPITAL A..MATHEMATICAL MONOSPACE CAPITAL Z
1D6A8..1D6C0 ; Lu # [25] MATHEMATICAL BOLD CAPITAL ALPHA..MATHEMATICAL BOLD CAPITAL OMEGA
1D6E2..1D6FA ; Lu # [25] MATHEMATICAL ITALIC CAPITAL ALPHA..MATHEMATICAL ITALIC CAPITAL OMEGA
1D71C..1D734 ; Lu # [25] MATHEMATICAL BOLD ITALIC CAPITAL ALPHA..MATHEMATICAL BOLD ITALIC CAPITAL OMEGA

1D756..1D76E ; Lu # [25] MATHEMATICAL SANS-SERIF BOLD CAPITAL ALPHA..MATHEMATICAL SANS-SERIF BOLD CAPITAL OMEGA
1D790..1D7A8 ; Lu # [25] MATHEMATICAL SANS-SERIF BOLD ITALIC CAPITAL ALPHA..MATHEMATICAL SANS-SERIF BOLD ITALIC CAPITAL OMEGA

Total code points: 1168

==

0061..007A ; Ll # [26] LATIN SMALL LETTER A..LATIN SMALL LETTER Z
00AA ; Ll # FEMININE ORDINAL INDICATOR
00B5 ; Ll # MICRO SIGN
00BA ; Ll # MASCULINE ORDINAL INDICATOR
00DF..00F6 ; Ll # [24] LATIN SMALL LETTER SHARP S..LATIN SMALL LETTER O WITH DIAERESIS
00F8..00FF ; Ll # [8] LATIN SMALL LETTER O WITH STROKE..LATIN SMALL LETTER Y WITH DIAERESIS
0101 ; Ll # LATIN SMALL LETTER A WITH MACRON
0103 ; Ll # LATIN SMALL LETTER A WITH BREVE
0105 ; Ll # LATIN SMALL LETTER A WITH OGONEK
0107 ; Ll # LATIN SMALL LETTER C WITH ACUTE
0109 ; Ll # LATIN SMALL LETTER C WITH CIRCUMFLEX
010B ; Ll # LATIN SMALL LETTER C WITH DOT ABOVE
010D ; Ll # LATIN SMALL LETTER C WITH CARON
010F ; Ll # LATIN SMALL LETTER D WITH CARON
0111 ; Ll # LATIN SMALL LETTER D WITH STROKE
0113 ; Ll # LATIN SMALL LETTER E WITH MACRON
0115 ; Ll # LATIN SMALL LETTER E WITH BREVE
0117 ; Ll # LATIN SMALL LETTER E WITH DOT ABOVE
0119 ; Ll # LATIN SMALL LETTER E WITH OGONEK
011B ; Ll # LATIN SMALL LETTER E WITH CARON
011D ; Ll # LATIN SMALL LETTER G WITH CIRCUMFLEX
011F ; Ll # LATIN SMALL LETTER G WITH BREVE
0121 ; Ll # LATIN SMALL LETTER G WITH DOT ABOVE
0123 ; Ll # LATIN SMALL LETTER G WITH CEDILLA
0125 ; Ll # LATIN SMALL LETTER H WITH CIRCUMFLEX
0127 ; Ll # LATIN SMALL LETTER H WITH STROKE
0129 ; Ll # LATIN SMALL LETTER I WITH TILDE
012B ; Ll # LATIN SMALL LETTER I WITH MACRON
012D ; Ll # LATIN SMALL LETTER I WITH BREVE
012F ; Ll # LATIN SMALL LETTER I WITH OGONEK
0131 ; Ll # LATIN SMALL LETTER DOTLESS I
0133 ; Ll # LATIN SMALL LIGATURE IJ
0135 ; Ll # LATIN SMALL LETTER J WITH CIRCUMFLEX
0137..0138 ; Ll # [2] LATIN SMALL LETTER K WITH CEDILLA..LATIN SMALL LETTER KRA

013A ; Ll # LATIN SMALL LETTER L WITH ACUTE
013C ; Ll # LATIN SMALL LETTER L WITH CEDILLA
013E ; Ll # LATIN SMALL LETTER L WITH CARON
0140 ; Ll # LATIN SMALL LETTER L WITH MIDDLE DOT
0142 ; Ll # LATIN SMALL LETTER L WITH STROKE
0144 ; Ll # LATIN SMALL LETTER N WITH ACUTE
0146 ; Ll # LATIN SMALL LETTER N WITH CEDILLA
0148..0149 ; Ll # [2] LATIN SMALL LETTER N WITH CARON..LATIN SMALL LETTER N PRECEDED BY APOSTROPHE
014B ; Ll # LATIN SMALL LETTER ENG
014D ; Ll # LATIN SMALL LETTER O WITH MACRON
014F ; Ll # LATIN SMALL LETTER O WITH BREVE
0151 ; Ll # LATIN SMALL LETTER O WITH DOUBLE ACUTE
0153 ; Ll # LATIN SMALL LIGATURE OE
0155 ; Ll # LATIN SMALL LETTER R WITH ACUTE
0157 ; Ll # LATIN SMALL LETTER R WITH CEDILLA
0159 ; Ll # LATIN SMALL LETTER R WITH CARON
015B ; Ll # LATIN SMALL LETTER S WITH ACUTE
015D ; Ll # LATIN SMALL LETTER S WITH CIRCUMFLEX
015F ; Ll # LATIN SMALL LETTER S WITH CEDILLA
0161 ; Ll # LATIN SMALL LETTER S WITH CARON
0163 ; Ll # LATIN SMALL LETTER T WITH CEDILLA
0165 ; Ll # LATIN SMALL LETTER T WITH CARON
0167 ; Ll # LATIN SMALL LETTER T WITH STROKE
0169 ; Ll # LATIN SMALL LETTER U WITH TILDE
016B ; Ll # LATIN SMALL LETTER U WITH MACRON
016D ; Ll # LATIN SMALL LETTER U WITH BREVE
016F ; Ll # LATIN SMALL LETTER U WITH RING ABOVE
0171 ; Ll # LATIN SMALL LETTER U WITH DOUBLE ACUTE
0173 ; Ll # LATIN SMALL LETTER U WITH OGONEK
0175 ; Ll # LATIN SMALL LETTER W WITH CIRCUMFLEX
0177 ; Ll # LATIN SMALL LETTER Y WITH CIRCUMFLEX
017A ; Ll # LATIN SMALL LETTER Z WITH ACUTE
017C ; Ll # LATIN SMALL LETTER Z WITH DOT ABOVE
017E..0180 ; Ll # [3] LATIN SMALL LETTER Z WITH CARON..LATIN SMALL LETTER B WITH STROKE
0183 ; Ll # LATIN SMALL LETTER B WITH TOPBAR
0185 ; Ll # LATIN SMALL LETTER TONE SIX
0188 ; Ll # LATIN SMALL LETTER C WITH HOOK
018C..018D ; Ll # [2] LATIN SMALL LETTER D WITH TOPBAR..LATIN SMALL LETTER TURNED DELTA
0192 ; Ll # LATIN SMALL LETTER F WITH HOOK
0195 ; Ll # LATIN SMALL LETTER HV
0199..019B ; Ll # [3] LATIN SMALL LETTER K WITH HOOK..LATIN SMALL LETTER LAMBDA WITH STROKE
019E ; Ll # LATIN SMALL LETTER N WITH LONG RIGHT LEG

01A1 ; Ll # LATIN SMALL LETTER O WITH HORN
01A3 ; Ll # LATIN SMALL LETTER OI
01A5 ; Ll # LATIN SMALL LETTER P WITH HOOK
01A8 ; Ll # LATIN SMALL LETTER TONE TWO
01AA..01AB ; Ll # [2] LATIN LETTER REVERSED ESH LOOP..LATIN SMALL LETTER T WITH PALATAL HOOK
01AD ; Ll # LATIN SMALL LETTER T WITH HOOK
01B0 ; Ll # LATIN SMALL LETTER U WITH HORN
01B4 ; Ll # LATIN SMALL LETTER Y WITH HOOK
01B6 ; Ll # LATIN SMALL LETTER Z WITH STROKE
01B9..01BA ; Ll # [2] LATIN SMALL LETTER EZH REVERSED..LATIN SMALL LETTER EZH WITH TAIL
01BD..01BF ; Ll # [3] LATIN SMALL LETTER TONE FIVE..LATIN LETTER WYNN
01C6 ; Ll # LATIN SMALL LETTER DZ WITH CARON
01C9 ; Ll # LATIN SMALL LETTER LJ
01CC ; Ll # LATIN SMALL LETTER NJ
01CE ; Ll # LATIN SMALL LETTER A WITH CARON
01D0 ; Ll # LATIN SMALL LETTER I WITH CARON
01D2 ; Ll # LATIN SMALL LETTER O WITH CARON
01D4 ; Ll # LATIN SMALL LETTER U WITH CARON
01D6 ; Ll # LATIN SMALL LETTER U WITH DIAERESIS AND MACRON
01D8 ; Ll # LATIN SMALL LETTER U WITH DIAERESIS AND ACUTE
01DA ; Ll # LATIN SMALL LETTER U WITH DIAERESIS AND CARON
01DC..01DD ; Ll # [2] LATIN SMALL LETTER U WITH DIAERESIS AND GRAVE..LATIN SMALL LETTER TURNED E
01DF ; Ll # LATIN SMALL LETTER A WITH DIAERESIS AND MACRON
01E1 ; Ll # LATIN SMALL LETTER A WITH DOT ABOVE AND MACRON
01E3 ; Ll # LATIN SMALL LETTER AE WITH MACRON
01E5 ; Ll # LATIN SMALL LETTER G WITH STROKE
01E7 ; Ll # LATIN SMALL LETTER G WITH CARON
01E9 ; Ll # LATIN SMALL LETTER K WITH CARON
01EB ; Ll # LATIN SMALL LETTER O WITH OGONEK
01ED ; Ll # LATIN SMALL LETTER O WITH OGONEK AND MACRON
01EF..01F0 ; Ll # [2] LATIN SMALL LETTER EZH WITH CARON..LATIN SMALL LETTER J WITH CARON
01F3 ; Ll # LATIN SMALL LETTER DZ
01F5 ; Ll # LATIN SMALL LETTER G WITH ACUTE
01F9 ; Ll # LATIN SMALL LETTER N WITH GRAVE
01FB ; Ll # LATIN SMALL LETTER A WITH RING ABOVE AND ACUTE
01FD ; Ll # LATIN SMALL LETTER AE WITH ACUTE
01FF ; Ll # LATIN SMALL LETTER O WITH STROKE AND ACUTE
0201 ; Ll # LATIN SMALL LETTER A WITH DOUBLE GRAVE
0203 ; Ll # LATIN SMALL LETTER A WITH INVERTED BREVE
0205 ; Ll # LATIN SMALL LETTER E WITH DOUBLE GRAVE
0207 ; Ll # LATIN SMALL LETTER E WITH INVERTED BREVE
0209 ; Ll # LATIN SMALL LETTER I WITH DOUBLE GRAVE

020B ; Ll # LATIN SMALL LETTER I WITH INVERTED BREVE
020D ; Ll # LATIN SMALL LETTER O WITH DOUBLE GRAVE
020F ; Ll # LATIN SMALL LETTER O WITH INVERTED BREVE
0211 ; Ll # LATIN SMALL LETTER R WITH DOUBLE GRAVE
0213 ; Ll # LATIN SMALL LETTER R WITH INVERTED BREVE
0215 ; Ll # LATIN SMALL LETTER U WITH DOUBLE GRAVE
0217 ; Ll # LATIN SMALL LETTER U WITH INVERTED BREVE
0219 ; Ll # LATIN SMALL LETTER S WITH COMMA BELOW
021B ; Ll # LATIN SMALL LETTER T WITH COMMA BELOW
021D ; Ll # LATIN SMALL LETTER YOGH
021F ; Ll # LATIN SMALL LETTER H WITH CARON
0223 ; Ll # LATIN SMALL LETTER OU
0225 ; Ll # LATIN SMALL LETTER Z WITH HOOK
0227 ; Ll # LATIN SMALL LETTER A WITH DOT ABOVE
0229 ; Ll # LATIN SMALL LETTER E WITH CEDILLA
022B ; Ll # LATIN SMALL LETTER O WITH DIAERESIS AND MACRON
022D ; Ll # LATIN SMALL LETTER O WITH TILDE AND MACRON
022F ; Ll # LATIN SMALL LETTER O WITH DOT ABOVE
0231 ; Ll # LATIN SMALL LETTER O WITH DOT ABOVE AND MACRON
0233 ; Ll # LATIN SMALL LETTER Y WITH MACRON
0250..02AD ; Ll # [94] LATIN SMALL LETTER TURNED A..LATIN LETTER BIDENTAL PERCUSSIVE
0390 ; Ll # GREEK SMALL LETTER IOTA WITH DIALYTIKA AND TONOS
03AC..03CE ; Ll # [35] GREEK SMALL LETTER ALPHA WITH TONOS..GREEK SMALL LETTER OMEGA WITH TONOS
03D0..03D1 ; Ll # [2] GREEK BETA SYMBOL..GREEK THETA SYMBOL
03D5..03D7 ; Ll # [3] GREEK PHI SYMBOL..GREEK KAI SYMBOL
03DB ; Ll # GREEK SMALL LETTER STIGMA
03DD ; Ll # GREEK SMALL LETTER DIGAMMA
03DF ; Ll # GREEK SMALL LETTER KOPPA
03E1 ; Ll # GREEK SMALL LETTER SAMPI
03E3 ; Ll # COPTIC SMALL LETTER SHEI
03E5 ; Ll # COPTIC SMALL LETTER FEI
03E7 ; Ll # COPTIC SMALL LETTER KHEI
03E9 ; Ll # COPTIC SMALL LETTER HORI
03EB ; Ll # COPTIC SMALL LETTER GANGIA
03ED ; Ll # COPTIC SMALL LETTER SHIMA
03EF..03F3 ; Ll # [5] COPTIC SMALL LETTER DEI..GREEK LETTER YOT
03F5 ; Ll # GREEK LUNATE EPSILON SYMBOL
0430..045F ; Ll # [48] CYRILLIC SMALL LETTER A..CYRILLIC SMALL LETTER DZHE
0461 ; Ll # CYRILLIC SMALL LETTER OMEGA
0463 ; Ll # CYRILLIC SMALL LETTER YAT
0465 ; Ll # CYRILLIC SMALL LETTER IOTIFIED E
0467 ; Ll # CYRILLIC SMALL LETTER LITTLE YUS
0469 ; Ll # CYRILLIC SMALL LETTER IOTIFIED LITTLE YUS
046B ; Ll # CYRILLIC SMALL LETTER BIG YUS

046D ; Ll # CYRILLIC SMALL LETTER IOTIFIED BIG YUS
046F ; Ll # CYRILLIC SMALL LETTER KSI
0471 ; Ll # CYRILLIC SMALL LETTER PSI
0473 ; Ll # CYRILLIC SMALL LETTER FITA
0475 ; Ll # CYRILLIC SMALL LETTER IZHITSA
0477 ; Ll # CYRILLIC SMALL LETTER IZHITSA WITH DOUBLE GRAVE ACCENT
0479 ; Ll # CYRILLIC SMALL LETTER UK
047B ; Ll # CYRILLIC SMALL LETTER ROUND OMEGA
047D ; Ll # CYRILLIC SMALL LETTER OMEGA WITH TITLO
047F ; Ll # CYRILLIC SMALL LETTER OT
0481 ; Ll # CYRILLIC SMALL LETTER KOPPA
048D ; Ll # CYRILLIC SMALL LETTER SEMISOFT SIGN
048F ; Ll # CYRILLIC SMALL LETTER ER WITH TICK
0491 ; Ll # CYRILLIC SMALL LETTER GHE WITH UPTURN
0493 ; Ll # CYRILLIC SMALL LETTER GHE WITH STROKE
0495 ; Ll # CYRILLIC SMALL LETTER GHE WITH MIDDLE HOOK
0497 ; Ll # CYRILLIC SMALL LETTER ZHE WITH DESCENDER
0499 ; Ll # CYRILLIC SMALL LETTER ZE WITH DESCENDER
049B ; Ll # CYRILLIC SMALL LETTER KA WITH DESCENDER
049D ; Ll # CYRILLIC SMALL LETTER KA WITH VERTICAL STROKE
049F ; Ll # CYRILLIC SMALL LETTER KA WITH STROKE
04A1 ; Ll # CYRILLIC SMALL LETTER BASHKIR KA
04A3 ; Ll # CYRILLIC SMALL LETTER EN WITH DESCENDER
04A5 ; Ll # CYRILLIC SMALL LIGATURE EN GHE
04A7 ; Ll # CYRILLIC SMALL LETTER PE WITH MIDDLE HOOK
04A9 ; Ll # CYRILLIC SMALL LETTER ABKHASIAN HA
04AB ; Ll # CYRILLIC SMALL LETTER ES WITH DESCENDER
04AD ; Ll # CYRILLIC SMALL LETTER TE WITH DESCENDER
04AF ; Ll # CYRILLIC SMALL LETTER STRAIGHT U
04B1 ; Ll # CYRILLIC SMALL LETTER STRAIGHT U WITH STROKE
04B3 ; Ll # CYRILLIC SMALL LETTER HA WITH DESCENDER
04B5 ; Ll # CYRILLIC SMALL LIGATURE TE TSE
04B7 ; Ll # CYRILLIC SMALL LETTER CHE WITH DESCENDER
04B9 ; Ll # CYRILLIC SMALL LETTER CHE WITH VERTICAL STROKE
04BB ; Ll # CYRILLIC SMALL LETTER SHHA
04BD ; Ll # CYRILLIC SMALL LETTER ABKHASIAN CHE
04BF ; Ll # CYRILLIC SMALL LETTER ABKHASIAN CHE WITH DESCENDER
04C2 ; Ll # CYRILLIC SMALL LETTER ZHE WITH BREVE
04C4 ; Ll # CYRILLIC SMALL LETTER KA WITH HOOK
04C8 ; Ll # CYRILLIC SMALL LETTER EN WITH HOOK
04CC ; Ll # CYRILLIC SMALL LETTER KHAKASSIAN CHE
04D1 ; Ll # CYRILLIC SMALL LETTER A WITH BREVE
04D3 ; Ll # CYRILLIC SMALL LETTER A WITH DIAERESIS
04D5 ; Ll # CYRILLIC SMALL LIGATURE A IE
04D7 ; Ll # CYRILLIC SMALL LETTER IE WITH BREVE
04D9 ; Ll # CYRILLIC SMALL LETTER SCHWA

04DB ; Ll # CYRILLIC SMALL LETTER SCHWA WITH DIAERESIS
04DD ; Ll # CYRILLIC SMALL LETTER ZHE WITH DIAERESIS
04DF ; Ll # CYRILLIC SMALL LETTER ZE WITH DIAERESIS
04E1 ; Ll # CYRILLIC SMALL LETTER ABKHASIAN DZE
04E3 ; Ll # CYRILLIC SMALL LETTER I WITH MACRON
04E5 ; Ll # CYRILLIC SMALL LETTER I WITH DIAERESIS
04E7 ; Ll # CYRILLIC SMALL LETTER O WITH DIAERESIS
04E9 ; Ll # CYRILLIC SMALL LETTER BARRED O
04EB ; Ll # CYRILLIC SMALL LETTER BARRED O WITH DIAERESIS
04ED ; Ll # CYRILLIC SMALL LETTER E WITH DIAERESIS
04EF ; Ll # CYRILLIC SMALL LETTER U WITH MACRON
04F1 ; Ll # CYRILLIC SMALL LETTER U WITH DIAERESIS
04F3 ; Ll # CYRILLIC SMALL LETTER U WITH DOUBLE ACUTE
04F5 ; Ll # CYRILLIC SMALL LETTER CHE WITH DIAERESIS
04F9 ; Ll # CYRILLIC SMALL LETTER YERU WITH DIAERESIS
0561..0587 ; Ll # [39] ARMENIAN SMALL LETTER AYB..ARMENIAN SMALL LIGATURE ECH YIWN
1E01 ; Ll # LATIN SMALL LETTER A WITH RING BELOW
1E03 ; Ll # LATIN SMALL LETTER B WITH DOT ABOVE
1E05 ; Ll # LATIN SMALL LETTER B WITH DOT BELOW
1E07 ; Ll # LATIN SMALL LETTER B WITH LINE BELOW
1E09 ; Ll # LATIN SMALL LETTER C WITH CEDILLA AND ACUTE
1E0B ; Ll # LATIN SMALL LETTER D WITH DOT ABOVE
1E0D ; Ll # LATIN SMALL LETTER D WITH DOT BELOW
1E0F ; Ll # LATIN SMALL LETTER D WITH LINE BELOW
1E11 ; Ll # LATIN SMALL LETTER D WITH CEDILLA
1E13 ; Ll # LATIN SMALL LETTER D WITH CIRCUMFLEX BELOW
1E15 ; Ll # LATIN SMALL LETTER E WITH MACRON AND GRAVE
1E17 ; Ll # LATIN SMALL LETTER E WITH MACRON AND ACUTE
1E19 ; Ll # LATIN SMALL LETTER E WITH CIRCUMFLEX BELOW
1E1B ; Ll # LATIN SMALL LETTER E WITH TILDE BELOW
1E1D ; Ll # LATIN SMALL LETTER E WITH CEDILLA AND BREVE
1E1F ; Ll # LATIN SMALL LETTER F WITH DOT ABOVE
1E21 ; Ll # LATIN SMALL LETTER G WITH MACRON
1E23 ; Ll # LATIN SMALL LETTER H WITH DOT ABOVE
1E25 ; Ll # LATIN SMALL LETTER H WITH DOT BELOW
1E27 ; Ll # LATIN SMALL LETTER H WITH DIAERESIS
1E29 ; Ll # LATIN SMALL LETTER H WITH CEDILLA
1E2B ; Ll # LATIN SMALL LETTER H WITH BREVE BELOW
1E2D ; Ll # LATIN SMALL LETTER I WITH TILDE BELOW
1E2F ; Ll # LATIN SMALL LETTER I WITH DIAERESIS AND ACUTE
1E31 ; Ll # LATIN SMALL LETTER K WITH ACUTE
1E33 ; Ll # LATIN SMALL LETTER K WITH DOT BELOW
1E35 ; Ll # LATIN SMALL LETTER K WITH LINE BELOW
1E37 ; Ll # LATIN SMALL LETTER L WITH DOT BELOW
1E39 ; Ll # LATIN SMALL LETTER L WITH DOT BELOW AND MACRON

1E3B ; Ll # LATIN SMALL LETTER L WITH LINE BELOW
1E3D ; Ll # LATIN SMALL LETTER L WITH CIRCUMFLEX BELOW
1E3F ; Ll # LATIN SMALL LETTER M WITH ACUTE
1E41 ; Ll # LATIN SMALL LETTER M WITH DOT ABOVE
1E43 ; Ll # LATIN SMALL LETTER M WITH DOT BELOW
1E45 ; Ll # LATIN SMALL LETTER N WITH DOT ABOVE
1E47 ; Ll # LATIN SMALL LETTER N WITH DOT BELOW
1E49 ; Ll # LATIN SMALL LETTER N WITH LINE BELOW
1E4B ; Ll # LATIN SMALL LETTER N WITH CIRCUMFLEX BELOW
1E4D ; Ll # LATIN SMALL LETTER O WITH TILDE AND ACUTE
1E4F ; Ll # LATIN SMALL LETTER O WITH TILDE AND DIAERESIS
1E51 ; Ll # LATIN SMALL LETTER O WITH MACRON AND GRAVE
1E53 ; Ll # LATIN SMALL LETTER O WITH MACRON AND ACUTE
1E55 ; Ll # LATIN SMALL LETTER P WITH ACUTE
1E57 ; Ll # LATIN SMALL LETTER P WITH DOT ABOVE
1E59 ; Ll # LATIN SMALL LETTER R WITH DOT ABOVE
1E5B ; Ll # LATIN SMALL LETTER R WITH DOT BELOW
1E5D ; Ll # LATIN SMALL LETTER R WITH DOT BELOW AND MACRON
1E5F ; Ll # LATIN SMALL LETTER R WITH LINE BELOW
1E61 ; Ll # LATIN SMALL LETTER S WITH DOT ABOVE
1E63 ; Ll # LATIN SMALL LETTER S WITH DOT BELOW
1E65 ; Ll # LATIN SMALL LETTER S WITH ACUTE AND DOT ABOVE
1E67 ; Ll # LATIN SMALL LETTER S WITH CARON AND DOT ABOVE
1E69 ; Ll # LATIN SMALL LETTER S WITH DOT BELOW AND DOT ABOVE
1E6B ; Ll # LATIN SMALL LETTER T WITH DOT ABOVE
1E6D ; Ll # LATIN SMALL LETTER T WITH DOT BELOW
1E6F ; Ll # LATIN SMALL LETTER T WITH LINE BELOW
1E71 ; Ll # LATIN SMALL LETTER T WITH CIRCUMFLEX BELOW
1E73 ; Ll # LATIN SMALL LETTER U WITH DIAERESIS BELOW
1E75 ; Ll # LATIN SMALL LETTER U WITH TILDE BELOW
1E77 ; Ll # LATIN SMALL LETTER U WITH CIRCUMFLEX BELOW
1E79 ; Ll # LATIN SMALL LETTER U WITH TILDE AND ACUTE
1E7B ; Ll # LATIN SMALL LETTER U WITH MACRON AND DIAERESIS
1E7D ; Ll # LATIN SMALL LETTER V WITH TILDE
1E7F ; Ll # LATIN SMALL LETTER V WITH DOT BELOW
1E81 ; Ll # LATIN SMALL LETTER W WITH GRAVE
1E83 ; Ll # LATIN SMALL LETTER W WITH ACUTE
1E85 ; Ll # LATIN SMALL LETTER W WITH DIAERESIS
1E87 ; Ll # LATIN SMALL LETTER W WITH DOT ABOVE
1E89 ; Ll # LATIN SMALL LETTER W WITH DOT BELOW
1E8B ; Ll # LATIN SMALL LETTER X WITH DOT ABOVE
1E8D ; Ll # LATIN SMALL LETTER X WITH DIAERESIS
1E8F ; Ll # LATIN SMALL LETTER Y WITH DOT ABOVE
1E91 ; Ll # LATIN SMALL LETTER Z WITH CIRCUMFLEX
1E93 ; Ll # LATIN SMALL LETTER Z WITH DOT BELOW
1E95..1E9B ; Ll # [7] LATIN SMALL LETTER Z WITH LINE BELOW..LATIN SMALL

LETTER LONG S WITH DOT ABOVE
1EA1 ; Ll # LATIN SMALL LETTER A WITH DOT BELOW
1EA3 ; Ll # LATIN SMALL LETTER A WITH HOOK ABOVE
1EA5 ; Ll # LATIN SMALL LETTER A WITH CIRCUMFLEX AND ACUTE
1EA7 ; Ll # LATIN SMALL LETTER A WITH CIRCUMFLEX AND GRAVE
1EA9 ; Ll # LATIN SMALL LETTER A WITH CIRCUMFLEX AND HOOK ABOVE
1EAB ; Ll # LATIN SMALL LETTER A WITH CIRCUMFLEX AND TILDE
1EAD ; Ll # LATIN SMALL LETTER A WITH CIRCUMFLEX AND DOT BELOW
1EAF ; Ll # LATIN SMALL LETTER A WITH BREVE AND ACUTE
1EB1 ; Ll # LATIN SMALL LETTER A WITH BREVE AND GRAVE
1EB3 ; Ll # LATIN SMALL LETTER A WITH BREVE AND HOOK ABOVE
1EB5 ; Ll # LATIN SMALL LETTER A WITH BREVE AND TILDE
1EB7 ; Ll # LATIN SMALL LETTER A WITH BREVE AND DOT BELOW
1EB9 ; Ll # LATIN SMALL LETTER E WITH DOT BELOW
1EBB ; Ll # LATIN SMALL LETTER E WITH HOOK ABOVE
1EBD ; Ll # LATIN SMALL LETTER E WITH TILDE
1EBF ; Ll # LATIN SMALL LETTER E WITH CIRCUMFLEX AND ACUTE
1EC1 ; Ll # LATIN SMALL LETTER E WITH CIRCUMFLEX AND GRAVE
1EC3 ; Ll # LATIN SMALL LETTER E WITH CIRCUMFLEX AND HOOK ABOVE
1EC5 ; Ll # LATIN SMALL LETTER E WITH CIRCUMFLEX AND TILDE
1EC7 ; Ll # LATIN SMALL LETTER E WITH CIRCUMFLEX AND DOT BELOW
1EC9 ; Ll # LATIN SMALL LETTER I WITH HOOK ABOVE
1ECB ; Ll # LATIN SMALL LETTER I WITH DOT BELOW
1ECD ; Ll # LATIN SMALL LETTER O WITH DOT BELOW
1ECF ; Ll # LATIN SMALL LETTER O WITH HOOK ABOVE
1ED1 ; Ll # LATIN SMALL LETTER O WITH CIRCUMFLEX AND ACUTE
1ED3 ; Ll # LATIN SMALL LETTER O WITH CIRCUMFLEX AND GRAVE
1ED5 ; Ll # LATIN SMALL LETTER O WITH CIRCUMFLEX AND HOOK ABOVE
1ED7 ; Ll # LATIN SMALL LETTER O WITH CIRCUMFLEX AND TILDE
1ED9 ; Ll # LATIN SMALL LETTER O WITH CIRCUMFLEX AND DOT BELOW
1EDB ; Ll # LATIN SMALL LETTER O WITH HORN AND ACUTE
1EDD ; Ll # LATIN SMALL LETTER O WITH HORN AND GRAVE
1EDF ; Ll # LATIN SMALL LETTER O WITH HORN AND HOOK ABOVE
1EE1 ; Ll # LATIN SMALL LETTER O WITH HORN AND TILDE
1EE3 ; Ll # LATIN SMALL LETTER O WITH HORN AND DOT BELOW
1EE5 ; Ll # LATIN SMALL LETTER U WITH DOT BELOW
1EE7 ; Ll # LATIN SMALL LETTER U WITH HOOK ABOVE
1EE9 ; Ll # LATIN SMALL LETTER U WITH HORN AND ACUTE
1EEB ; Ll # LATIN SMALL LETTER U WITH HORN AND GRAVE
1EED ; Ll # LATIN SMALL LETTER U WITH HORN AND HOOK ABOVE
1EEF ; Ll # LATIN SMALL LETTER U WITH HORN AND TILDE
1EF1 ; Ll # LATIN SMALL LETTER U WITH HORN AND DOT BELOW
1EF3 ; Ll # LATIN SMALL LETTER Y WITH GRAVE
1EF5 ; Ll # LATIN SMALL LETTER Y WITH DOT BELOW
1EF7 ; Ll # LATIN SMALL LETTER Y WITH HOOK ABOVE
1EF9 ; Ll # LATIN SMALL LETTER Y WITH TILDE

1F00..1F07 ; Ll # [8] GREEK SMALL LETTER ALPHA WITH PSILI..GREEK SMALL LETTER ALPHA WITH DASIA AND PERISPOMENI
1F10..1F15 ; Ll # [6] GREEK SMALL LETTER EPSILON WITH PSILI..GREEK SMALL LETTER EPSILON WITH DASIA AND OXIA
1F20..1F27 ; Ll # [8] GREEK SMALL LETTER ETA WITH PSILI..GREEK SMALL LETTER ETA WITH DASIA AND PERISPOMENI
1F30..1F37 ; Ll # [8] GREEK SMALL LETTER IOTA WITH PSILI..GREEK SMALL LETTER IOTA WITH DASIA AND PERISPOMENI
1F40..1F45 ; Ll # [6] GREEK SMALL LETTER OMICRON WITH PSILI..GREEK SMALL LETTER OMICRON WITH DASIA AND OXIA
1F50..1F57 ; Ll # [8] GREEK SMALL LETTER UPSILON WITH PSILI..GREEK SMALL LETTER UPSILON WITH DASIA AND PERISPOMENI
1F60..1F67 ; Ll # [8] GREEK SMALL LETTER OMEGA WITH PSILI..GREEK SMALL LETTER OMEGA WITH DASIA AND PERISPOMENI
1F70..1F7D ; Ll # [14] GREEK SMALL LETTER ALPHA WITH VARIA..GREEK SMALL LETTER OMEGA WITH OXIA
1F80..1F87 ; Ll # [8] GREEK SMALL LETTER ALPHA WITH PSILI AND YPOGEGRAMMENI..GREEK SMALL LETTER ALPHA WITH DASIA AND PERISPOMENI AND YPOGEGRAMMENI
1F90..1F97 ; Ll # [8] GREEK SMALL LETTER ETA WITH PSILI AND YPOGEGRAMMENI..GREEK SMALL LETTER ETA WITH DASIA AND PERISPOMENI AND YPOGEGRAMMENI
1FA0..1FA7 ; Ll # [8] GREEK SMALL LETTER OMEGA WITH PSILI AND YPOGEGRAMMENI..GREEK SMALL LETTER OMEGA WITH DASIA AND PERISPOMENI AND YPOGEGRAMMENI
1FB0..1FB4 ; Ll # [5] GREEK SMALL LETTER ALPHA WITH VRACHY..GREEK SMALL LETTER ALPHA WITH OXIA AND YPOGEGRAMMENI
1FB6..1FB7 ; Ll # [2] GREEK SMALL LETTER ALPHA WITH PERISPOMENI..GREEK SMALL LETTER ALPHA WITH PERISPOMENI AND YPOGEGRAMMENI
1FBE ; Ll # GREEK PROSGEGRAMMENI
1FC2..1FC4 ; Ll # [3] GREEK SMALL LETTER ETA WITH VARIA AND YPOGEGRAMMENI..GREEK SMALL LETTER ETA WITH OXIA AND YPOGEGRAMMENI
1FC6..1FC7 ; Ll # [2] GREEK SMALL LETTER ETA WITH PERISPOMENI..GREEK SMALL LETTER ETA WITH PERISPOMENI AND YPOGEGRAMMENI
1FD0..1FD3 ; Ll # [4] GREEK SMALL LETTER IOTA WITH VRACHY..GREEK SMALL LETTER IOTA WITH DIALYTIKA AND OXIA
1FD6..1FD7 ; Ll # [2] GREEK SMALL LETTER IOTA WITH PERISPOMENI..GREEK SMALL LETTER IOTA WITH DIALYTIKA AND PERISPOMENI
1FE0..1FE7 ; Ll # [8] GREEK SMALL LETTER UPSILON WITH VRACHY..GREEK SMALL LETTER UPSILON WITH DIALYTIKA AND PERISPOMENI
1FF2..1FF4 ; Ll # [3] GREEK SMALL LETTER OMEGA WITH VARIA AND YPOGEGRAMMENI..GREEK SMALL LETTER OMEGA WITH OXIA AND YPOGEGRAMMENI
1FF6..1FF7 ; Ll # [2] GREEK SMALL LETTER OMEGA WITH PERISPOMENI..GREEK SMALL LETTER OMEGA WITH PERISPOMENI AND YPOGEGRAMMENI
207F ; Ll # SUPERSCRIPT LATIN SMALL LETTER N
210A ; Ll # SCRIPT SMALL G

210E..210F ; Ll # [2] PLANCK CONSTANT..PLANCK CONSTANT OVER TWO PI
2113 ; Ll # SCRIPT SMALL L
212F ; Ll # SCRIPT SMALL E
2134 ; Ll # SCRIPT SMALL O
2139 ; Ll # INFORMATION SOURCE
FB00..FB06 ; Ll # [7] LATIN SMALL LIGATURE FF..LATIN SMALL LIGATURE ST
FB13..FB17 ; Ll # [5] ARMENIAN SMALL LIGATURE MEN NOW..ARMENIAN SMALL LIGATURE MEN XEH
FF41..FF5A ; Ll # [26] FULLWIDTH LATIN SMALL LETTER A..FULLWIDTH LATIN SMALL LETTER Z
10428..1044D ; Ll # [38] DESERET SMALL LETTER LONG I..DESERET SMALL LETTER ENG
1D41A..1D433 ; Ll # [26] MATHEMATICAL BOLD SMALL A..MATHEMATICAL BOLD SMALL Z
1D44E..1D454 ; Ll # [7] MATHEMATICAL ITALIC SMALL A..MATHEMATICAL ITALIC SMALL G
1D456..1D467 ; Ll # [18] MATHEMATICAL ITALIC SMALL I..MATHEMATICAL ITALIC SMALL Z
1D482..1D49B ; Ll # [26] MATHEMATICAL BOLD ITALIC SMALL A..MATHEMATICAL BOLD ITALIC SMALL Z
1D4B6..1D4B9 ; Ll # [4] MATHEMATICAL SCRIPT SMALL A..MATHEMATICAL SCRIPT SMALL D
1D4BB ; Ll # MATHEMATICAL SCRIPT SMALL F
1D4BD..1D4C0 ; Ll # [4] MATHEMATICAL SCRIPT SMALL H..MATHEMATICAL SCRIPT SMALL K
1D4C2..1D4C3 ; Ll # [2] MATHEMATICAL SCRIPT SMALL M..MATHEMATICAL SCRIPT SMALL N
1D4C5..1D4CF ; Ll # [11] MATHEMATICAL SCRIPT SMALL P..MATHEMATICAL SCRIPT SMALL Z
1D4EA..1D503 ; Ll # [26] MATHEMATICAL BOLD SCRIPT SMALL A..MATHEMATICAL BOLD SCRIPT SMALL Z
1D51E..1D537 ; Ll # [26] MATHEMATICAL FRAKTUR SMALL A..MATHEMATICAL FRAKTUR SMALL Z
1D552..1D56B ; Ll # [26] MATHEMATICAL DOUBLE-STRUCK SMALL A..MATHEMATICAL DOUBLE-STRUCK SMALL Z
1D586..1D59F ; Ll # [26] MATHEMATICAL BOLD FRAKTUR SMALL A..MATHEMATICAL BOLD FRAKTUR SMALL Z
1D5BA..1D5D3 ; Ll # [26] MATHEMATICAL SANS-SERIF SMALL A..MATHEMATICAL SANS-SERIF SMALL Z
1D5EE..1D607 ; Ll # [26] MATHEMATICAL SANS-SERIF BOLD SMALL A..MATHEMATICAL SANS-SERIF BOLD SMALL Z
1D622..1D63B ; Ll # [26] MATHEMATICAL SANS-SERIF ITALIC SMALL A..MATHEMATICAL SANS-SERIF ITALIC SMALL Z
1D656..1D66F ; Ll # [26] MATHEMATICAL SANS-SERIF BOLD ITALIC SMALL A..MATHEMATICAL SANS-SERIF BOLD ITALIC SMALL Z
1D68A..1D6A3 ; Ll # [26] MATHEMATICAL MONOSPACE SMALL A..MATHEMATICAL

MONOSPACE SMALL Z
1D6C2..1D6DA ; Ll # [25] MATHEMATICAL BOLD SMALL ALPHA..MATHEMATICAL BOLD SMALL OMEGA
1D6DC..1D6E1 ; Ll # [6] MATHEMATICAL BOLD EPSILON SYMBOL..MATHEMATICAL BOLD PI SYMBOL
1D6FC..1D714 ; Ll # [25] MATHEMATICAL ITALIC SMALL ALPHA..MATHEMATICAL ITALIC SMALL OMEGA
1D716..1D71B ; Ll # [6] MATHEMATICAL ITALIC EPSILON SYMBOL..MATHEMATICAL ITALIC PI SYMBOL
1D736..1D74E ; Ll # [25] MATHEMATICAL BOLD ITALIC SMALL ALPHA..
MATHEMATICAL BOLD ITALIC SMALL OMEGA
1D750..1D755 ; Ll # [6] MATHEMATICAL BOLD ITALIC EPSILON SYMBOL..
MATHEMATICAL BOLD ITALIC PI SYMBOL
1D770..1D788 ; Ll # [25] MATHEMATICAL SANS-SERIF BOLD SMALL ALPHA..
MATHEMATICAL SANS-SERIF BOLD SMALL OMEGA
1D78A..1D78F ; Ll # [6] MATHEMATICAL SANS-SERIF BOLD EPSILON SYMBOL..
MATHEMATICAL SANS-SERIF BOLD PI SYMBOL
1D7AA..1D7C2 ; Ll # [25] MATHEMATICAL SANS-SERIF BOLD ITALIC SMALL ALPHA..
MATHEMATICAL SANS-SERIF BOLD ITALIC SMALL OMEGA
1D7C4..1D7C9 ; Ll # [6] MATHEMATICAL SANS-SERIF BOLD ITALIC EPSILON SYMBOL..
MATHEMATICAL SANS-SERIF BOLD ITALIC PI SYMBOL

Total code points: 1331

==

01C5 ; Lt # LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON
01C8 ; Lt # LATIN CAPITAL LETTER L WITH SMALL LETTER J
01CB ; Lt # LATIN CAPITAL LETTER N WITH SMALL LETTER J
01F2 ; Lt # LATIN CAPITAL LETTER D WITH SMALL LETTER Z
1F88..1F8F ; Lt # [8] GREEK CAPITAL LETTER ALPHA WITH PSILI AND PROSGEGRAMMENI..GREEK CAPITAL LETTER ALPHA WITH DASIA AND PERISPOMENI AND PROSGEGRAMMENI
1F98..1F9F ; Lt # [8] GREEK CAPITAL LETTER ETA WITH PSILI AND PROSGEGRAMMENI..GREEK CAPITAL LETTER ETA WITH DASIA AND PERISPOMENI AND PROSGEGRAMMENI
1FA8..1FAF ; Lt # [8] GREEK CAPITAL LETTER OMEGA WITH PSILI AND PROSGEGRAMMENI..GREEK CAPITAL LETTER OMEGA WITH DASIA AND PERISPOMENI AND PROSGEGRAMMENI
1FBC ; Lt # GREEK CAPITAL LETTER ALPHA WITH PROSGEGRAMMENI
1FCC ; Lt # GREEK CAPITAL LETTER ETA WITH PROSGEGRAMMENI
1FFC ; Lt # GREEK CAPITAL LETTER OMEGA WITH PROSGEGRAMMENI

Total code points: 31

==

02B0..02B8 ; Lm # [9] MODIFIER LETTER SMALL H..MODIFIER LETTER SMALL Y
02BB..02C1 ; Lm # [7] MODIFIER LETTER TURNED COMMA..MODIFIER LETTER REVERSED GLOTTAL STOP
02D0..02D1 ; Lm # [2] MODIFIER LETTER TRIANGULAR COLON..MODIFIER LETTER HALF TRIANGULAR COLON
02E0..02E4 ; Lm # [5] MODIFIER LETTER SMALL GAMMA..MODIFIER LETTER SMALL REVERSED GLOTTAL STOP
02EE ; Lm # MODIFIER LETTER DOUBLE APOSTROPHE
037A ; Lm # GREEK YPOGEGRAMMENI
0559 ; Lm # ARMENIAN MODIFIER LETTER LEFT HALF RING
0640 ; Lm # ARABIC TATWEEL
06E5..06E6 ; Lm # [2] ARABIC SMALL WAW..ARABIC SMALL YEH
0E46 ; Lm # THAI CHARACTER MAIYAMOK
0EC6 ; Lm # LAO KO LA
1843 ; Lm # MONGOLIAN LETTER TODO LONG VOWEL SIGN
3005 ; Lm # IDEOGRAPHIC ITERATION MARK
3031..3035 ; Lm # [5] VERTICAL KANA REPEAT MARK..VERTICAL KANA REPEAT MARK LOWER HALF
309D..309E ; Lm # [2] HIRAGANA ITERATION MARK..HIRAGANA VOICED ITERATION MARK
30FC..30FE ; Lm # [3] KATAKANA-HIRAGANA PROLONGED SOUND MARK.. KATAKANA VOICED ITERATION MARK
FF70 ; Lm # HALFWIDTH KATAKANA-HIRAGANA PROLONGED SOUND MARK
FF9E..FF9F ; Lm # [2] HALFWIDTH KATAKANA VOICED SOUND MARK..HALFWIDTH KATAKANA SEMI-VOICED SOUND MARK

Total code points: 46

==

01BB ; Lo # LATIN LETTER TWO WITH STROKE
01C0..01C3 ; Lo # [4] LATIN LETTER DENTAL CLICK..LATIN LETTER RETROFLEX CLICK
05D0..05EA ; Lo # [27] HEBREW LETTER ALEF..HEBREW LETTER TAV
05F0..05F2 ; Lo # [3] HEBREW LIGATURE YIDDISH DOUBLE VAV..HEBREW LIGATURE YIDDISH DOUBLE YOD
0621..063A ; Lo # [26] ARABIC LETTER HAMZA..ARABIC LETTER GHAIN
0641..064A ; Lo # [10] ARABIC LETTER FEH..ARABIC LETTER YEH
0671..06D3 ; Lo # [99] ARABIC LETTER ALEF WASLA..ARABIC LETTER YEH BARREE WITH HAMZA ABOVE
06D5 ; Lo # ARABIC LETTER AE
06FA..06FC ; Lo # [3] ARABIC LETTER SHEEN WITH DOT BELOW..ARABIC LETTER GHAIN WITH DOT BELOW
0710 ; Lo # SYRIAC LETTER ALAPH
0712..072C ; Lo # [27] SYRIAC LETTER BETH..SYRIAC LETTER TAW

0780..07A5 ; Lo # [38] THAANA LETTER HAA..THAANA LETTER WAAVU
0905..0939 ; Lo # [53] DEVANAGARI LETTER A..DEVANAGARI LETTER HA
093D ; Lo # DEVANAGARI SIGN AVAGRAHA
0950 ; Lo # DEVANAGARI OM
0958..0961 ; Lo # [10] DEVANAGARI LETTER QA..DEVANAGARI LETTER VOCALIC LL
0985..098C ; Lo # [8] BENGALI LETTER A..BENGALI LETTER VOCALIC L
098F..0990 ; Lo # [2] BENGALI LETTER E..BENGALI LETTER AI
0993..09A8 ; Lo # [22] BENGALI LETTER O..BENGALI LETTER NA
09AA..09B0 ; Lo # [7] BENGALI LETTER PA..BENGALI LETTER RA
09B2 ; Lo # BENGALI LETTER LA
09B6..09B9 ; Lo # [4] BENGALI LETTER SHA..BENGALI LETTER HA
09DC..09DD ; Lo # [2] BENGALI LETTER RRA..BENGALI LETTER RHA
09DF..09E1 ; Lo # [3] BENGALI LETTER YYA..BENGALI LETTER VOCALIC LL
09F0..09F1 ; Lo # [2] BENGALI LETTER RA WITH MIDDLE DIAGONAL..BENGALI LETTER RA WITH LOWER DIAGONAL
0A05..0A0A ; Lo # [6] GURMUKHI LETTER A..GURMUKHI LETTER UU
0A0F..0A10 ; Lo # [2] GURMUKHI LETTER EE..GURMUKHI LETTER AI
0A13..0A28 ; Lo # [22] GURMUKHI LETTER OO..GURMUKHI LETTER NA
0A2A..0A30 ; Lo # [7] GURMUKHI LETTER PA..GURMUKHI LETTER RA
0A32..0A33 ; Lo # [2] GURMUKHI LETTER LA..GURMUKHI LETTER LLA
0A35..0A36 ; Lo # [2] GURMUKHI LETTER VA..GURMUKHI LETTER SHA
0A38..0A39 ; Lo # [2] GURMUKHI LETTER SA..GURMUKHI LETTER HA
0A59..0A5C ; Lo # [4] GURMUKHI LETTER KHHA..GURMUKHI LETTER RRA
0A5E ; Lo # GURMUKHI LETTER FA
0A72..0A74 ; Lo # [3] GURMUKHI IRI..GURMUKHI EK ONKAR
0A85..0A8B ; Lo # [7] GUJARATI LETTER A..GUJARATI LETTER VOCALIC R
0A8D ; Lo # GUJARATI VOWEL CANDRA E
0A8F..0A91 ; Lo # [3] GUJARATI LETTER E..GUJARATI VOWEL CANDRA O
0A93..0AA8 ; Lo # [22] GUJARATI LETTER O..GUJARATI LETTER NA
0AAA..0AB0 ; Lo # [7] GUJARATI LETTER PA..GUJARATI LETTER RA
0AB2..0AB3 ; Lo # [2] GUJARATI LETTER LA..GUJARATI LETTER LLA
0AB5..0AB9 ; Lo # [5] GUJARATI LETTER VA..GUJARATI LETTER HA
0ABD ; Lo # GUJARATI SIGN AVAGRAHA
0AD0 ; Lo # GUJARATI OM
0AE0 ; Lo # GUJARATI LETTER VOCALIC RR
0B05..0B0C ; Lo # [8] ORIYA LETTER A..ORIYA LETTER VOCALIC L
0B0F..0B10 ; Lo # [2] ORIYA LETTER E..ORIYA LETTER AI
0B13..0B28 ; Lo # [22] ORIYA LETTER O..ORIYA LETTER NA
0B2A..0B30 ; Lo # [7] ORIYA LETTER PA..ORIYA LETTER RA
0B32..0B33 ; Lo # [2] ORIYA LETTER LA..ORIYA LETTER LLA
0B36..0B39 ; Lo # [4] ORIYA LETTER SHA..ORIYA LETTER HA
0B3D ; Lo # ORIYA SIGN AVAGRAHA
0B5C..0B5D ; Lo # [2] ORIYA LETTER RRA..ORIYA LETTER RHA
0B5F..0B61 ; Lo # [3] ORIYA LETTER YYA..ORIYA LETTER VOCALIC LL
0B85..0B8A ; Lo # [6] TAMIL LETTER A..TAMIL LETTER UU
0B8E..0B90 ; Lo # [3] TAMIL LETTER E..TAMIL LETTER AI

0B92..0B95 ; Lo # [4] TAMIL LETTER O..TAMIL LETTER KA
0B99..0B9A ; Lo # [2] TAMIL LETTER NGA..TAMIL LETTER CA
0B9C ; Lo # TAMIL LETTER JA
0B9E..0B9F ; Lo # [2] TAMIL LETTER NYA..TAMIL LETTER TTA
0BA3..0BA4 ; Lo # [2] TAMIL LETTER NNA..TAMIL LETTER TA
0BA8..0BAA ; Lo # [3] TAMIL LETTER NA..TAMIL LETTER PA
0BAE..0BB5 ; Lo # [8] TAMIL LETTER MA..TAMIL LETTER VA
0BB7..0BB9 ; Lo # [3] TAMIL LETTER SSA..TAMIL LETTER HA
0C05..0C0C ; Lo # [8] TELUGU LETTER A..TELUGU LETTER VOCALIC L
0C0E..0C10 ; Lo # [3] TELUGU LETTER E..TELUGU LETTER AI
0C12..0C28 ; Lo # [23] TELUGU LETTER O..TELUGU LETTER NA
0C2A..0C33 ; Lo # [10] TELUGU LETTER PA..TELUGU LETTER LLA
0C35..0C39 ; Lo # [5] TELUGU LETTER VA..TELUGU LETTER HA
0C60..0C61 ; Lo # [2] TELUGU LETTER VOCALIC RR..TELUGU LETTER VOCALIC LL
0C85..0C8C ; Lo # [8] KANNADA LETTER A..KANNADA LETTER VOCALIC L
0C8E..0C90 ; Lo # [3] KANNADA LETTER E..KANNADA LETTER AI
0C92..0CA8 ; Lo # [23] KANNADA LETTER O..KANNADA LETTER NA
0CAA..0CB3 ; Lo # [10] KANNADA LETTER PA..KANNADA LETTER LLA
0CB5..0CB9 ; Lo # [5] KANNADA LETTER VA..KANNADA LETTER HA
0CDE ; Lo # KANNADA LETTER FA
0CE0..0CE1 ; Lo # [2] KANNADA LETTER VOCALIC RR..KANNADA LETTER VOCALIC LL
0D05..0D0C ; Lo # [8] MALAYALAM LETTER A..MALAYALAM LETTER VOCALIC L
0D0E..0D10 ; Lo # [3] MALAYALAM LETTER E..MALAYALAM LETTER AI
0D12..0D28 ; Lo # [23] MALAYALAM LETTER O..MALAYALAM LETTER NA
0D2A..0D39 ; Lo # [16] MALAYALAM LETTER PA..MALAYALAM LETTER HA
0D60..0D61 ; Lo # [2] MALAYALAM LETTER VOCALIC RR..MALAYALAM LETTER VOCALIC LL
0D85..0D96 ; Lo # [18] SINHALA LETTER AYANNA..SINHALA LETTER AUYANNA
0D9A..0DB1 ; Lo # [24] SINHALA LETTER ALPAPRAANA KAYANNA..SINHALA LETTER DANTAJA NAYANNA
0DB3..0DBB ; Lo # [9] SINHALA LETTER SANYAKA DAYANNA..SINHALA LETTER RAYANNA
0DBD ; Lo # SINHALA LETTER DANTAJA LAYANNA
0DC0..0DC6 ; Lo # [7] SINHALA LETTER VAYANNA..SINHALA LETTER FAYANNA
0E01..0E30 ; Lo # [48] THAI CHARACTER KO KAI..THAI CHARACTER SARA A
0E32..0E33 ; Lo # [2] THAI CHARACTER SARA AA..THAI CHARACTER SARA AM
0E40..0E45 ; Lo # [6] THAI CHARACTER SARA E..THAI CHARACTER LAKKHANGYAO
0E81..0E82 ; Lo # [2] LAO LETTER KO..LAO LETTER KHO SUNG
0E84 ; Lo # LAO LETTER KHO TAM
0E87..0E88 ; Lo # [2] LAO LETTER NGO..LAO LETTER CO
0E8A ; Lo # LAO LETTER SO TAM
0E8D ; Lo # LAO LETTER NYO
0E94..0E97 ; Lo # [4] LAO LETTER DO..LAO LETTER THO TAM
0E99..0E9F ; Lo # [7] LAO LETTER NO..LAO LETTER FO SUNG
0EA1..0EA3 ; Lo # [3] LAO LETTER MO..LAO LETTER LO LING

0EA5 ; Lo # LAO LETTER LO LOOT
0EA7 ; Lo # LAO LETTER WO
0EAA..0EAB ; Lo # [2] LAO LETTER SO SUNG..LAO LETTER HO SUNG
0EAD..0EB0 ; Lo # [4] LAO LETTER O..LAO VOWEL SIGN A
0EB2..0EB3 ; Lo # [2] LAO VOWEL SIGN AA..LAO VOWEL SIGN AM
0EBD ; Lo # LAO SEMIVOWEL SIGN NYO
0EC0..0EC4 ; Lo # [5] LAO VOWEL SIGN E..LAO VOWEL SIGN AI
0EDC..0EDD ; Lo # [2] LAO HO NO..LAO HO MO
0F00 ; Lo # TIBETAN SYLLABLE OM
0F40..0F47 ; Lo # [8] TIBETAN LETTER KA..TIBETAN LETTER JA
0F49..0F6A ; Lo # [34] TIBETAN LETTER NYA..TIBETAN LETTER FIXED-FORM RA
0F88..0F8B ; Lo # [4] TIBETAN SIGN LCE TSA CAN..TIBETAN SIGN GRU MED RGYINGS
1000..1021 ; Lo # [34] MYANMAR LETTER KA..MYANMAR LETTER A
1023..1027 ; Lo # [5] MYANMAR LETTER I..MYANMAR LETTER E
1029..102A ; Lo # [2] MYANMAR LETTER O..MYANMAR LETTER AU
1050..1055 ; Lo # [6] MYANMAR LETTER SHA..MYANMAR LETTER VOCALIC LL
10D0..10F6 ; Lo # [39] GEORGIAN LETTER AN..GEORGIAN LETTER FI
1100..1159 ; Lo # [90] HANGUL CHOSEONG KIYEOK..HANGUL CHOSEONG YEORINHIEUH
115F..11A2 ; Lo # [68] HANGUL CHOSEONG FILLER..HANGUL JUNGSEONG SSANGARAEA
11A8..11F9 ; Lo # [82] HANGUL JONGSEONG KIYEOK..HANGUL JONGSEONG YEORINHIEUH
1200..1206 ; Lo # [7] ETHIOPIC SYLLABLE HA..ETHIOPIC SYLLABLE HO
1208..1246 ; Lo # [63] ETHIOPIC SYLLABLE LA..ETHIOPIC SYLLABLE QO
1248 ; Lo # ETHIOPIC SYLLABLE QWA
124A..124D ; Lo # [4] ETHIOPIC SYLLABLE QWI..ETHIOPIC SYLLABLE QWE
1250..1256 ; Lo # [7] ETHIOPIC SYLLABLE QHA..ETHIOPIC SYLLABLE QHO
1258 ; Lo # ETHIOPIC SYLLABLE QHWA
125A..125D ; Lo # [4] ETHIOPIC SYLLABLE QHWI..ETHIOPIC SYLLABLE QHWE
1260..1286 ; Lo # [39] ETHIOPIC SYLLABLE BA..ETHIOPIC SYLLABLE XO
1288 ; Lo # ETHIOPIC SYLLABLE XWA
128A..128D ; Lo # [4] ETHIOPIC SYLLABLE XWI..ETHIOPIC SYLLABLE XWE
1290..12AE ; Lo # [31] ETHIOPIC SYLLABLE NA..ETHIOPIC SYLLABLE KO
12B0 ; Lo # ETHIOPIC SYLLABLE KWA
12B2..12B5 ; Lo # [4] ETHIOPIC SYLLABLE KWI..ETHIOPIC SYLLABLE KWE
12B8..12BE ; Lo # [7] ETHIOPIC SYLLABLE KXA..ETHIOPIC SYLLABLE KXO
12C0 ; Lo # ETHIOPIC SYLLABLE KXWA
12C2..12C5 ; Lo # [4] ETHIOPIC SYLLABLE KXWI..ETHIOPIC SYLLABLE KXWE
12C8..12CE ; Lo # [7] ETHIOPIC SYLLABLE WA..ETHIOPIC SYLLABLE WO
12D0..12D6 ; Lo # [7] ETHIOPIC SYLLABLE PHARYNGEAL A..ETHIOPIC SYLLABLE PHARYNGEAL O
12D8..12EE ; Lo # [23] ETHIOPIC SYLLABLE ZA..ETHIOPIC SYLLABLE YO
12F0..130E ; Lo # [31] ETHIOPIC SYLLABLE DA..ETHIOPIC SYLLABLE GO
1310 ; Lo # ETHIOPIC SYLLABLE GWA
1312..1315 ; Lo # [4] ETHIOPIC SYLLABLE GWI..ETHIOPIC SYLLABLE GWE

1318..131E ; Lo # [7] ETHIOPIC SYLLABLE GGA..ETHIOPIC SYLLABLE GGO
1320..1346 ; Lo # [39] ETHIOPIC SYLLABLE THA..ETHIOPIC SYLLABLE TZO
1348..135A ; Lo # [19] ETHIOPIC SYLLABLE FA..ETHIOPIC SYLLABLE FYA
13A0..13F4 ; Lo # [85] CHEROKEE LETTER A..CHEROKEE LETTER YV
1401..166C ; Lo # [620] CANADIAN SYLLABICS E..CANADIAN SYLLABICS CARRIER TTSA
166F..1676 ; Lo # [8] CANADIAN SYLLABICS QAI..CANADIAN SYLLABICS NNGAA
1681..169A ; Lo # [26] OGHAM LETTER BEITH..OGHAM LETTER PEITH
16A0..16EA ; Lo # [75] RUNIC LETTER FEHU FEOH FE F..RUNIC LETTER X
1780..17B3 ; Lo # [52] KHMER LETTER KA..KHMER INDEPENDENT VOWEL QAU
1820..1842 ; Lo # [35] MONGOLIAN LETTER A..MONGOLIAN LETTER CHI
1844..1877 ; Lo # [52] MONGOLIAN LETTER TODO E..MONGOLIAN LETTER MANCHU ZHA
1880..18A8 ; Lo # [41] MONGOLIAN LETTER ALI GALI ANUSVARA ONE..MONGOLIAN LETTER MANCHU ALI GALI BHA
2135..2138 ; Lo # [4] ALEF SYMBOL..DALET SYMBOL
3006 ; Lo # IDEOGRAPHIC CLOSING MARK
3041..3094 ; Lo # [84] HIRAGANA LETTER SMALL A..HIRAGANA LETTER VU
30A1..30FA ; Lo # [90] KATAKANA LETTER SMALL A..KATAKANA LETTER VO
3105..312C ; Lo # [40] BOPOMOFO LETTER B..BOPOMOFO LETTER GN
3131..318E ; Lo # [94] HANGUL LETTER KIYEOK..HANGUL LETTER ARAEAE
31A0..31B7 ; Lo # [24] BOPOMOFO LETTER BU..BOPOMOFO FINAL LETTER H
3400..4DB5 ; Lo # [6582] CJK UNIFIED IDEOGRAPH-3400..CJK UNIFIED IDEOGRAPH-4DB5
4E00..9FA5 ; Lo # [20902] CJK UNIFIED IDEOGRAPH-4E00..CJK UNIFIED IDEOGRAPH-9FA5
A000..A48C ; Lo # [1165] YI SYLLABLE IT..YI SYLLABLE YYR
AC00..D7A3 ; Lo # [11172] HANGUL SYLLABLE GA..HANGUL SYLLABLE HIH
F900..FA2D ; Lo # [302] CJK COMPATIBILITY IDEOGRAPH-F900..CJK COMPATIBILITY IDEOGRAPH-FA2D
FB1D ; Lo # HEBREW LETTER YOD WITH HIRIQ
FB1F..FB28 ; Lo # [10] HEBREW LIGATURE YIDDISH YOD YOD PATAH..HEBREW LETTER WIDE TAV
FB2A..FB36 ; Lo # [13] HEBREW LETTER SHIN WITH SHIN DOT..HEBREW LETTER ZAYIN WITH DAGESH
FB38..FB3C ; Lo # [5] HEBREW LETTER TET WITH DAGESH..HEBREW LETTER LAMED WITH DAGESH
FB3E ; Lo # HEBREW LETTER MEM WITH DAGESH
FB40..FB41 ; Lo # [2] HEBREW LETTER NUN WITH DAGESH..HEBREW LETTER SAMEKH WITH DAGESH
FB43..FB44 ; Lo # [2] HEBREW LETTER FINAL PE WITH DAGESH..HEBREW LETTER PE WITH DAGESH
FB46..FBB1 ; Lo # [108] HEBREW LETTER TSADI WITH DAGESH..ARABIC LETTER YEH BARREE WITH HAMZA ABOVE FINAL FORM
FBD3..FD3D ; Lo # [363] ARABIC LETTER NG ISOLATED FORM..ARABIC LIGATURE ALEF WITH FATHATAN ISOLATED FORM

FD50..FD8F ; Lo # [64] ARABIC LIGATURE TEH WITH JEEM WITH MEEM INITIAL FORM..ARABIC LIGATURE MEEM WITH KHAH WITH MEEM INITIAL FORM
FD92..FDC7 ; Lo # [54] ARABIC LIGATURE MEEM WITH JEEM WITH KHAH INITIAL FORM..ARABIC LIGATURE NOON WITH JEEM WITH YEH FINAL FORM
FDF0..FDFB ; Lo # [12] ARABIC LIGATURE SALLA USED AS KORANIC STOP SIGN ISOLATED FORM..ARABIC LIGATURE JALLAJALALOUHOU
FE70..FE72 ; Lo # [3] ARABIC FATHATAN ISOLATED FORM..ARABIC DAMMATAN ISOLATED FORM
FE74 ; Lo # ARABIC KASRATAN ISOLATED FORM
FE76..FEFC ; Lo # [135] ARABIC FATHA ISOLATED FORM..ARABIC LIGATURE LAM WITH ALEF FINAL FORM
FF66..FF6F ; Lo # [10] HALFWIDTH KATAKANA LETTER WO..HALFWIDTH KATAKANA LETTER SMALL TU
FF71..FF9D ; Lo # [45] HALFWIDTH KATAKANA LETTER A..HALFWIDTH KATAKANA LETTER N
FFA0..FFBE ; Lo # [31] HALFWIDTH HANGUL FILLER..HALFWIDTH HANGUL LETTER HIEUH
FFC2..FFC7 ; Lo # [6] HALFWIDTH HANGUL LETTER A..HALFWIDTH HANGUL LETTER E
FFCA..FFCF ; Lo # [6] HALFWIDTH HANGUL LETTER YEO..HALFWIDTH HANGUL LETTER OE
FFD2..FFD7 ; Lo # [6] HALFWIDTH HANGUL LETTER YO..HALFWIDTH HANGUL LETTER YU
FFDA..FFDC ; Lo # [3] HALFWIDTH HANGUL LETTER EU..HALFWIDTH HANGUL LETTER I
10300..1031E ; Lo # [31] OLD ITALIC LETTER A..OLD ITALIC LETTER UU
10330..10349 ; Lo # [26] GOTHIC LETTER AHSA..GOTHIC LETTER OTHAL
20000..2A6D6 ; Lo # [42711] CJK UNIFIED IDEOGRAPH-20000..CJK UNIFIED IDEOGRAPH-2A6D6
2F800..2FA1D ; Lo # [542] CJK COMPATIBILITY IDEOGRAPH-2F800..CJK COMPATIBILITY IDEOGRAPH-2FA1D

Total code points: 87186

==

0300..034E ; Mn # [79] COMBINING GRAVE ACCENT..COMBINING UPWARDS ARROW BELOW
0360..0362 ; Mn # [3] COMBINING DOUBLE TILDE..COMBINING DOUBLE RIGHTWARDS ARROW BELOW
0483..0486 ; Mn # [4] COMBINING CYRILLIC TITLO..COMBINING CYRILLIC PSILI PNEUMATA
0591..05A1 ; Mn # [17] HEBREW ACCENT ETNAHTA..HEBREW ACCENT PAZER
05A3..05B9 ; Mn # [23] HEBREW ACCENT MUNAH..HEBREW POINT HOLAM
05BB..05BD ; Mn # [3] HEBREW POINT QUBUTS..HEBREW POINT METEG
05BF ; Mn # HEBREW POINT RAFE

05C1..05C2 ; Mn # [2] HEBREW POINT SHIN DOT..HEBREW POINT SIN DOT
05C4 ; Mn # HEBREW MARK UPPER DOT
064B..0655 ; Mn # [11] ARABIC FATHATAN..ARABIC HAMZA BELOW
0670 ; Mn # ARABIC LETTER SUPERSCRIPT ALEF
06D6..06DC ; Mn # [7] ARABIC SMALL HIGH LIGATURE SAD WITH LAM WITH ALEF MAKSURA..ARABIC SMALL HIGH SEEN
06DF..06E4 ; Mn # [6] ARABIC SMALL HIGH ROUNDED ZERO..ARABIC SMALL HIGH MADDA
06E7..06E8 ; Mn # [2] ARABIC SMALL HIGH YEH..ARABIC SMALL HIGH NOON
06EA..06ED ; Mn # [4] ARABIC EMPTY CENTRE LOW STOP..ARABIC SMALL LOW MEEM
0711 ; Mn # SYRIAC LETTER SUPERSCRIPT ALAPH
0730..074A ; Mn # [27] SYRIAC PTHAHA ABOVE..SYRIAC BARREKH
07A6..07B0 ; Mn # [11] THAANA ABAFILI..THAANA SUKUN
0901..0902 ; Mn # [2] DEVANAGARI SIGN CANDRABINDU..DEVANAGARI SIGN ANUSVARA
093C ; Mn # DEVANAGARI SIGN NUKTA
0941..0948 ; Mn # [8] DEVANAGARI VOWEL SIGN U..DEVANAGARI VOWEL SIGN AI
094D ; Mn # DEVANAGARI SIGN VIRAMA
0951..0954 ; Mn # [4] DEVANAGARI STRESS SIGN UDATTA..DEVANAGARI ACUTE ACCENT
0962..0963 ; Mn # [2] DEVANAGARI VOWEL SIGN VOCALIC L..DEVANAGARI VOWEL SIGN VOCALIC LL
0981 ; Mn # BENGALI SIGN CANDRABINDU
09BC ; Mn # BENGALI SIGN NUKTA
09C1..09C4 ; Mn # [4] BENGALI VOWEL SIGN U..BENGALI VOWEL SIGN VOCALIC RR
09CD ; Mn # BENGALI SIGN VIRAMA
09E2..09E3 ; Mn # [2] BENGALI VOWEL SIGN VOCALIC L..BENGALI VOWEL SIGN VOCALIC LL
0A02 ; Mn # GURMUKHI SIGN BINDI
0A3C ; Mn # GURMUKHI SIGN NUKTA
0A41..0A42 ; Mn # [2] GURMUKHI VOWEL SIGN U..GURMUKHI VOWEL SIGN UU
0A47..0A48 ; Mn # [2] GURMUKHI VOWEL SIGN EE..GURMUKHI VOWEL SIGN AI
0A4B..0A4D ; Mn # [3] GURMUKHI VOWEL SIGN OO..GURMUKHI SIGN VIRAMA
0A70..0A71 ; Mn # [2] GURMUKHI TIPPI..GURMUKHI ADDAK
0A81..0A82 ; Mn # [2] GUJARATI SIGN CANDRABINDU..GUJARATI SIGN ANUSVARA
0ABC ; Mn # GUJARATI SIGN NUKTA
0AC1..0AC5 ; Mn # [5] GUJARATI VOWEL SIGN U..GUJARATI VOWEL SIGN CANDRA E
0AC7..0AC8 ; Mn # [2] GUJARATI VOWEL SIGN E..GUJARATI VOWEL SIGN AI
0ACD ; Mn # GUJARATI SIGN VIRAMA
0B01 ; Mn # ORIYA SIGN CANDRABINDU
0B3C ; Mn # ORIYA SIGN NUKTA
0B3F ; Mn # ORIYA VOWEL SIGN I
0B41..0B43 ; Mn # [3] ORIYA VOWEL SIGN U..ORIYA VOWEL SIGN VOCALIC R
0B4D ; Mn # ORIYA SIGN VIRAMA
0B56 ; Mn # ORIYA AI LENGTH MARK

0B82 ; Mn # TAMIL SIGN ANUSVARA
0BC0 ; Mn # TAMIL VOWEL SIGN II
0BCD ; Mn # TAMIL SIGN VIRAMA
0C3E..0C40 ; Mn # [3] TELUGU VOWEL SIGN AA..TELUGU VOWEL SIGN II
0C46..0C48 ; Mn # [3] TELUGU VOWEL SIGN E..TELUGU VOWEL SIGN AI
0C4A..0C4D ; Mn # [4] TELUGU VOWEL SIGN O..TELUGU SIGN VIRAMA
0C55..0C56 ; Mn # [2] TELUGU LENGTH MARK..TELUGU AI LENGTH MARK
0CBF ; Mn # KANNADA VOWEL SIGN I
0CC6 ; Mn # KANNADA VOWEL SIGN E
0CCC..0CCD ; Mn # [2] KANNADA VOWEL SIGN AU..KANNADA SIGN VIRAMA
0D41..0D43 ; Mn # [3] MALAYALAM VOWEL SIGN U..MALAYALAM VOWEL SIGN VOCALIC R
0D4D ; Mn # MALAYALAM SIGN VIRAMA
0DCA ; Mn # SINHALA SIGN AL-LAKUNA
0DD2..0DD4 ; Mn # [3] SINHALA VOWEL SIGN KETTI IS-PILLA..SINHALA VOWEL SIGN KETTI PAA-PILLA
0DD6 ; Mn # SINHALA VOWEL SIGN DIGA PAA-PILLA
0E31 ; Mn # THAI CHARACTER MAI HAN-AKAT
0E34..0E3A ; Mn # [7] THAI CHARACTER SARA I..THAI CHARACTER PHINTHU
0E47..0E4E ; Mn # [8] THAI CHARACTER MAITAIKHU..THAI CHARACTER YAMAKKAN
0EB1 ; Mn # LAO VOWEL SIGN MAI KAN
0EB4..0EB9 ; Mn # [6] LAO VOWEL SIGN I..LAO VOWEL SIGN UU
0EBB..0EBC ; Mn # [2] LAO VOWEL SIGN MAI KON..LAO SEMIVOWEL SIGN LO
0EC8..0ECD ; Mn # [6] LAO TONE MAI EK..LAO NIGGAHITA
0F18..0F19 ; Mn # [2] TIBETAN ASTROLOGICAL SIGN -KHYUD PA..TIBETAN ASTROLOGICAL SIGN SDONG TSHUGS
0F35 ; Mn # TIBETAN MARK NGAS BZUNG NYI ZLA
0F37 ; Mn # TIBETAN MARK NGAS BZUNG SGOR RTAGS
0F39 ; Mn # TIBETAN MARK TSA -PHRU
0F71..0F7E ; Mn # [14] TIBETAN VOWEL SIGN AA..TIBETAN SIGN RJES SU NGA RO
0F80..0F84 ; Mn # [5] TIBETAN VOWEL SIGN REVERSED I..TIBETAN MARK HALANTA
0F86..0F87 ; Mn # [2] TIBETAN SIGN LCI RTAGS..TIBETAN SIGN YANG RTAGS
0F90..0F97 ; Mn # [8] TIBETAN SUBJOINED LETTER KA..TIBETAN SUBJOINED LETTER JA
0F99..0FBC ; Mn # [36] TIBETAN SUBJOINED LETTER NYA..TIBETAN SUBJOINED LETTER FIXED-FORM RA
0FC6 ; Mn # TIBETAN SYMBOL PADMA GDAN
102D..1030 ; Mn # [4] MYANMAR VOWEL SIGN I..MYANMAR VOWEL SIGN UU
1032 ; Mn # MYANMAR VOWEL SIGN AI
1036..1037 ; Mn # [2] MYANMAR SIGN ANUSVARA..MYANMAR SIGN DOT BELOW
1039 ; Mn # MYANMAR SIGN VIRAMA
1058..1059 ; Mn # [2] MYANMAR VOWEL SIGN VOCALIC L..MYANMAR VOWEL SIGN VOCALIC LL
17B7..17BD ; Mn # [7] KHMER VOWEL SIGN I..KHMER VOWEL SIGN UA
17C6 ; Mn # KHMER SIGN NIKAHIT
17C9..17D3 ; Mn # [11] KHMER SIGN MUUSIKATOAN..KHMER SIGN BATHAMASAT

18A9 ; Mn # MONGOLIAN LETTER ALI GALI DAGALGA
20D0..20DC ; Mn # [13] COMBINING LEFT HARPOON ABOVE..COMBINING FOUR DOTS ABOVE
20E1 ; Mn # COMBINING LEFT RIGHT ARROW ABOVE
302A..302F ; Mn # [6] IDEOGRAPHIC LEVEL TONE MARK..HANGUL DOUBLE DOT TONE MARK
3099..309A ; Mn # [2] COMBINING KATAKANA-HIRAGANA VOICED SOUND MARK.. COMBINING KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK
FB1E ; Mn # HEBREW POINT JUDEO-SPANISH VARIKA
FE20..FE23 ; Mn # [4] COMBINING LIGATURE LEFT HALF..COMBINING DOUBLE TILDE RIGHT HALF
1D167..1D169 ; Mn # [3] MUSICAL SYMBOL COMBINING TREMOLO-1..MUSICAL SYMBOL COMBINING TREMOLO-3
1D17B..1D182 ; Mn # [8] MUSICAL SYMBOL COMBINING ACCENT..MUSICAL SYMBOL COMBINING LOURE
1D185..1D18B ; Mn # [7] MUSICAL SYMBOL COMBINING DOIT..MUSICAL SYMBOL COMBINING TRIPLE TONGUE
1D1AA..1D1AD ; Mn # [4] MUSICAL SYMBOL COMBINING DOWN BOW..MUSICAL SYMBOL COMBINING SNAP PIZZICATO

Total code points: 469

==

0488..0489 ; Me # [2] COMBINING CYRILLIC HUNDRED THOUSANDS SIGN.. COMBINING CYRILLIC MILLIONS SIGN
06DD..06DE ; Me # [2] ARABIC END OF AYAH..ARABIC START OF RUB EL HIZB
20DD..20E0 ; Me # [4] COMBINING ENCLOSING CIRCLE..COMBINING ENCLOSING CIRCLE BACKSLASH
20E2..20E3 ; Me # [2] COMBINING ENCLOSING SCREEN..COMBINING ENCLOSING KEYCAP

Total code points: 10

==

0903 ; Mc # DEVANAGARI SIGN VISARGA
093E..0940 ; Mc # [3] DEVANAGARI VOWEL SIGN AA..DEVANAGARI VOWEL SIGN II
0949..094C ; Mc # [4] DEVANAGARI VOWEL SIGN CANDRA O..DEVANAGARI VOWEL SIGN AU
0982..0983 ; Mc # [2] BENGALI SIGN ANUSVARA..BENGALI SIGN VISARGA
09BE..09C0 ; Mc # [3] BENGALI VOWEL SIGN AA..BENGALI VOWEL SIGN II
09C7..09C8 ; Mc # [2] BENGALI VOWEL SIGN E..BENGALI VOWEL SIGN AI
09CB..09CC ; Mc # [2] BENGALI VOWEL SIGN O..BENGALI VOWEL SIGN AU
09D7 ; Mc # BENGALI AU LENGTH MARK
0A3E..0A40 ; Mc # [3] GURMUKHI VOWEL SIGN AA..GURMUKHI VOWEL SIGN II

0A83 ; Mc # GUJARATI SIGN VISARGA
0ABE..0AC0 ; Mc # [3] GUJARATI VOWEL SIGN AA..GUJARATI VOWEL SIGN II
0AC9 ; Mc # GUJARATI VOWEL SIGN CANDRA O
0ACB..0ACC ; Mc # [2] GUJARATI VOWEL SIGN O..GUJARATI VOWEL SIGN AU
0B02..0B03 ; Mc # [2] ORIYA SIGN ANUSVARA..ORIYA SIGN VISARGA
0B3E ; Mc # ORIYA VOWEL SIGN AA
0B40 ; Mc # ORIYA VOWEL SIGN II
0B47..0B48 ; Mc # [2] ORIYA VOWEL SIGN E..ORIYA VOWEL SIGN AI
0B4B..0B4C ; Mc # [2] ORIYA VOWEL SIGN O..ORIYA VOWEL SIGN AU
0B57 ; Mc # ORIYA AU LENGTH MARK
0B83 ; Mc # TAMIL SIGN VISARGA
0BBE..0BBF ; Mc # [2] TAMIL VOWEL SIGN AA..TAMIL VOWEL SIGN I
0BC1..0BC2 ; Mc # [2] TAMIL VOWEL SIGN U..TAMIL VOWEL SIGN UU
0BC6..0BC8 ; Mc # [3] TAMIL VOWEL SIGN E..TAMIL VOWEL SIGN AI
0BCA..0BCC ; Mc # [3] TAMIL VOWEL SIGN O..TAMIL VOWEL SIGN AU
0BD7 ; Mc # TAMIL AU LENGTH MARK
0C01..0C03 ; Mc # [3] TELUGU SIGN CANDRABINDU..TELUGU SIGN VISARGA
0C41..0C44 ; Mc # [4] TELUGU VOWEL SIGN U..TELUGU VOWEL SIGN VOCALIC RR
0C82..0C83 ; Mc # [2] KANNADA SIGN ANUSVARA..KANNADA SIGN VISARGA
0CBE ; Mc # KANNADA VOWEL SIGN AA
0CC0..0CC4 ; Mc # [5] KANNADA VOWEL SIGN II..KANNADA VOWEL SIGN VOCALIC RR
0CC7..0CC8 ; Mc # [2] KANNADA VOWEL SIGN EE..KANNADA VOWEL SIGN AI
0CCA..0CCB ; Mc # [2] KANNADA VOWEL SIGN O..KANNADA VOWEL SIGN OO
0CD5..0CD6 ; Mc # [2] KANNADA LENGTH MARK..KANNADA AI LENGTH MARK
0D02..0D03 ; Mc # [2] MALAYALAM SIGN ANUSVARA..MALAYALAM SIGN VISARGA
0D3E..0D40 ; Mc # [3] MALAYALAM VOWEL SIGN AA..MALAYALAM VOWEL SIGN II
0D46..0D48 ; Mc # [3] MALAYALAM VOWEL SIGN E..MALAYALAM VOWEL SIGN AI
0D4A..0D4C ; Mc # [3] MALAYALAM VOWEL SIGN O..MALAYALAM VOWEL SIGN AU
0D57 ; Mc # MALAYALAM AU LENGTH MARK
0D82..0D83 ; Mc # [2] SINHALA SIGN ANUSVARAYA..SINHALA SIGN VISARGAYA
0DCF..0DD1 ; Mc # [3] SINHALA VOWEL SIGN AELA-PILLA..SINHALA VOWEL SIGN DIGA AEDA-PILLA
0DD8..0DDF ; Mc # [8] SINHALA VOWEL SIGN GAETTA-PILLA..SINHALA VOWEL SIGN GAYANUKITTA
0DF2..0DF3 ; Mc # [2] SINHALA VOWEL SIGN DIGA GAETTA-PILLA..SINHALA VOWEL SIGN DIGA GAYANUKITTA
0F3E..0F3F ; Mc # [2] TIBETAN SIGN YAR TSHES..TIBETAN SIGN MAR TSHES
0F7F ; Mc # TIBETAN SIGN RNAM BCAD
102C ; Mc # MYANMAR VOWEL SIGN AA
1031 ; Mc # MYANMAR VOWEL SIGN E
1038 ; Mc # MYANMAR SIGN VISARGA
1056..1057 ; Mc # [2] MYANMAR VOWEL SIGN VOCALIC R..MYANMAR VOWEL SIGN VOCALIC RR
17B4..17B6 ; Mc # [3] KHMER VOWEL INHERENT AQ..KHMER VOWEL SIGN AA
17BE..17C5 ; Mc # [8] KHMER VOWEL SIGN OE..KHMER VOWEL SIGN AU

17C7..17C8 ; Mc # [2] KHMER SIGN REAHMUK..KHMER SIGN YUUKALEAPINTU
1D165..1D166 ; Mc # [2] MUSICAL SYMBOL COMBINING STEM..MUSICAL SYMBOL COMBINING SPRECHGESANG STEM
1D16D..1D172 ; Mc # [6] MUSICAL SYMBOL COMBINING AUGMENTATION DOT..MUSICAL SYMBOL COMBINING FLAG-5

Total code points: 126

==

0030..0039 ; Nd # [10] DIGIT ZERO..DIGIT NINE
0660..0669 ; Nd # [10] ARABIC-INDIC DIGIT ZERO..ARABIC-INDIC DIGIT NINE
06F0..06F9 ; Nd # [10] EXTENDED ARABIC-INDIC DIGIT ZERO..EXTENDED ARABIC-INDIC DIGIT NINE
0966..096F ; Nd # [10] DEVANAGARI DIGIT ZERO..DEVANAGARI DIGIT NINE
09E6..09EF ; Nd # [10] BENGALI DIGIT ZERO..BENGALI DIGIT NINE
0A66..0A6F ; Nd # [10] GURMUKHI DIGIT ZERO..GURMUKHI DIGIT NINE
0AE6..0AEF ; Nd # [10] GUJARATI DIGIT ZERO..GUJARATI DIGIT NINE
0B66..0B6F ; Nd # [10] ORIYA DIGIT ZERO..ORIYA DIGIT NINE
0BE7..0BEF ; Nd # [9] TAMIL DIGIT ONE..TAMIL DIGIT NINE
0C66..0C6F ; Nd # [10] TELUGU DIGIT ZERO..TELUGU DIGIT NINE
0CE6..0CEF ; Nd # [10] KANNADA DIGIT ZERO..KANNADA DIGIT NINE
0D66..0D6F ; Nd # [10] MALAYALAM DIGIT ZERO..MALAYALAM DIGIT NINE
0E50..0E59 ; Nd # [10] THAI DIGIT ZERO..THAI DIGIT NINE
0ED0..0ED9 ; Nd # [10] LAO DIGIT ZERO..LAO DIGIT NINE
0F20..0F29 ; Nd # [10] TIBETAN DIGIT ZERO..TIBETAN DIGIT NINE
1040..1049 ; Nd # [10] MYANMAR DIGIT ZERO..MYANMAR DIGIT NINE
1369..1371 ; Nd # [9] ETHIOPIC DIGIT ONE..ETHIOPIC DIGIT NINE
17E0..17E9 ; Nd # [10] KHMER DIGIT ZERO..KHMER DIGIT NINE
1810..1819 ; Nd # [10] MONGOLIAN DIGIT ZERO..MONGOLIAN DIGIT NINE
FF10..FF19 ; Nd # [10] FULLWIDTH DIGIT ZERO..FULLWIDTH DIGIT NINE
1D7CE..1D7FF ; Nd # [50] MATHEMATICAL BOLD DIGIT ZERO..MATHEMATICAL MONOSPACE DIGIT NINE

Total code points: 248

==

16EE..16F0 ; Nl # [3] RUNIC ARLAUG SYMBOL..RUNIC BELGTHOR SYMBOL
2160..2183 ; Nl # [36] ROMAN NUMERAL ONE..ROMAN NUMERAL REVERSED ONE HUNDRED
3007 ; Nl # IDEOGRAPHIC NUMBER ZERO
3021..3029 ; Nl # [9] HANGZHOU NUMERAL ONE..HANGZHOU NUMERAL NINE
3038..303A ; Nl # [3] HANGZHOU NUMERAL TEN..HANGZHOU NUMERAL THIRTY
1034A ; Nl # GOTHIC LETTER NINE HUNDRED

Total code points: 53

==

00B2..00B3 ; No # [2] SUPERSCRIPT TWO..SUPERSCRIPT THREE
00B9 ; No # SUPERSCRIPT ONE
00BC..00BE ; No # [3] VULGAR FRACTION ONE QUARTER..VULGAR FRACTION THREE QUARTERS
09F4..09F9 ; No # [6] BENGALI CURRENCY NUMERATOR ONE..BENGALI CURRENCY DENOMINATOR SIXTEEN
0BF0..0BF2 ; No # [3] TAMIL NUMBER TEN..TAMIL NUMBER ONE THOUSAND
0F2A..0F33 ; No # [10] TIBETAN DIGIT HALF ONE..TIBETAN DIGIT HALF ZERO
1372..137C ; No # [11] ETHIOPIC NUMBER TEN..ETHIOPIC NUMBER TEN THOUSAND
2070 ; No # SUPERSCRIPT ZERO
2074..2079 ; No # [6] SUPERSCRIPT FOUR..SUPERSCRIPT NINE
2080..2089 ; No # [10] SUBSCRIPT ZERO..SUBSCRIPT NINE
2153..215F ; No # [13] VULGAR FRACTION ONE THIRD..FRACTION NUMERATOR ONE
2460..249B ; No # [60] CIRCLED DIGIT ONE..NUMBER TWENTY FULL STOP
24EA ; No # CIRCLED DIGIT ZERO
2776..2793 ; No # [30] DINGBAT NEGATIVE CIRCLED DIGIT ONE..DINGBAT NEGATIVE CIRCLED SANS-SERIF NUMBER TEN
3192..3195 ; No # [4] IDEOGRAPHIC ANNOTATION ONE MARK..IDEOGRAPHIC ANNOTATION FOUR MARK
3220..3229 ; No # [10] PARENTHESIZED IDEOGRAPH ONE..PARENTHESIZED IDEOGRAPH TEN
3280..3289 ; No # [10] CIRCLED IDEOGRAPH ONE..CIRCLED IDEOGRAPH TEN
10320..10323 ; No # [4] OLD ITALIC NUMERAL ONE..OLD ITALIC NUMERAL FIFTY

Total code points: 185

==

0020 ; Zs # SPACE
00A0 ; Zs # NO-BREAK SPACE
1680 ; Zs # OGHAM SPACE MARK
2000..200B ; Zs # [12] EN QUAD..ZERO WIDTH SPACE
202F ; Zs # NARROW NO-BREAK SPACE
3000 ; Zs # IDEOGRAPHIC SPACE

Total code points: 17

==

2028 ; Zl # LINE SEPARATOR

Total code points: 1

==

2029 ; Zp # PARAGRAPH SEPARATOR

Total code points: 1

==

0000..001F ; Cc # [32] <control>..<control>
007F..009F ; Cc # [33] <control>..<control>

Total code points: 65

==

070F ; Cf # SYRIAC ABBREVIATION MARK
180B..180E ; Cf # [4] MONGOLIAN FREE VARIATION SELECTOR ONE..MONGOLIAN VOWEL SEPARATOR
200C..200F ; Cf # [4] ZERO WIDTH NON-JOINER..RIGHT-TO-LEFT MARK
202A..202E ; Cf # [5] LEFT-TO-RIGHT EMBEDDING..RIGHT-TO-LEFT OVERRIDE
206A..206F ; Cf # [6] INHIBIT SYMMETRIC SWAPPING..NOMINAL DIGIT SHAPES
FEFF ; Cf # ZERO WIDTH NO-BREAK SPACE
FFF9..FFFB ; Cf # [3] INTERLINEAR ANNOTATION ANCHOR..INTERLINEAR ANNOTATION TERMINATOR
1D173..1D17A ; Cf # [8] MUSICAL SYMBOL BEGIN BEAM..MUSICAL SYMBOL END PHRASE
E0001 ; Cf # LANGUAGE TAG
E0020..E007F ; Cf # [96] TAG SPACE..CANCEL TAG

Total code points: 129

==

E000..F8FF ; Co # [6400]
F0000..FFFFD ; Co # [65534]
100000..10FFFD; Co # [65534]

Total code points: 137468

==

D800..DFFF ; Cs # [2048]

Total code points: 2048

```
# ================================================

002D          ; Pd #       HYPHEN-MINUS
00AD          ; Pd #       SOFT HYPHEN
058A          ; Pd #       ARMENIAN HYPHEN
1806          ; Pd #       MONGOLIAN TODO SOFT HYPHEN
2010..2015    ; Pd #   [6] HYPHEN..HORIZONTAL BAR
301C          ; Pd #       WAVE DASH
3030          ; Pd #       WAVY DASH
FE31..FE32    ; Pd #   [2] PRESENTATION FORM FOR VERTICAL EM DASH..
PRESENTATION FORM FOR VERTICAL EN DASH
FE58          ; Pd #       SMALL EM DASH
FE63          ; Pd #       SMALL HYPHEN-MINUS
FF0D          ; Pd #       FULLWIDTH HYPHEN-MINUS

# Total code points: 17

# ================================================

0028          ; Ps #       LEFT PARENTHESIS
005B          ; Ps #       LEFT SQUARE BRACKET
007B          ; Ps #       LEFT CURLY BRACKET
0F3A          ; Ps #       TIBETAN MARK GUG RTAGS GYON
0F3C          ; Ps #       TIBETAN MARK ANG KHANG GYON
169B          ; Ps #       OGHAM FEATHER MARK
201A          ; Ps #       SINGLE LOW-9 QUOTATION MARK
201E          ; Ps #       DOUBLE LOW-9 QUOTATION MARK
2045          ; Ps #       LEFT SQUARE BRACKET WITH QUILL
207D          ; Ps #       SUPERSCRIPT LEFT PARENTHESIS
208D          ; Ps #       SUBSCRIPT LEFT PARENTHESIS
2329          ; Ps #       LEFT-POINTING ANGLE BRACKET
3008          ; Ps #       LEFT ANGLE BRACKET
300A          ; Ps #       LEFT DOUBLE ANGLE BRACKET
300C          ; Ps #       LEFT CORNER BRACKET
300E          ; Ps #       LEFT WHITE CORNER BRACKET
3010          ; Ps #       LEFT BLACK LENTICULAR BRACKET
3014          ; Ps #       LEFT TORTOISE SHELL BRACKET
3016          ; Ps #       LEFT WHITE LENTICULAR BRACKET
3018          ; Ps #       LEFT WHITE TORTOISE SHELL BRACKET
301A          ; Ps #       LEFT WHITE SQUARE BRACKET
301D          ; Ps #       REVERSED DOUBLE PRIME QUOTATION MARK
FD3E          ; Ps #       ORNATE LEFT PARENTHESIS
FE35          ; Ps #       PRESENTATION FORM FOR VERTICAL LEFT PARENTHESIS
FE37          ; Ps #       PRESENTATION FORM FOR VERTICAL LEFT CURLY BRACKET
FE39          ; Ps #       PRESENTATION FORM FOR VERTICAL LEFT TORTOISE SHELL
BRACKET
```

FE3B ; Ps # PRESENTATION FORM FOR VERTICAL LEFT BLACK LENTICULAR BRACKET
FE3D ; Ps # PRESENTATION FORM FOR VERTICAL LEFT DOUBLE ANGLE BRACKET
FE3F ; Ps # PRESENTATION FORM FOR VERTICAL LEFT ANGLE BRACKET
FE41 ; Ps # PRESENTATION FORM FOR VERTICAL LEFT CORNER BRACKET
FE43 ; Ps # PRESENTATION FORM FOR VERTICAL LEFT WHITE CORNER BRACKET
FE59 ; Ps # SMALL LEFT PARENTHESIS
FE5B ; Ps # SMALL LEFT CURLY BRACKET
FE5D ; Ps # SMALL LEFT TORTOISE SHELL BRACKET
FF08 ; Ps # FULLWIDTH LEFT PARENTHESIS
FF3B ; Ps # FULLWIDTH LEFT SQUARE BRACKET
FF5B ; Ps # FULLWIDTH LEFT CURLY BRACKET
FF62 ; Ps # HALFWIDTH LEFT CORNER BRACKET

Total code points: 38

==

0029 ; Pe # RIGHT PARENTHESIS
005D ; Pe # RIGHT SQUARE BRACKET
007D ; Pe # RIGHT CURLY BRACKET
0F3B ; Pe # TIBETAN MARK GUG RTAGS GYAS
0F3D ; Pe # TIBETAN MARK ANG KHANG GYAS
169C ; Pe # OGHAM REVERSED FEATHER MARK
2046 ; Pe # RIGHT SQUARE BRACKET WITH QUILL
207E ; Pe # SUPERSCRIPT RIGHT PARENTHESIS
208E ; Pe # SUBSCRIPT RIGHT PARENTHESIS
232A ; Pe # RIGHT-POINTING ANGLE BRACKET
3009 ; Pe # RIGHT ANGLE BRACKET
300B ; Pe # RIGHT DOUBLE ANGLE BRACKET
300D ; Pe # RIGHT CORNER BRACKET
300F ; Pe # RIGHT WHITE CORNER BRACKET
3011 ; Pe # RIGHT BLACK LENTICULAR BRACKET
3015 ; Pe # RIGHT TORTOISE SHELL BRACKET
3017 ; Pe # RIGHT WHITE LENTICULAR BRACKET
3019 ; Pe # RIGHT WHITE TORTOISE SHELL BRACKET
301B ; Pe # RIGHT WHITE SQUARE BRACKET
301E..301F ; Pe # [2] DOUBLE PRIME QUOTATION MARK..LOW DOUBLE PRIME QUOTATION MARK
FD3F ; Pe # ORNATE RIGHT PARENTHESIS
FE36 ; Pe # PRESENTATION FORM FOR VERTICAL RIGHT PARENTHESIS
FE38 ; Pe # PRESENTATION FORM FOR VERTICAL RIGHT CURLY BRACKET
FE3A ; Pe # PRESENTATION FORM FOR VERTICAL RIGHT TORTOISE SHELL BRACKET

FE3C ; Pe # PRESENTATION FORM FOR VERTICAL RIGHT BLACK LENTICULAR BRACKET
FE3E ; Pe # PRESENTATION FORM FOR VERTICAL RIGHT DOUBLE ANGLE BRACKET
FE40 ; Pe # PRESENTATION FORM FOR VERTICAL RIGHT ANGLE BRACKET
FE42 ; Pe # PRESENTATION FORM FOR VERTICAL RIGHT CORNER BRACKET
FE44 ; Pe # PRESENTATION FORM FOR VERTICAL RIGHT WHITE CORNER BRACKET
FE5A ; Pe # SMALL RIGHT PARENTHESIS
FE5C ; Pe # SMALL RIGHT CURLY BRACKET
FE5E ; Pe # SMALL RIGHT TORTOISE SHELL BRACKET
FF09 ; Pe # FULLWIDTH RIGHT PARENTHESIS
FF3D ; Pe # FULLWIDTH RIGHT SQUARE BRACKET
FF5D ; Pe # FULLWIDTH RIGHT CURLY BRACKET
FF63 ; Pe # HALFWIDTH RIGHT CORNER BRACKET

Total code points: 37

==

005F ; Pc # LOW LINE
203F..2040 ; Pc # [2] UNDERTIE..CHARACTER TIE
30FB ; Pc # KATAKANA MIDDLE DOT
FE33..FE34 ; Pc # [2] PRESENTATION FORM FOR VERTICAL LOW LINE..PRESENTATION FORM FOR VERTICAL WAVY LOW LINE
FE4D..FE4F ; Pc # [3] DASHED LOW LINE..WAVY LOW LINE
FF3F ; Pc # FULLWIDTH LOW LINE
FF65 ; Pc # HALFWIDTH KATAKANA MIDDLE DOT

Total code points: 11

==

0021..0023 ; Po # [3] EXCLAMATION MARK..NUMBER SIGN
0025..0027 ; Po # [3] PERCENT SIGN..APOSTROPHE
002A ; Po # ASTERISK
002C ; Po # COMMA
002E..002F ; Po # [2] FULL STOP..SOLIDUS
003A..003B ; Po # [2] COLON..SEMICOLON
003F..0040 ; Po # [2] QUESTION MARK..COMMERCIAL AT
005C ; Po # REVERSE SOLIDUS
00A1 ; Po # INVERTED EXCLAMATION MARK
00B7 ; Po # MIDDLE DOT
00BF ; Po # INVERTED QUESTION MARK
037E ; Po # GREEK QUESTION MARK
0387 ; Po # GREEK ANO TELEIA

055A..055F ; Po # [6] ARMENIAN APOSTROPHE..ARMENIAN ABBREVIATION MARK
0589 ; Po # ARMENIAN FULL STOP
05BE ; Po # HEBREW PUNCTUATION MAQAF
05C0 ; Po # HEBREW PUNCTUATION PASEQ
05C3 ; Po # HEBREW PUNCTUATION SOF PASUQ
05F3..05F4 ; Po # [2] HEBREW PUNCTUATION GERESH..HEBREW PUNCTUATION GERSHAYIM
060C ; Po # ARABIC COMMA
061B ; Po # ARABIC SEMICOLON
061F ; Po # ARABIC QUESTION MARK
066A..066D ; Po # [4] ARABIC PERCENT SIGN..ARABIC FIVE POINTED STAR
06D4 ; Po # ARABIC FULL STOP
0700..070D ; Po # [14] SYRIAC END OF PARAGRAPH..SYRIAC HARKLEAN ASTERISCUS
0964..0965 ; Po # [2] DEVANAGARI DANDA..DEVANAGARI DOUBLE DANDA
0970 ; Po # DEVANAGARI ABBREVIATION SIGN
0DF4 ; Po # SINHALA PUNCTUATION KUNDDALIYA
0E4F ; Po # THAI CHARACTER FONGMAN
0E5A..0E5B ; Po # [2] THAI CHARACTER ANGKHANKHU..THAI CHARACTER KHOMUT
0F04..0F12 ; Po # [15] TIBETAN MARK INITIAL YIG MGO MDUN MA..TIBETAN MARK RGYA GRAM SHAD
0F85 ; Po # TIBETAN MARK PALUTA
104A..104F ; Po # [6] MYANMAR SIGN LITTLE SECTION..MYANMAR SYMBOL GENITIVE
10FB ; Po # GEORGIAN PARAGRAPH SEPARATOR
1361..1368 ; Po # [8] ETHIOPIC WORDSPACE..ETHIOPIC PARAGRAPH SEPARATOR
166D..166E ; Po # [2] CANADIAN SYLLABICS CHI SIGN..CANADIAN SYLLABICS FULL STOP
16EB..16ED ; Po # [3] RUNIC SINGLE PUNCTUATION..RUNIC CROSS PUNCTUATION
17D4..17DA ; Po # [7] KHMER SIGN KHAN..KHMER SIGN KOOMUUT
17DC ; Po # KHMER SIGN AVAKRAHASANYA
1800..1805 ; Po # [6] MONGOLIAN BIRGA..MONGOLIAN FOUR DOTS
1807..180A ; Po # [4] MONGOLIAN SIBE SYLLABLE BOUNDARY MARKER.. MONGOLIAN NIRUGU
2016..2017 ; Po # [2] DOUBLE VERTICAL LINE..DOUBLE LOW LINE
2020..2027 ; Po # [8] DAGGER..HYPHENATION POINT
2030..2038 ; Po # [9] PER MILLE SIGN..CARET
203B..203E ; Po # [4] REFERENCE MARK..OVERLINE
2041..2043 ; Po # [3] CARET INSERTION POINT..HYPHEN BULLET
2048..204D ; Po # [6] QUESTION EXCLAMATION MARK..BLACK RIGHTWARDS BULLET
3001..3003 ; Po # [3] IDEOGRAPHIC COMMA..DITTO MARK
FE30 ; Po # PRESENTATION FORM FOR VERTICAL TWO DOT LEADER
FE49..FE4C ; Po # [4] DASHED OVERLINE..DOUBLE WAVY OVERLINE
FE50..FE52 ; Po # [3] SMALL COMMA..SMALL FULL STOP
FE54..FE57 ; Po # [4] SMALL SEMICOLON..SMALL EXCLAMATION MARK
FE5F..FE61 ; Po # [3] SMALL NUMBER SIGN..SMALL ASTERISK
FE68 ; Po # SMALL REVERSE SOLIDUS

FE6A..FE6B ; Po # [2] SMALL PERCENT SIGN..SMALL COMMERCIAL AT
FF01..FF03 ; Po # [3] FULLWIDTH EXCLAMATION MARK..FULLWIDTH NUMBER SIGN
FF05..FF07 ; Po # [3] FULLWIDTH PERCENT SIGN..FULLWIDTH APOSTROPHE
FF0A ; Po # FULLWIDTH ASTERISK
FF0C ; Po # FULLWIDTH COMMA
FF0E..FF0F ; Po # [2] FULLWIDTH FULL STOP..FULLWIDTH SOLIDUS
FF1A..FF1B ; Po # [2] FULLWIDTH COLON..FULLWIDTH SEMICOLON
FF1F..FF20 ; Po # [2] FULLWIDTH QUESTION MARK..FULLWIDTH COMMERCIAL AT
FF3C ; Po # FULLWIDTH REVERSE SOLIDUS
FF61 ; Po # HALFWIDTH IDEOGRAPHIC FULL STOP
FF64 ; Po # HALFWIDTH IDEOGRAPHIC COMMA

Total code points: 185

==

002B ; Sm # PLUS SIGN
003C..003E ; Sm # [3] LESS-THAN SIGN..GREATER-THAN SIGN
007C ; Sm # VERTICAL LINE
007E ; Sm # TILDE
00AC ; Sm # NOT SIGN
00B1 ; Sm # PLUS-MINUS SIGN
00D7 ; Sm # MULTIPLICATION SIGN
00F7 ; Sm # DIVISION SIGN
2044 ; Sm # FRACTION SLASH
207A..207C ; Sm # [3] SUPERSCRIPT PLUS SIGN..SUPERSCRIPT EQUALS SIGN
208A..208C ; Sm # [3] SUBSCRIPT PLUS SIGN..SUBSCRIPT EQUALS SIGN
2190..2194 ; Sm # [5] LEFTWARDS ARROW..LEFT RIGHT ARROW
219A..219B ; Sm # [2] LEFTWARDS ARROW WITH STROKE..RIGHTWARDS ARROW WITH STROKE
21A0 ; Sm # RIGHTWARDS TWO HEADED ARROW
21A3 ; Sm # RIGHTWARDS ARROW WITH TAIL
21A6 ; Sm # RIGHTWARDS ARROW FROM BAR
21AE ; Sm # LEFT RIGHT ARROW WITH STROKE
21CE..21CF ; Sm # [2] LEFT RIGHT DOUBLE ARROW WITH STROKE..RIGHTWARDS DOUBLE ARROW WITH STROKE
21D2 ; Sm # RIGHTWARDS DOUBLE ARROW
21D4 ; Sm # LEFT RIGHT DOUBLE ARROW
2200..22F1 ; Sm # [242] FOR ALL..DOWN RIGHT DIAGONAL ELLIPSIS
2308..230B ; Sm # [4] LEFT CEILING..RIGHT FLOOR
2320..2321 ; Sm # [2] TOP HALF INTEGRAL..BOTTOM HALF INTEGRAL
25B7 ; Sm # WHITE RIGHT-POINTING TRIANGLE
25C1 ; Sm # WHITE LEFT-POINTING TRIANGLE
266F ; Sm # MUSIC SHARP SIGN
FB29 ; Sm # HEBREW LETTER ALTERNATIVE PLUS SIGN
FE62 ; Sm # SMALL PLUS SIGN

FE64..FE66 ; Sm # [3] SMALL LESS-THAN SIGN..SMALL EQUALS SIGN
FF0B ; Sm # FULLWIDTH PLUS SIGN
FF1C..FF1E ; Sm # [3] FULLWIDTH LESS-THAN SIGN..FULLWIDTH GREATER-THAN SIGN
FF5C ; Sm # FULLWIDTH VERTICAL LINE
FF5E ; Sm # FULLWIDTH TILDE
FFE2 ; Sm # FULLWIDTH NOT SIGN
FFE9..FFEC ; Sm # [4] HALFWIDTH LEFTWARDS ARROW..HALFWIDTH DOWNWARDS ARROW
1D6C1 ; Sm # MATHEMATICAL BOLD NABLA
1D6DB ; Sm # MATHEMATICAL BOLD PARTIAL DIFFERENTIAL
1D6FB ; Sm # MATHEMATICAL ITALIC NABLA
1D715 ; Sm # MATHEMATICAL ITALIC PARTIAL DIFFERENTIAL
1D735 ; Sm # MATHEMATICAL BOLD ITALIC NABLA
1D74F ; Sm # MATHEMATICAL BOLD ITALIC PARTIAL DIFFERENTIAL
1D76F ; Sm # MATHEMATICAL SANS-SERIF BOLD NABLA
1D789 ; Sm # MATHEMATICAL SANS-SERIF BOLD PARTIAL DIFFERENTIAL
1D7A9 ; Sm # MATHEMATICAL SANS-SERIF BOLD ITALIC NABLA
1D7C3 ; Sm # MATHEMATICAL SANS-SERIF BOLD ITALIC PARTIAL DIFFERENTIAL

Total code points: 309

==

0024 ; Sc # DOLLAR SIGN
00A2..00A5 ; Sc # [4] CENT SIGN..YEN SIGN
09F2..09F3 ; Sc # [2] BENGALI RUPEE MARK..BENGALI RUPEE SIGN
0E3F ; Sc # THAI CURRENCY SYMBOL BAHT
17DB ; Sc # KHMER CURRENCY SYMBOL RIEL
20A0..20AF ; Sc # [16] EURO-CURRENCY SIGN..DRACHMA SIGN
FE69 ; Sc # SMALL DOLLAR SIGN
FF04 ; Sc # FULLWIDTH DOLLAR SIGN
FFE0..FFE1 ; Sc # [2] FULLWIDTH CENT SIGN..FULLWIDTH POUND SIGN
FFE5..FFE6 ; Sc # [2] FULLWIDTH YEN SIGN..FULLWIDTH WON SIGN

Total code points: 31

==

005E ; Sk # CIRCUMFLEX ACCENT
0060 ; Sk # GRAVE ACCENT
00A8 ; Sk # DIAERESIS
00AF ; Sk # MACRON
00B4 ; Sk # ACUTE ACCENT
00B8 ; Sk # CEDILLA

02B9..02BA ; Sk # [2] MODIFIER LETTER PRIME..MODIFIER LETTER DOUBLE PRIME
02C2..02CF ; Sk # [14] MODIFIER LETTER LEFT ARROWHEAD..MODIFIER LETTER LOW ACUTE ACCENT
02D2..02DF ; Sk # [14] MODIFIER LETTER CENTRED RIGHT HALF RING..MODIFIER LETTER CROSS ACCENT
02E5..02ED ; Sk # [9] MODIFIER LETTER EXTRA-HIGH TONE BAR..MODIFIER LETTER UNASPIRATED
0374..0375 ; Sk # [2] GREEK NUMERAL SIGN..GREEK LOWER NUMERAL SIGN
0384..0385 ; Sk # [2] GREEK TONOS..GREEK DIALYTIKA TONOS
1FBD ; Sk # GREEK KORONIS
1FBF..1FC1 ; Sk # [3] GREEK PSILI..GREEK DIALYTIKA AND PERISPOMENI
1FCD..1FCF ; Sk # [3] GREEK PSILI AND VARIA..GREEK PSILI AND PERISPOMENI
1FDD..1FDF ; Sk # [3] GREEK DASIA AND VARIA..GREEK DASIA AND PERISPOMENI
1FED..1FEF ; Sk # [3] GREEK DIALYTIKA AND VARIA..GREEK VARIA
1FFD..1FFE ; Sk # [2] GREEK OXIA..GREEK DASIA
309B..309C ; Sk # [2] KATAKANA-HIRAGANA VOICED SOUND MARK..KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK
FF3E ; Sk # FULLWIDTH CIRCUMFLEX ACCENT
FF40 ; Sk # FULLWIDTH GRAVE ACCENT
FFE3 ; Sk # FULLWIDTH MACRON

Total code points: 69

==

00A6..00A7 ; So # [2] BROKEN BAR..SECTION SIGN
00A9 ; So # COPYRIGHT SIGN
00AE ; So # REGISTERED SIGN
00B0 ; So # DEGREE SIGN
00B6 ; So # PILCROW SIGN
0482 ; So # CYRILLIC THOUSANDS SIGN
06E9 ; So # ARABIC PLACE OF SAJDAH
06FD..06FE ; So # [2] ARABIC SIGN SINDHI AMPERSAND..ARABIC SIGN SINDHI POSTPOSITION MEN
09FA ; So # BENGALI ISSHAR
0B70 ; So # ORIYA ISSHAR
0F01..0F03 ; So # [3] TIBETAN MARK GTER YIG MGO TRUNCATED A..TIBETAN MARK GTER YIG MGO -UM GTER TSHEG MA
0F13..0F17 ; So # [5] TIBETAN MARK CARET -DZUD RTAGS ME LONG CAN..TIBETAN ASTROLOGICAL SIGN SGRA GCAN -CHAR RTAGS
0F1A..0F1F ; So # [6] TIBETAN SIGN RDEL DKAR GCIG..TIBETAN SIGN RDEL DKAR RDEL NAG
0F34 ; So # TIBETAN MARK BSDUS RTAGS
0F36 ; So # TIBETAN MARK CARET -DZUD RTAGS BZHI MIG CAN
0F38 ; So # TIBETAN MARK CHE MGO
0FBE..0FC5 ; So # [8] TIBETAN KU RU KHA..TIBETAN SYMBOL RDO RJE

0FC7..0FCC ; So # [6] TIBETAN SYMBOL RDO RJE RGYA GRAM..TIBETAN SYMBOL NOR BU BZHI -KHYIL
0FCF ; So # TIBETAN SIGN RDEL NAG GSUM
2100..2101 ; So # [2] ACCOUNT OF..ADDRESSED TO THE SUBJECT
2103..2106 ; So # [4] DEGREE CELSIUS..CADA UNA
2108..2109 ; So # [2] SCRUPLE..DEGREE FAHRENHEIT
2114 ; So # L B BAR SYMBOL
2116..2118 ; So # [3] NUMERO SIGN..SCRIPT CAPITAL P
211E..2123 ; So # [6] PRESCRIPTION TAKE..VERSICLE
2125 ; So # OUNCE SIGN
2127 ; So # INVERTED OHM SIGN
2129 ; So # TURNED GREEK SMALL LETTER IOTA
212E ; So # ESTIMATED SYMBOL
2132 ; So # TURNED CAPITAL F
213A ; So # ROTATED CAPITAL Q
2195..2199 ; So # [5] UP DOWN ARROW..SOUTH WEST ARROW
219C..219F ; So # [4] LEFTWARDS WAVE ARROW..UPWARDS TWO HEADED ARROW
21A1..21A2 ; So # [2] DOWNWARDS TWO HEADED ARROW..LEFTWARDS ARROW WITH TAIL
21A4..21A5 ; So # [2] LEFTWARDS ARROW FROM BAR..UPWARDS ARROW FROM BAR
21A7..21AD ; So # [7] DOWNWARDS ARROW FROM BAR..LEFT RIGHT WAVE ARROW
21AF..21CD ; So # [31] DOWNWARDS ZIGZAG ARROW..LEFTWARDS DOUBLE ARROW WITH STROKE
21D0..21D1 ; So # [2] LEFTWARDS DOUBLE ARROW..UPWARDS DOUBLE ARROW
21D3 ; So # DOWNWARDS DOUBLE ARROW
21D5..21F3 ; So # [31] UP DOWN DOUBLE ARROW..UP DOWN WHITE ARROW
2300..2307 ; So # [8] DIAMETER SIGN..WAVY LINE
230C..231F ; So # [20] BOTTOM RIGHT CROP..BOTTOM RIGHT CORNER
2322..2328 ; So # [7] FROWN..KEYBOARD
232B..237B ; So # [81] ERASE TO THE LEFT..NOT CHECK MARK
237D..239A ; So # [30] SHOULDERED OPEN BOX..CLEAR SCREEN SYMBOL
2400..2426 ; So # [39] SYMBOL FOR NULL..SYMBOL FOR SUBSTITUTE FORM TWO
2440..244A ; So # [11] OCR HOOK..OCR DOUBLE BACKSLASH
249C..24E9 ; So # [78] PARENTHESIZED LATIN SMALL LETTER A..CIRCLED LATIN SMALL LETTER Z
2500..2595 ; So # [150] BOX DRAWINGS LIGHT HORIZONTAL..RIGHT ONE EIGHTH BLOCK
25A0..25B6 ; So # [23] BLACK SQUARE..BLACK RIGHT-POINTING TRIANGLE
25B8..25C0 ; So # [9] BLACK RIGHT-POINTING SMALL TRIANGLE..BLACK LEFT-POINTING TRIANGLE
25C2..25F7 ; So # [54] BLACK LEFT-POINTING SMALL TRIANGLE..WHITE CIRCLE WITH UPPER RIGHT QUADRANT
2600..2613 ; So # [20] BLACK SUN WITH RAYS..SALTIRE
2619..266E ; So # [86] REVERSED ROTATED FLORAL HEART BULLET..MUSIC NATURAL SIGN
2670..2671 ; So # [2] WEST SYRIAC CROSS..EAST SYRIAC CROSS

2701..2704 ; So # [4] UPPER BLADE SCISSORS..WHITE SCISSORS
2706..2709 ; So # [4] TELEPHONE LOCATION SIGN..ENVELOPE
270C..2727 ; So # [28] VICTORY HAND..WHITE FOUR POINTED STAR
2729..274B ; So # [35] STRESS OUTLINED WHITE STAR..HEAVY EIGHT TEARDROP-SPOKED PROPELLER ASTERISK
274D ; So # SHADOWED WHITE CIRCLE
274F..2752 ; So # [4] LOWER RIGHT DROP-SHADOWED WHITE SQUARE..UPPER RIGHT SHADOWED WHITE SQUARE
2756 ; So # BLACK DIAMOND MINUS WHITE X
2758..275E ; So # [7] LIGHT VERTICAL BAR..HEAVY DOUBLE COMMA QUOTATION MARK ORNAMENT
2761..2767 ; So # [7] CURVED STEM PARAGRAPH SIGN ORNAMENT..ROTATED FLORAL HEART BULLET
2794 ; So # HEAVY WIDE-HEADED RIGHTWARDS ARROW
2798..27AF ; So # [24] HEAVY SOUTH EAST ARROW..NOTCHED LOWER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
27B1..27BE ; So # [14] NOTCHED UPPER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW..OPEN-OUTLINED RIGHTWARDS ARROW
2800..28FF ; So # [256] BRAILLE PATTERN BLANK..BRAILLE PATTERN DOTS-12345678
2E80..2E99 ; So # [26] CJK RADICAL REPEAT..CJK RADICAL RAP
2E9B..2EF3 ; So # [89] CJK RADICAL CHOKE..CJK RADICAL C-SIMPLIFIED TURTLE
2F00..2FD5 ; So # [214] KANGXI RADICAL ONE..KANGXI RADICAL FLUTE
2FF0..2FFB ; So # [12] IDEOGRAPHIC DESCRIPTION CHARACTER LEFT TO RIGHT..IDEOGRAPHIC DESCRIPTION CHARACTER OVERLAID
3004 ; So # JAPANESE INDUSTRIAL STANDARD SYMBOL
3012..3013 ; So # [2] POSTAL MARK..GETA MARK
3020 ; So # POSTAL MARK FACE
3036..3037 ; So # [2] CIRCLED POSTAL MARK..IDEOGRAPHIC TELEGRAPH LINE FEED SEPARATOR SYMBOL
303E..303F ; So # [2] IDEOGRAPHIC VARIATION INDICATOR..IDEOGRAPHIC HALF FILL SPACE
3190..3191 ; So # [2] IDEOGRAPHIC ANNOTATION LINKING MARK..IDEOGRAPHIC ANNOTATION REVERSE MARK
3196..319F ; So # [10] IDEOGRAPHIC ANNOTATION TOP MARK..IDEOGRAPHIC ANNOTATION MAN MARK
3200..321C ; So # [29] PARENTHESIZED HANGUL KIYEOK..PARENTHESIZED HANGUL CIEUC U
322A..3243 ; So # [26] PARENTHESIZED IDEOGRAPH MOON..PARENTHESIZED IDEOGRAPH REACH
3260..327B ; So # [28] CIRCLED HANGUL KIYEOK..CIRCLED HANGUL HIEUH A
327F ; So # KOREAN STANDARD SYMBOL
328A..32B0 ; So # [39] CIRCLED IDEOGRAPH MOON..CIRCLED IDEOGRAPH NIGHT
32C0..32CB ; So # [12] IDEOGRAPHIC TELEGRAPH SYMBOL FOR JANUARY..IDEOGRAPHIC TELEGRAPH SYMBOL FOR DECEMBER
32D0..32FE ; So # [47] CIRCLED KATAKANA A..CIRCLED KATAKANA WO
3300..3376 ; So # [119] SQUARE APAATO..SQUARE PC

337B..33DD ; So # [99] SQUARE ERA NAME HEISEI..SQUARE WB
33E0..33FE ; So # [31] IDEOGRAPHIC TELEGRAPH SYMBOL FOR DAY ONE..
IDEOGRAPHIC TELEGRAPH SYMBOL FOR DAY THIRTY-ONE
A490..A4A1 ; So # [18] YI RADICAL QOT..YI RADICAL GA
A4A4..A4B3 ; So # [16] YI RADICAL DDUR..YI RADICAL JO
A4B5..A4C0 ; So # [12] YI RADICAL JJY..YI RADICAL SHAT
A4C2..A4C4 ; So # [3] YI RADICAL SHOP..YI RADICAL ZZIET
A4C6 ; So # YI RADICAL KE
FFE4 ; So # FULLWIDTH BROKEN BAR
FFE8 ; So # HALFWIDTH FORMS LIGHT VERTICAL
FFED..FFEE ; So # [2] HALFWIDTH BLACK SQUARE..HALFWIDTH WHITE CIRCLE
FFFC..FFFD ; So # [2] OBJECT REPLACEMENT CHARACTER..REPLACEMENT
CHARACTER
1D000..1D0F5 ; So # [246] BYZANTINE MUSICAL SYMBOL PSILI..BYZANTINE MUSICAL
SYMBOL GORGON NEO KATO
1D100..1D126 ; So # [39] MUSICAL SYMBOL SINGLE BARLINE..MUSICAL SYMBOL
DRUM CLEF-2
1D12A..1D164 ; So # [59] MUSICAL SYMBOL DOUBLE SHARP..MUSICAL SYMBOL ONE
HUNDRED TWENTY-EIGHTH NOTE
1D16A..1D16C ; So # [3] MUSICAL SYMBOL FINGERED TREMOLO-1..MUSICAL SYMBOL
FINGERED TREMOLO-3
1D183..1D184 ; So # [2] MUSICAL SYMBOL ARPEGGIATO UP..MUSICAL SYMBOL
ARPEGGIATO DOWN
1D18C..1D1A9 ; So # [30] MUSICAL SYMBOL RINFORZANDO..MUSICAL SYMBOL
DEGREE SLASH
1D1AE..1D1DD ; So # [48] MUSICAL SYMBOL PEDAL MARK..MUSICAL SYMBOL PES
SUBPUNCTIS

Total code points: 2432

==

00AB ; Pi # LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
2018 ; Pi # LEFT SINGLE QUOTATION MARK
201B..201C ; Pi # [2] SINGLE HIGH-REVERSED-9 QUOTATION MARK..LEFT DOUBLE
QUOTATION MARK
201F ; Pi # DOUBLE HIGH-REVERSED-9 QUOTATION MARK
2039 ; Pi # SINGLE LEFT-POINTING ANGLE QUOTATION MARK

Total code points: 6

==

00BB ; Pf # RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
2019 ; Pf # RIGHT SINGLE QUOTATION MARK
201D ; Pf # RIGHT DOUBLE QUOTATION MARK

203A ; Pf # SINGLE RIGHT-POINTING ANGLE QUOTATION MARK

Total code points: 4

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<xsd:attribute name="myAtt1"/>

	<xsd:attribute name="myAtt2" type="xsd:decimal"/>

	<xsd:attribute name="myAtt3">
		<xsd:simpleType>
			<xsd:restriction base="xsd:int">
				<xsd:minInclusive value="10"/>
				<xsd:maxInclusive value="20"/>
			</xsd:restriction>
		</xsd:simpleType>
	</xsd:attribute>

	<xsd:simpleType name="myType1">
		<xsd:restriction base="xsd:string">
			<xsd:maxLength value="5"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:attribute name="myAtt4" type="myType1"/>



	<xsd:element name="foo">
		<xsd:complexType>
			<xsd:attribute ref="myAtt1" use="optional"/>
			<xsd:attribute ref="myAtt2" use="required"/>
			<xsd:attribute ref="myAtt3" use="prohibited"/>
			<xsd:attribute ref="myAtt4"/>
			<xsd:attribute name="myAtt5" type="xsd:date" id="myDate"/>
			<xsd:attribute name="myAtt6">
				<xsd:simpleType>
					<xsd:restriction base="xsd:float">
						<xsd:totalDigits value="5"/>
					</xsd:restriction>
				</xsd:simpleType>
			</xsd:attribute>
		</xsd:complexType>
	</xsd:element>

</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<xsd:element name="Nachname" type="xsd:string"/>
	<xsd:complexType name="PersonType">
		<xsd:sequence>
			<xsd:element ref="Vorname" maxOccurs="unbounded"/>
			<xsd:element ref="Nachname" maxOccurs="unbounded"/>
			<xsd:element name="Qualifikationsprofil" type="QualifikationsprofilType"
minOccurs="0"/>
		</xsd:sequence>
		<xsd:attribute name="PersID" type="xsd:ID" use="required"/>
		<xsd:attribute name="Gehaltsgruppe" default="1a">
			<xsd:simpleType>
				<xsd:restriction base="xsd:NMTOKEN">
					<xsd:enumeration value="1"/>
					<xsd:enumeration value="1a"/>
					<xsd:enumeration value="2"/>
				</xsd:restriction>
			</xsd:simpleType>
		</xsd:attribute>
		<xsd:attribute name="mitarbeitInProjekt" type="xsd:IDREFS" use="required"/>
	</xsd:complexType>
	<xsd:complexType name="ProjektType">
		<xsd:attribute name="ID" type="xsd:ID" use="required"/>
		<xsd:attribute name="date" type="xsd:date"/>
		<xsd:attribute name="budget" default="10000.00">
			<xsd:simpleType>
				<xsd:restriction base="xsd:double">
					<xsd:fractionDigits value="2"/>
				</xsd:restriction>
			</xsd:simpleType>
		</xsd:attribute>
		<xsd:attribute name="Projektleiter" type="xsd:IDREF" use="required"/>
		<xsd:attribute name="Mitarbeiter" type="xsd:IDREFS" use="required"/>
	</xsd:complexType>
	<xsd:element name="ProjektVerwaltung">
		<xsd:complexType>
			<xsd:sequence>
				<xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/>
				<xsd:element name="Projekt" type="ProjektType" maxOccurs="unbounded"/>
			</xsd:sequence>
			<xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
		</xsd:complexType>
	</xsd:element>
	<xsd:complexType name="QualifikationsprofilType" mixed="true">
```

```
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element ref="Qualifikation"/>
                <xsd:element ref="Leistungsstufe"/>
                <xsd:any namespace="http://www.w3.org/1999/xhtml"/>
            </xsd:choice>
        </xsd:complexType>
        <xsd:element name="Qualifikation" type="xsd:string"/>
        <xsd:element name="Leistungsstufe" type="xsd:string"/>
        <xsd:element name="Vorname" type="xsd:string"/>
</xsd:schema>
```

TIBCO provides software and services that help companies orchestrate assets across their enterprise in real-time. TIBCO is the largest independent provider of such software, with almost 20 years of experience and 2,000 customers.

Upcoming Events

- Gartner Application Integration & Web Services Summit; May 17-19, Los Angeles.
- METAmorphosis 2004: The Adaptive Organization; May 25-27, Chicago.
- SIA Technology Management Conference; June 8, New York City.

In the News

- TIBCO Extends BAM Leadership With TIBCO OpsFactor™
- TIBCO Launches New Government Solutions Practice and EBC
- TIBCO Names Former Intuit CFO Eric Dunn to Board of Directors
- TIBCO® BusinessWorks Improves Business Agility for Securities Firms
- TIBCO to Host First Annual User Conference: TUCON™ 2004

TIBCO CEO on CNNfn

See TIBCO's Chairman and CEO, Vivek Ranadivé, on CNNfn's Stock of the Day.
MediaPlayer 9 | RealMedia 10

- Read Gartner's report "How the Pieces in a BAM Architecture Work."

ALTOVA markup your mind!

www.altova.com

Welcome to ALTOVA!

We apologize, but you will be unable to see our real ALTOVA web site, because your **Netscape v4** browser running on Unknown does not support accepted web standards (i.e. you are using an outdated or non-W3C-conforming version of a browser).

What web standards?

The ones created by the World Wide Web Consortium - the people who invented the Web itself. The W3C created these standards so the Web would work better for everyone. New browsers, mainly, support these standards; old browsers, mainly, don't.

As a member of the W3C, ALTOVA is committed to supporting the development of interoperable web standards, and as such we are using XML, XSLT, XHTML, and CSS technologies on our web site, which are either not supported by your current browser or can cause severe compatibility problems in your current browser.

What can I do?

You might consider upgrading your browser. Doing so will improve your web experience, enabling you to use and view sites as their creators intended.

The following browsers support numerous web standards including CSS, XHTML, and XML (listed in alphabetical order):

- Internet Explorer 6 or higher for Windows
- Internet Explorer 5 or higher for Macintosh
- Mozilla 1 or higher
- Netscape 7 or higher
- Opera 6 or higher

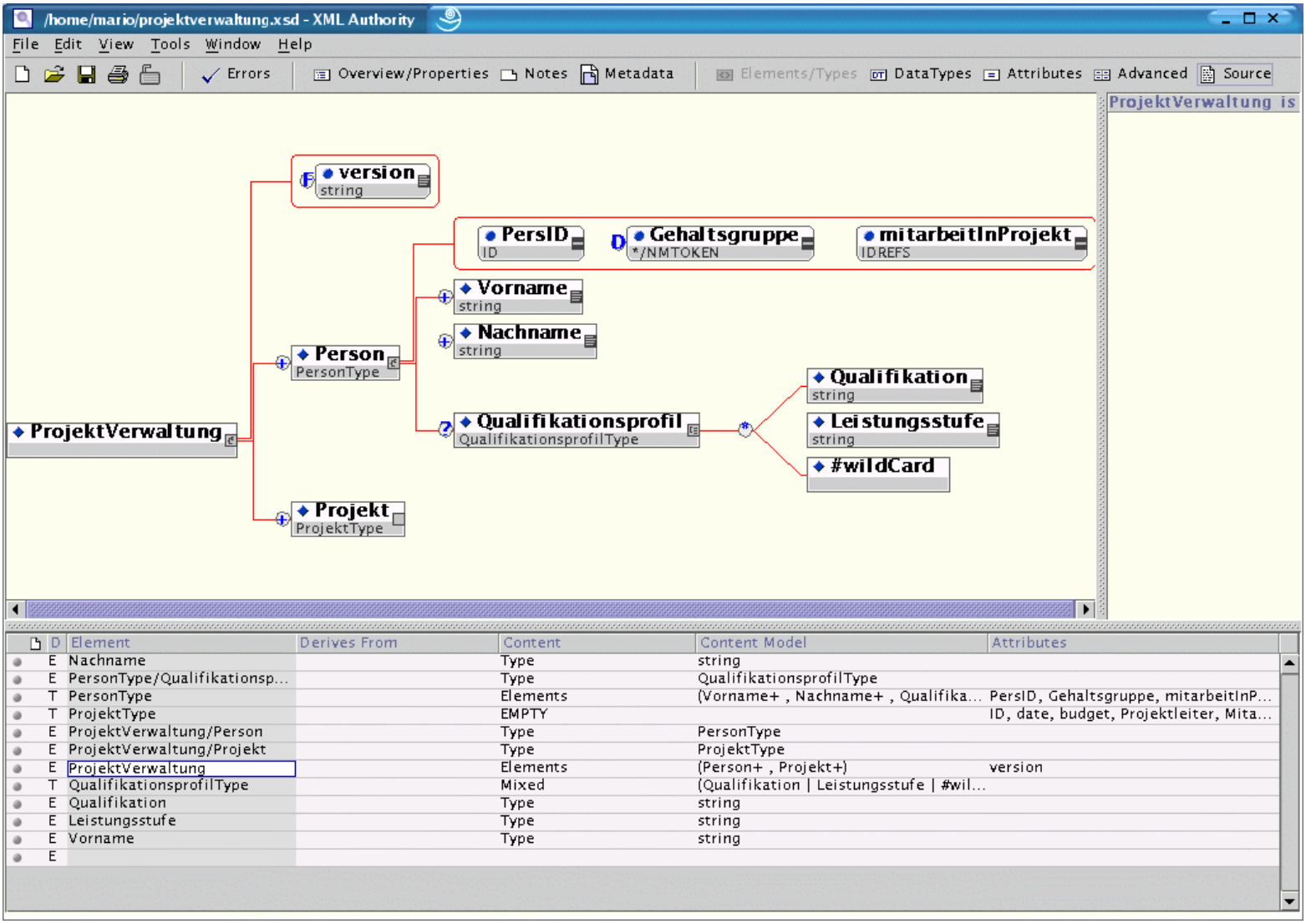
Please note that this page does not pretend to be an exhaustive list of browsers that support web standards, nor a test of browser compliance, nor a ranking or side-by-side comparison of various manufacturers' browsers.

What if you are mistaken?

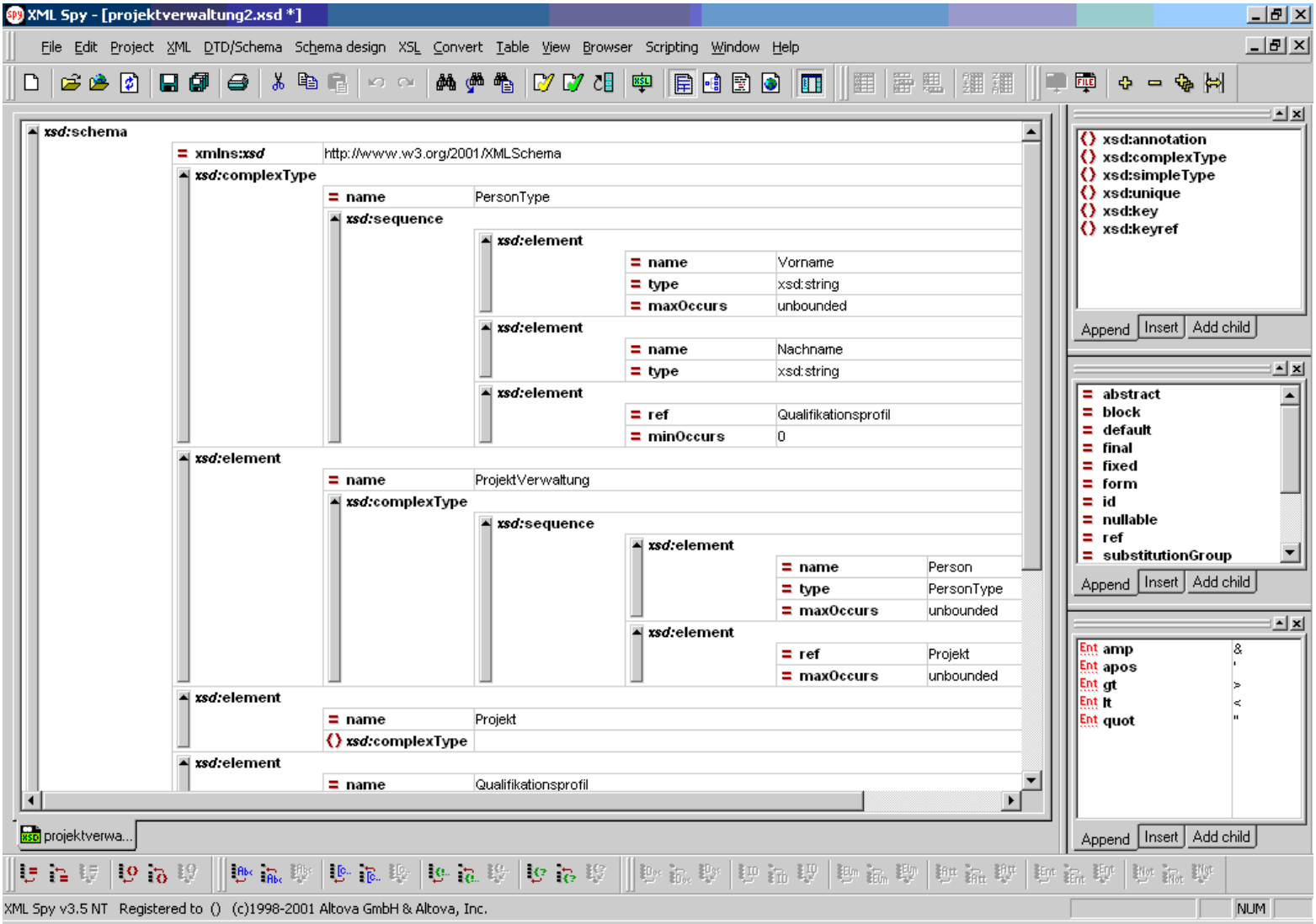
Do you believe we have incorrectly identified your browser as **Netscape v4** running on Unknown?

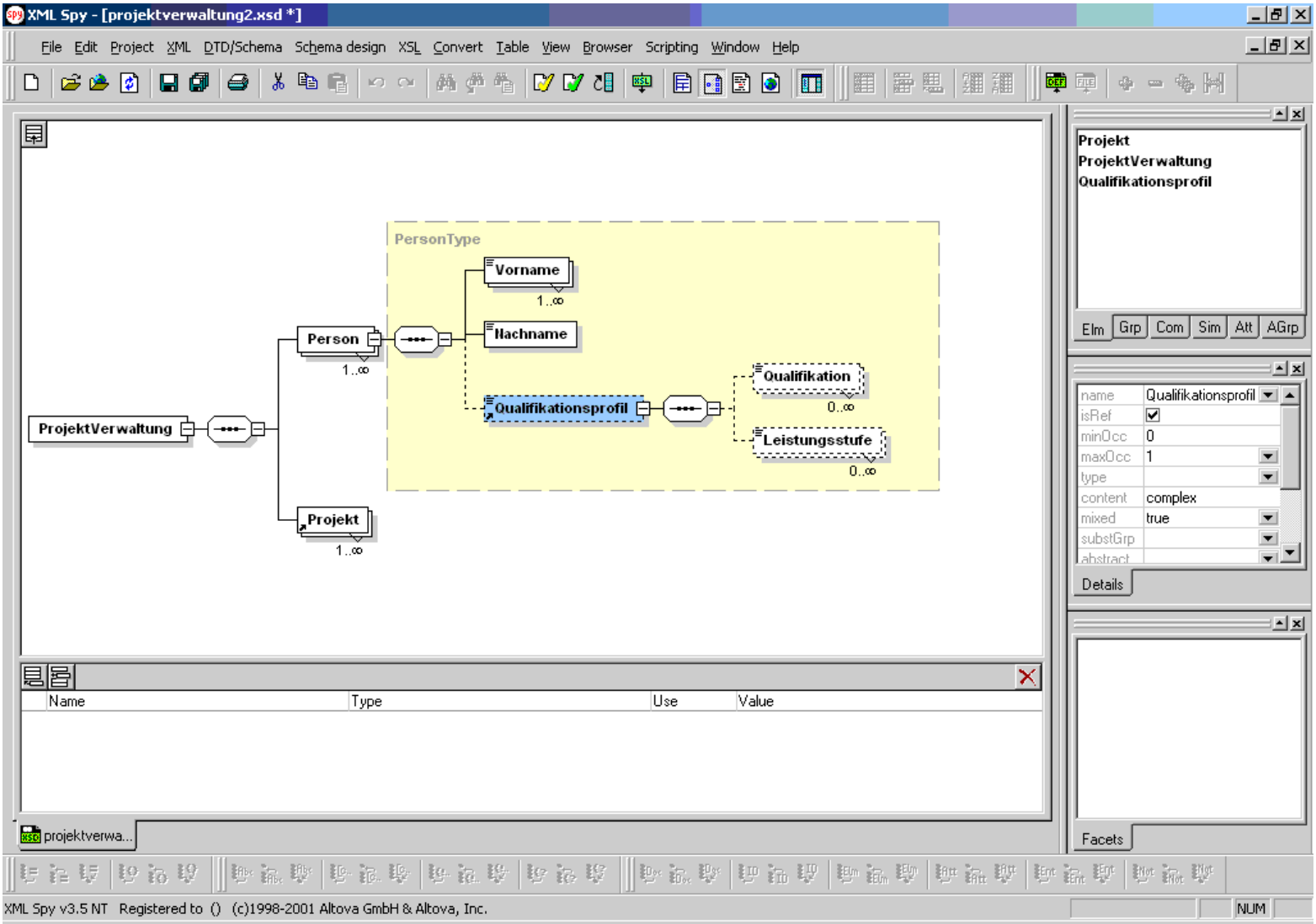
While we are using the "user-agent" information that is actually provided by your browser to determine it's version, it is still possible that we get the wrong information - e.g. if you are visiting our site through a proxy or firewall that doesn't pass on the correct "user-agent" information to our server.

If that is the case, or if you simply wish to access our site through an old browser despite our warning, please feel free to click here to enter the ALTOVA web site...



D	Element	Derives From	Content	Content Model	Attributes
E	Nachname		Type	string	
E	PersonType/Qualifikationsp...		Type	QualifikationsprofilType	
T	PersonType		Elements	(Vorname+ , Nachname+ , Qualifika...	PersID, Gehaltsgruppe, mitarbeitInP...
T	ProjektType		EMPTY		ID, date, budget, Projektleiter, Mita...
E	ProjektVerwaltung/Person		Type	PersonType	
E	ProjektVerwaltung/Projekt		Type	ProjektType	
E	ProjektVerwaltung		Elements	(Person+ , Projekt+)	version
T	QualifikationsprofilType		Mixed	(Qualifikation | Leistungsstufe | #wil...	
E	Qualifikation		Type	string	
E	Leistungsstufe		Type	string	
E	Vorname		Type	string	
E					


XML Spy - [projektverwaltung2.xsd *]
File Edit Project XML DTD/Schema Schema design XSL Convert Table View Browser Scripting Window Help
xsd:schema
xmlns:xsd http://www.w3.org/2001/XMLSchema
xsd:complexType
name PersonType
xsd:sequence
xsd:element
name Vorname
type xsd:string
maxOccurs unbounded
xsd:element
name Nachname
type xsd:string
xsd:element
ref Qualifikationsprofil
minOccurs 0
xsd:element
name ProjektVerwaltung
xsd:complexType
xsd:sequence
xsd:element
name Person
type PersonType
maxOccurs unbounded
xsd:element
ref Projekt
maxOccurs unbounded
xsd:element
name Projekt
xsd:complexType
xsd:element
name Qualifikationsprofil
projektverwa...
xsd:annotation
xsd:complexType
xsd:simpleType
xsd:unique
xsd:key
xsd:keyref
Append Insert Add child
abstract
block
default
final
fixed
form
id
nullable
ref
substitutionGroup
Append Insert Add child
Ent amp &
Ent apos '
Ent gt >
Ent lt <
Ent quot "
Append Insert Add child
XML Spy v3.5 NT Registered to () (c)1998-2001 Altova GmbH & Altova, Inc.
NUM


XML Spy - [projektverwaltung2.xsd *]
File Edit Project XML DTD/Schema Schema design XSL Convert Table View Browser Scripting Window Help
PersonType
Vorname
1..∞
Person
1..∞
Nachname
Qualifikation
0..∞
Qualifikationsprofil
ProjektVerwaltung
Leistungsstufe
0..∞
Projekt
1..∞
Projekt
ProjektVerwaltung
Qualifikationsprofil
Elm Grp Com Sim Att AGrp
name Qualifikationsprofil
isRef
minOcc 0
maxOcc 1
type
content complex
mixed true
substGrp
abstract
Details
Name Type Use Value
projektverwa...
Facets
XML Spy v3.5 NT Registered to () (c)1998-2001 Altova GmbH & Altova, Inc.
NUM

Last modified: January 23, 2004

Core Standards

XML Schemas

This document is designed to provide a reference list on "schemas" in the context of XML, RDF, SGML, EXPRESS, etc. [**Note:** Content and organization are provisional; need to clean up references in part 3 especially.]

Contents

XML Schema Definition Language: W3C XML Schema Working Group and Schema Specifications

WG Chairs

For up-to-date information, see Description from the XML Activity Statement.

[Fall 1999] The co-chairs of the XML Schema Working Group are Dave Hollander (CommerceNet) and C. M. Sperberg-McQueen (W3C).

Expected WG Deliverables

"The Schema Working group plans to deliver Requirements, Working Drafts, and Proposed Recommendations on data typing and schema language in 1999."

WG Description

"While XML 1.0 supplies a mechanism, the Document Type Definition (DTD) for declaring constraints on the use of markup, automated processing of XML documents requires more rigorous and comprehensive facilities in this area. Requirements are for constraints on how the component parts of an application fit together, the document structure, attributes, data-typing, and so on. The XML Schema Working Group is addressing means for defining the structure, content and semantics of XML documents." See XML Schema Requirements Comments mailing list: www-xml-schema-comments@w3.org and archive.

XML Schema Developers List: The **xmlschema-dev@w3.org** discussion list is publicly archived on the W3C server. Henry Thompson announced this public list on April 07, 2000 with a message "'XML Schema Developers List Launched': To accompany the XML Schema Last Call drafts, the W3C is pleased to announce the opening of a public mailing list for XML Schema implementation developers, `xmlschema-dev@w3.org`. To subscribe, send mail to xmlschema-dev-request@w3.org with 'subscribe' as the subject."

See also "XML Schema: Tools - Usage - Development".

WG Published Deliverables

XML Schema Requirements

- ***"XML Schema Requirements."*** W3C Note 15 February 1999. Edited by Ashok Malhotra (IBM) and Murray Maloney (Veo Systems Inc.). The document 'specifies the purpose, basic usage scenarios, design principles, and base requirements for an XML schema language.[local archive copy]

XML Schema Formalization

[March 21, 2001] W3C Publishes XML Schema Formalization. A communiqué from Matthew Fuchs (Commerce One) highlights the technical significance of W3C's recent XML Schema formalization, published in a W3C Working Draft as *XML Schema: Formal Description*. The document supplies formal a description of XML types and validity as specified by the recently-issued Proposed Recommendation XML Schema Part 1: Structures. From the Introduction: "This formalization is a formal, declarative system for describing and naming XML Schema information, specifying XML instance type information, and validating instances against schemas. The goals of the formalization are to: (1) Provide a semantic framework for software systems that use the W3C XML Schema specification, such as the W3C XML Query Algebra; (2) Specify names for all components of an XML Schema, so that they can be uniquely identified by URIs. Such unique identifiers may be useful to XML Query, RDF, and topic maps, among others; (3) Formally define validation at a declarative level; (4) Define the mapping from the current XML Schema syntax onto the structures described here, as well as the mapping between the XML Schema component mode and our component model. Many potential applications of XML Schema may benefit from the definition of a formal model. We have focused on the material in Part I (Structures), as this is the most complex; a basic understanding of first-order predicate logic, which is part of most computer science curricula, is adequate to understand this document." [*Full context*]

XML Schema Definition Language - W3C Recommendation

- [May 03, 2001] W3C XML Schema Published as a W3C Recommendation. The World Wide Web Consortium (W3C) has announced the publication of the W3C XML Schema specification as a W3C Recommendation. A W3C 'Recommendation' "indicates that a specification is stable, contributes to Web interoperability, and has been reviewed by the W3C Membership, who are in favor of supporting its adoption by academic, industry, and research communities. XML Schemas define shared markup vocabularies, the structure of XML documents which use those vocabularies, and provide hooks to associate semantics with them. With over two years of development and testing through implementation, XML Schema provides an essential piece for XML to reach its full potential. The XML Schema specification consists of three parts. One part defines a set of simple datatypes, which can be associated with XML element types and attributes; this allows XML software to do a better job of managing dates,

numbers, and other special forms of information. The second part of the specification proposes methods for describing the structure and constraining the contents of XML documents, and defines the rules governing schema-validation of documents. The third part is a primer, which explains what schemas are, how they differ from DTDs, and how someone builds a schema. XML Schema introduces new levels of flexibility that may accelerate the adoption of XML for significant industrial use. For example, a schema author can build a schema that borrows from a previous schema, but overrides it where new unique features are needed. XML Schema allows the author to determine which parts of a document may be validated, or identify parts of a document where a schema may apply. XML Schema also provides a way for users of ecommerce systems to choose which XML Schema they use to validate elements in a given namespace, thus providing better assurance in ecommerce transactions and greater security against unauthorized changes to validation rules. Further, as XML Schema are XML documents themselves, they may be managed by XML authoring tools, or through XSLT." [*Full context*

- In connection with the release of the Schema Recommendation, W3C has also provided for the creation of a W3C XML Schema Test Collection, announced by Henry S. Thompson (University of Edinburgh and W3C; Oriol Carbo, University of Edinburgh and W3C). "Goals and Objectives: The W3C XML Schema Test Collection work aims at coordinating test suites for W3C XML Schema processors created by different developers." The main objectives as announced 2001-05-02 are: (1) to integrate existing tests for W3C XML Schema processors in a common environment so they can be accessed publicly and shared among developers; (2) to establish a standard approach to test material IPR which meets the needs of both contributors and users; (3) to collect and develop tools to automate the execution and presentation of the test suites; (4) to offer a standard description of tests related to W3C XML Schema processors: [...]; (5) [to provide test descriptions] understandable by a developer without the need to actually view the test file(s) themselves); (6) to offer a standard presentation of test results; (7) to design additional tests and add/regularise descriptions of the existing tests; (8) in due course, to provide an XSLT-based approach to comparing XML representations of the post schema-validation infoset as produced by different processors; we will shortly announce the availability of XML Schemas for both the ordinary Infoset and the PSVInfoset. "The W3C expects to author only a small part of the collection -- we are

counting on Member organisations and others to contribute the majority. To offer materials for the collection, please send e-mail to www-xml-schema-tests@w3.org." Note from Henry Thompson: "...the www-xml-schema-tests@w3.org mailing address is *not* a mailing list; it's for potential contributors to use to initiate discussions about contributions. For *discussions* of testing, I don't think we need a new mailing list; I'd expect xmlschema-dev@w3.org to be used for discussing W3C XML Schema testing..."

- Announcement: "World Wide Web Consortium Issues XML Schema as a W3C Recommendation. Two Years of Development Produces Comprehensive Solution for XML Vocabularies."
- Testimonials for XML Schema Recommendation - From Altova, Inc., Commerce One, IBM, IPR Systems, Lotus Development Corporation, Microsoft Corporation, Oracle Corporation, Reuters, Inc., SAP AG, webMethods, and University of Edinburgh.
- XML Schema Part 1: Structures. W3C Recommendation 02-May-2001. [Default] namespace: `http://www.w3.org/2001/XMLSchema`. Latest version URL: http://www.w3.org/TR/xmlschema-1/. Edited by Henry S. Thompson (University of Edinburgh), David Beech (Oracle Corporation), Murray Maloney (for Commerce One), and Noah Mendelsohn (Lotus Development Corporation). Available in XML format (with XML DTD and XSL stylesheet); see also the separate XML Schema and XML DTD for Part 1. [cache .ZIP, local copy]
- XML Schema Part 2: Datatypes. W3C Recommendation 02-May-2001. Latest version URL: http://www.w3.org/TR/xmlschema-2/. Edited by Paul V. Biron (Kaiser Permanente, for Health Level Seven) and Ashok Malhotra (Microsoft, formerly of IBM). Available in XML; also with a schema for built-in datatypes only. [cache]
- XML Schema Part 0: Primer. W3C Recommendation 02-May-2001. Latest version URL: http://www.w3.org/TR/xmlschema-0/. Edited by David C. Fallside (IBM). Appendices include: A. Acknowledgements; B. Simple Types & Their Facets; C. Using Entities D. Regular Expressions E. Index. [cache]
- Namespaces. [Default] namespace: `http://www.w3.org/2001/XMLSchema`. Also: "To facilitate usage in specifications other than the XML Schema definition language, such as those that do not want to know anything about aspects of the XML Schema definition language other than the datatypes, each 'built-in' datatype is also defined in the namespace whose URI is: `http://`

www.w3.org/2001/XMLSchema-datatypes. This applies to both 'built-in primitive' and 'built-in derived' datatypes."

- Mail Archives for W3C XML Schema development 'xmlschema-dev@w3.org'. Send subscription requests to xmlschema-dev-request@w3.org.
- Mail Archives for 'www-xml-schema-comments'
- XML Schema type library - Sample. See the Primer (Part 0) for reference/description. [cache]
- Errata for W3C XML Schema Rec
- Translations of W3C XML Schema Rec

XML Schema Definition Language - Proposed Recommendation

[March 31, 2001] XML Schema. W3C Proposed Recommendation 30-March-2001. "This version of this Proposed Recommendation replaces that published on 16-March-2001. The only change from that draft is that the type there called 'number' is here renamed 'decimal'. This type was called 'decimal' up until the draft of 16 March 2001, so this change simply restores the original name of this type." The deadline for review of this document is Monday 16 April 2001. XML Schema Part 0: Primer; XML Schema Part 1: Structures; XML Schema Part 2: Datatypes.

[March 16, 2001] **W3C Publishes XML Schema as a Proposed Recommendation.** The W3C XML Schema specification has advanced to the 'Proposed Recommendation' stage, indicating that "the specification is stable and that implementation experience has been gathered, showing that each feature of the specification can be implemented." The three-part document has been produced as part of the W3C XML Activity. This PR version replaces the Candidate Recommendation of October 24, 2000. The deadline for review of the PR specification is Monday April 16, 2001. Review comments may be sent to the publicly archived 'xmlschema-dev' mailing list. As with the Candidate Recommendation, "The XML Schema specification consists of three parts. One part defines a set of simple datatypes, which can be associated with XML element types and attributes; this allows XML software to do a better job of managing dates, numbers, and other special forms of information. The second part of the specification proposes methods for describing the structure and constraining the contents of XML documents, and defines the rules governing schema-validation of documents. The third part is a primer, which explains what schemas are, how they differ from DTDs, and how someone builds a schema." [*Full context*]

See also:

- DTD changes in XML Schema from CR to PR. Provided by Robin LaFontaine, based on DeltaXML schema comparison software. See DeltaXML XML Schema [comparison] software. [source]
- [March 31, 2001] Revised Online Validator for XML Schema (XSV) and XML Schema Update Tool (XSU). Henry S. Thompson (HCRC Language Technology Group, University of Edinburgh) has announced a revised 'beta test' release of his XSV 'Validator for XML Schema' service and corresponding XSU update tool for XML Schema CR **->** PR. This version of XSV supports checking XML schema documents with the namespace URI `http://www.w3.org/2001/XMLSchema`, corresponding to the W3C Proposed Recommendation for XML Schema. XSV is an open source "GPLed work-in-progress attempt at a conformant schema-aware processor." The online XSV interface provides two forms for W3C XML schema checking: "(1) one for checking a schema which is accessible via the Web, and/or schema-validating an instance with a schema of your own; (2) another form for use if you are behind a firewall or have a schema to check which is not accessible via the Web." In addition to source code distributions (Python), the latest version of XSV is available in a self-installing package for Win32 platforms. The XSU transformation tool provides for automated update of XML Schema documents from the XML Schema 20000922 version to the Proposed Recommendation (20010316) version. It is a service in 'beta test' which "attempts to convert valid XML schema documents with the namespace URI `http://www.w3.org/2000/10/XMLSchema` to valid schema documents with the namespace URI `http://www.w3.org/2001/XMLSchema`" using a transform sheet. In this connection, the XSU developers request sample XML schemas to provide a testing pool: they ask that users grant permission to W3C to retain input of tested XML schemas (just tick the checkbox). [*Full context*]

XML Schema Definition Language - Candidate Recommendation

- [October 24, 2000] Testimonials for XML Schema Candidate Recommendation. [cache]
- [October 24, 2000] Announcement (in part): A W3C press release announces the publication of XML Schema as a W3C Candidate Recommendation. "The World Wide Web Consortium (W3C) has issued XML Schema as a W3C Candidate Recommendation. Advancement

of the document to Candidate Recommendation is an invitation to the Web development community at large to make implementations of XML Schema and provide technical feedback. Simply defined, XML Schemas define shared markup vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. 'Databases, ERP and EDI systems all know the difference between a date and a string of text, but before today, there was no standard way to teach your XML systems the difference. Now there is,' declared Dave Hollander, co-chair of the W3C XML Schema Working Group and CTO of Contivo, Inc. 'W3C XML Schemas bring to XML the rich data descriptions that are common to other business systems but were missing from XML. Now, developers of XML ecommerce systems can test XML Schema's ability to define XML applications that are far more sophisticated in how they describe, create, manage and validate the information that fuels B2B ecommerce.' By bringing datatypes to XML, XML Schema increases XML's power and utility to the developers of electronic commerce systems, database authors and anyone interested in using and manipulating large volumes of data on the Web. By providing better integration with XML Namespaces, it makes it easier than it has ever been to define the elements and attributes in a namespace, and to validate documents which use multiple namespaces defined by different schemas. XML Schema introduces new levels of flexibility that may accelerate the adoption of XML for significant industrial use. For example, a schema author can build a schema that borrows from a previous schema, but overrides it where new unique features are needed. his principle, called inheritance, is similar to the behavior of Cascading Style Sheets, and allows the user to develop XML Schemas that best suit their needs, without building an entirely new vocabulary from scratch. XML Schema allows the author to determine which parts of a document may be validated, or identify parts of a document where a schema may apply. XML Schema also provides a way for users of ecommerce systems to choose which XML Schema they use to validate elements in a given namespace, thus providing better assurance in ecommerce transactions and greater security against unauthorized changes to validation rules. Further, as XML Schema are XML documents themselves, they may be managed by XML authoring tools, or through XSLT. . . Candidate Recommendation is W3C's public call for implementation, an explicit invitation for W3C members and the developer community at large to review the XML Schema specification

and build their own XML Schemas. This period of implementations and reporting allows the editors to learn how developers outside of the Working Group might use them, and where there may be ambiguities for implementors. Public testing and implementation contribute to a more robust XML Schema, and to more widespread use." See the full text of the announcement: "World Wide Web Consortium Issues XML Schema as a Candidate Recommendation. Implementation testing the key to Interoperability." [cache]

- [October 24, 2000] *XML Schema Part 1: Structures*. W3C Candidate Recommendation 24-October-2000, edited by Henry S. Thompson (University of Edinburgh), David Beech (Oracle Corp.), Murray Maloney (for Commerce One), and Noah Mendelsohn (Lotus Development Corporation). Part 1 defines the XML Schema definition language, "which offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those which exploit the XML Namespace facility. The schema language, which is itself represented in XML 1.0 and uses namespaces, substantially reconstructs and considerably extends the capabilities found in XML 1.0 document type definitions (DTDs). This specification depends on *XML Schema Part 2: Datatypes*. Appendix A supplies a normative "Schema for Schemas"; Appendix F contains a non-normative "DTD for Schemas"; Appendix J gives brief summaries of the substantive changes to this specification since the public working draft of 7 April 2000.
- XSD for XML Schema, [cache]
- DTD for XML Schema, [cache]
- [October 24, 2000] *XML Schema Part 2: Datatypes*. W3C Candidate Recommendation 24-October-2000, exited by Paul V. Biron (Kaiser Permanente, for Health Level Seven) and Ashok Malhotra (IBM). Part 2 of the specification for the XML Schema language "defines facilities for defining datatypes to be used specifications. The datatype language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs) for specifying datatypes on elements and attributes." Appendix A provides the normative "Schema for Datatype Definitions" and Appendix B gives the non-normative "DTD for Datatype Definitions."
- [October 24, 2000] *XML Schema Part 0: Primer*. W3C Candidate Recommendation 24-October-2000, edited by David C. Fallside (IBM). "*XML Schema Part 0: Primer* is a non-normative document intended to provide an easily readable description of the XML Schema facilities and is

oriented towards quickly understanding how to create schemas using the XML Schema language. *XML Schema Part 1: Structures* and *XML Schema Part 2: Datatypes* provide the complete normative description of the XML Schema language -- this primer describes the language features through numerous examples which are complemented by extensive references to the normative texts."

- [October 24, 2000] Commentary. See the longer memo from Henry S. Thompson (Janet Daly) with an explanation of why (I18N) WG dissented from the specification's treatment of dates and times, and the CR exit criteria. Also, in connection with this CR publication, Henry Thompson announced the availablility of a self-installing version of XSV, the W3C/University of Edinburgh XML Schema validator; 'WIN32 for now, UN*X coming soon'.

XML Schema Definition Language - Seventh Working Draft

- [September 22, 2000] *XML Schema Part 1: Structures*. Reference: W3C Working Draft 22-September-2000, edited by Henry S. Thompson (University of Edinburgh), David Beech (Oracle Corp.), Murray Maloney (for Commerce One), and Noah Mendelsohn (Lotus Development Corporation). The most important changes for the 07-September-2000 release are found in this *Structures* document. The non-normative 'Appendix H' in the *Structures* document supplies a "Description of changes" to the working draft since the previous public version of 07-April-2000. Some eighteen (18) changes are identified here. For example: [H1] "'Equivalence classes' have been renamed 'substitution groups', to reflect the fact that their semantics is not symmetrical; [H2] "The content model of the `complexType` element has been significantly changed, allowing for tighter content models and a better fit between the abstract component and its XML Representation"; [H3] "Empty content models are now signalled by an explicit empty content particle, mixed content by specifying the value true for the mixed attribute on `complexType` or `complexContent`; [H6] "A new form of schema composition operation, similar to that provided by `include` but allowing constrained redefinition of the included components has been added, using a `redefine` element"; [H8] "The defaulting for the `minOccurs` and `maxOccurs` attributes of `element` has been simplified: it is now *1* in both cases, with no interdependencies"; [H9] "The content model for the `group` element when it occurs at the top level has been tightened, to allow only a single `all`, `choice`, `group`, or `sequence` child"; [H13]

"Abstract types in element declarations are now allowed." See the main news entry and editorial notes provided in Henry Thompson's announcement 'New Pre-CR Public Working Drafts of XML Schema Released'. [cache]

- [September 22, 2000] *XML Schema Part 2: Datatypes*. Reference: W3C Working Draft 22-September-2000, edited by Paul V. Biron (Kaiser Permanente, for Health Level Seven) and Ashok Malhotra (IBM). "*XML Schema: Datatypes* is part 2 of a two-part draft of the specification for the XML Schema definition language. "This document proposes facilities for defining datatypes to be used in XML Schemas as well as other XML specifications. The datatype language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs) for specifying datatypes on elements and attributes." [cache]
- [September 22, 2000] *XML Schema Part 0: Primer*. Reference: W3C Working Draft, 22-September-2000, edited by David C. Fallside (IBM). "XML Schema Part 0: Primer is a non-normative document intended to provide an easily readable description of the XML Schema facilities and is oriented towards quickly understanding how to create schemas using the XML Schema language." [cache]

XML Schema Definition Language - Sixth Working Draft

- [April 07, 2000] *XML Schema Part 0: Primer*. W3C Working Draft 07-April-2000.
- [April 07, 2000] *XML Schema Part 1: Structures*. W3C Working Draft 07-April-2000. Last Call Ends 12 May 2000.
- [April 07, 2000] *XML Schema Part 2: Datatypes*. W3C Working Draft 07-April-2000. Last Call Ends 12 May 2000.

XML Schema Definition Language - Fifth Working Draft

- [February 26, 2000] *XML Schema Part 0: Primer*. Reference: W3C Working Draft, 25-February-2000, edited by David C. Fallside (IBM). The primer has been issued in conjunction with a new working draft [25 February 2000] of the normative tomes on XML Schema Structures and XML Schema Datatypes. Abstract: *XML Schema Part 0: Primer* is a non-normative document intended to provide an easily readable description of the XML Schema facilities and is oriented towards quickly understanding how to create schemas

using the XML Schema language. XML Schema Part 1: Structures and XML Schema Part 2: Datatypes provide the complete normative description of the XML Schema definition language, and the primer describes the language features through numerous examples which are complemented by extensive references to the normative texts. This 'Second Torah' commentary is officially a part of the W3C XML Activity. Discrepancies between the sacred text and its commentary are noted in the Primer. Description: "This document, *XML Schema Part 0: Primer*, provides an easily approachable description of the XML Schema definition language, and should be used alongside the formal descriptions of the language contained in Parts 1 and 2 of the XML Schema specification. The intended audience of this document includes application developers whose programs read and write schema documents, and schema authors who need to know about the features of the language, especially features that provide functionality above and beyond what is provided by DTDs. The text assumes that you have a basic understanding of XML 1.0 and XML-Namespaces. Each major section of the primer introduces new features of the language, and describes the features in the context of concrete examples. Section 2 covers the basic mechanisms of XML Schema. It describes how to declare the elements and attributes that appear in XML documents, the distinctions between simple and complex types, defining complex types, the use of simple types for element and attribute values, schema annotation, a simple mechanism for re-using element and attribute definitions, and null values. Section 3 covers some of XML Schema's advanced features, and in particular, it describes mechanisms for deriving types from existing types, and for controlling these derivations. The section also describes mechanisms for merging together fragments of a schema from multiple sources, and for element substitution. Section 4 covers more advanced features, including a powerful mechanism for specifying uniqueness among attributes and elements, a mechanism for using types across namespaces, a mechanism for extending types based on namespaces, and a description of how documents are checked for conformance. In addition to the sections just described, the primer has a number of appendices that contain detailed reference information on simple types and an associated regular expression language. The primer is a non-normative document, which means that it does not provide a definitive (from the W3C's point of view) specification of the XML Schema language."

- [February 26, 2000] *XML Schema Part 1: Structures*. Reference: W3C Working Draft 25-

February-2000; edited by Henry S. Thompson (University of Edinburgh), David Beech (Oracle Corp.), Murray Maloney (Commerce One), and Noah Mendelsohn (Lotus Development Corporation). The specification itself is available in XML as well as HTML format. The release includes a formal description of schema 'structures' facilities in schema and in XML DTD notation. "Following a period of review and polishing, it is the WG's intent to issue a Last Call for Review by other W3C working groups sometime during March, 2000, and to submit this specification thereafter for publication as a Candidate Recommendation." Document abstract: "*XML Schema: Structures* specifies the XML Schema definition language, which offers facilities for describing the structure and constraining the contents of XML 1.0 documents. The schema language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs). This specification depends on *XML Schema Part 2: Datatypes*." Description: "The purpose of *XML Schema: Structures* is to define the nature of XML schemas and their component parts, provide an inventory of XML markup constructs with which to represent schemas, and define the application of schemas to XML documents. The purpose of an *XML Schema: Structures* schema is to define and describe a class of XML documents by using schema components to constrain and document the meaning, usage and relationships of their constituent parts: datatypes, elements and their content and attributes and their values. Schemas may also provide for the specification of additional document information, such as default values for attributes and elements. Schemas have facilities for self-documentation. Thus, *XML Schema: Structures* can be used to define, describe and catalogue XML vocabularies for classes of XML documents. Any application that consumes well-formed XML can use the *XML Schema: Structures* formalism to express syntactic, structural and value constraints applicable to its document instances. The *XML Schema: Structures* formalism allows a useful level of constraint checking to be described and validated for a wide spectrum of XML applications. However, the language defined by this specification does not attempt to provide all the facilities that might be needed by any application. Some applications may require constraint capabilities not expressible in this language, and so may need to perform their own additional validations."

- [February 26, 2000] *XML Schema Part 2: Datatypes*. Reference: W3C Working Draft 25-February-2000; edited by Paul V. Biron (Kaiser

Permanente, for Health Level Seven) and Ashok Malhotra (IBM). The release includes separate documents containing the corresponding formal schema notation, an XML DTD, and schema for built-in datatypes only; there is an XML version as well. *XML Schema: Datatypes* is part 2 of a two-part draft of the specification for the XML Schema definition language. This document proposes facilities for defining datatypes to be used in XML Schemas and other XML specifications. The datatype language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs) for specifying datatypes on elements and attributes. Rationale for Part 2: "... validity constraints exist on the content of [XML document] instances that are not expressible in XML DTDs. The limited datatyping facilities in XML have prevented validating XML processors from supplying the rigorous type checking required in these situations. The result has been that individual applications writers have had to implement type checking in an ad hoc manner. This specification addresses the need of both document authors and applications writers for a robust, extensible datatype system for XML which could be incorporated into XML processors. As discussed below, these datatypes could be used in other XML-related standards as well." The concrete requirements to be fulfilled by this specification are articulated in the XML Schema Requirements document; it states that the XML Schema Language must: (1) provide for primitive data typing, including byte, date, integer, sequence, SQL & Java primitive data types, etc.; (2) define a type system that is adequate for import/export from database systems (*e.g.*, relational, object, OLAP); (3) distinguish requirements relating to lexical data representation vs. those governing an underlying information set; (4) allow creation of user-defined datatypes, such as datatypes that are derived from existing datatypes and which may constrain certain of its properties (*e.g.*, range, precision, length, format). "Although the Working Group does not anticipate further substantial changes to the functionality described here, this is still a working draft, subject to change based on experience and on comment by the public and other W3C working groups. Following a period of review and polishing, it is the WG's intent to issue a Last Call for Review by other W3C working groups sometime during March, 2000, and to submit this specification thereafter for publication as a Candidate Recommendation."

XML Schema Definition Language - Fourth Working Draft

- [December 17, 1999] *XML Schema Part 1: Structures* (W3C Working Draft 17-December-1999). Edited by Henry S. Thompson (University of Edinburgh), David Beech (Oracle Corp.), Murray Maloney (Commerce One), and Noah Mendelsohn (Lotus Development Corporation). *XML Schema: Structures* represents "part 1 of a two-part draft of the specification for the XML Schema definition language. This document proposes facilities for describing the structure and constraining the contents of XML 1.0 documents. The schema language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs)." The purpose of a *Structures* schema document is to "define and describe a class of XML documents by using these constructs to constrain and document the meaning, usage and relationships of their constituent parts: datatypes, elements and their content, attributes and their values. Schema constructs may also provide for the specification of additional information such as default values. Schemas are intended to document their own meaning, usage, and function through a common documentation vocabulary. Thus, XML Schema: Structures can be used to define, describe and catalogue XML vocabularies for classes of XML documents." [local archive copy]
- XML schema - 'Schema for Schemas'; [local archive copy]
- XML DTD - 'DTD for Schemas'; [local archive copy]
- *XML Schema: Structures* in XML format
- [December 20, 1999] Henry S. Thompson has made available the slides and some additional documentation from an intensive (full-day) tutorial session on the XML Schema Definition Language, presented at XML '99 in Philadelphia. The materials have been updated to match the 1999-12-17 versions of the Schema WD documents. (1) presentation slides in HTML, and (2) additional materials in HTML. Original/canonical formats: Powerpoint slides [local archive copy] and additional notes (.doc) [local archive copy]. Announced on XML-DEV 20-Dec-1999.
- dtddiff of 1999-12-17 schema proposal *vis-à-vis* 1999-11 WD. "Below is the result of running Earl Hood's dtddiff script to compare the DTD's from appendix B of the November and December drafts of the W3C Schema Proposal." From Bob DuCharme
- *XML Schema Part 2: Datatypes* (W3C Working

Draft 17-December-1999) has been edited by Paul V. Biron (Kaiser Permanente, for Health Level Seven) and Ashok Malhotra (IBM). *XML Schema: Datatypes* presents part 2 of a two-part draft of the specification for the XML Schema definition language. This document proposes facilities for defining datatypes to be used in XML Schemas and other XML specifications. The datatype language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs) for specifying datatypes on elements and attributes." It is well known that "validity constraints exist on the content of [XML document] instances that are not expressible in XML DTDs. The limited datatyping facilities in XML have prevented validating XML processors from supplying the rigorous type checking required in these situations. The result has been that individual applications writers have had to implement type checking in an *ad hoc* manner. This [Datatypes] specification addresses the need of both document authors and applications writers for a robust, extensible datatype system for XML which could be incorporated into XML processors." At the moment, these datatypes can be specified only "for element content that would be specified as #PCDATA and attribute values of various types in a DTD." [local archive copy]

- XML schema - Schema for datatypes; [local archive copy]
- XML DTD - DTD for datatypes; [local archive copy]
- *XML Schema: Datatypes* in XML Format

XML Schema Definition Language - Third Working Draft

- *XML Schema Part 1: Structures*. References: W3C Working Draft 5-November-1999, edited by Henry S. Thompson (University of Edinburgh), David Beech (Oracle Corp.), Murray Maloney (Commerce One), and Noah Mendelsohn (Lotus Development Corporation).
- Structures - HTML Format
- Structures - XML format
- Structures - XML Schema
- Structures - XML DTD
- Comments to www-xml-schema-comments@w3.org; and the mail archive
- dtddiff of XSDL November and September drafts, supplied by Bob DuCharme.
- *XML Schema Part 2: Datatypes*. References: W3C Working Draft 05-November-1999, edited by Paul V. Biron (Kaiser Permanente, for Health

Level Seven) and Ashok Malhotra (IBM).

- Datatypes - XML format
- Datatypes - XML Schema
- Datatypes - XML DTD

XML Schema Definition Language - Second Working Draft

- ***XML Schema Part 1: Structures***. Reference: W3C Working Draft 24-September-1999, edited by Henry S. Thompson (University of Edinburgh), David Beech (Oracle), Murray Maloney (Commerce One), and Noah Mendelsohn (Lotus). Part 1 "proposes facilities for describing the structure and constraining the contents of XML 1.0 documents. The schema language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs). . . The purpose of the *Structures* specification is to provide an inventory of XML markup constructs with which to write schemas." Such a schema is used "to define and describe a class of XML documents by using these constructs to constrain and document the meaning, usage and relationships of their constituent parts: datatypes, elements and their content, attributes and their values, entities and their contents and notations. Schema constructs may also provide for the specification of additional information such as default values. Schemas are intended to document their own meaning, usage, and function through a common documentation vocabulary. Thus, the *Structures* specification can be used to define, describe and catalogue XML vocabularies for classes of XML documents. Any application that consumes well-formed XML can use the XML Schema: Structures formalism to express syntactic, structural and value constraints applicable to its document instances. The *Structures* formalism will allow a useful level of constraint checking to be described and validated for a wide spectrum of XML applications. However, the language defined by this specification does not attempt to provide *all* the facilities that might be needed by *any* application. Some applications may require constraint capabilities not expressible in this language, and so may need to perform their own additional validations."

    - Structures - HTML Version, [local archive copy]
    - XML format, [local archive copy]
    - XML schema, [local archive copy]
    - XML DTD, [local archive copy]

- Comments to:www-xml-schema-comments@w3.org
- Comments public archive.
- *XML Schema Part 1: Structures - Proposed Updates from Simplification Task Force*. 13-August-1999. Based on W3C Working Draft 19-July-1999. Taskforce members: D. Beech, P. Biron, A. Brown, P. Chen, D. Fallside (ed), M. Fuchs, M. Murata, J. Robie. "The updates proposed in this document take the form of new text to replace text currently existing in sections 2, 3 and Appendix B of the 19 July XML Schema: Structures working draft. Within sections 2 and 3 of this proposal, subsections indicated by only headings are assumed to contain the text of the July 19 draft, although note that the heading itself of section 2.4 is updated." [local archive copy]
- *Proposal by the "Simple Syntax" Taskforce*. August 13, 1999. Taskforce members: D. Beech, P. Biron, A. Brown, P. Chen, D. Fallside (ed), M. Fuchs, M. Murata, J. Robie. "This paper describes the recommendation of the "Simple Syntax" Taskforce for simplifying the syntax of element definitions, type definitions, and element usage within content models. It provides a syntax description, examples, in addition to a summary of alternatives explored. A companion document XML Schema Part 1: Structures - Proposed Updates from Simplification Taskforce proposes new text for the Structures working draft." [local archive copy]
- Note on the 2nd WD, by Henry S. Thompson
- dtddiff of XSDL 5/99 and 9/99 - Provided by Bob DuCharme, XML-DEV.

- ***XML Schema Part 2: Datatypes***. Reference: W3C Working Draft 24-September-1999, edited by Paul V. Biron (Kaiser Permanente, for Health Level Seven) and Ashok Malhotra (IBM). The *Datatypes* document "specifies a language for defining datatypes to be used in XML Schemas and possibly elsewhere." As explained in the W3C's XML Schema Requirements document, the *Datatypes* design is motivated by the recognition that "document authors, including authors of traditional documents and those transporting data in XML, often require a high degree of type checking to ensure robustness in document understanding and data interchange." In many cases, "validity constraints exist on the content of

the [XML] instances that are not expressible in XML DTDs. The limited datatyping facilities in XML have prevented validating XML processors from supplying the rigorous type checking required in these situations. The result has been that individual applications writers have had to implement type checking in an ad hoc manner. This [*Datatypes*] specification addresses the need of both document authors and applications writers for a robust, extensible datatype system for XML which could be incorporated into XML processors." The facilities in the *Datatypes* WD have been designed in light of the formal requirements, which stipulate that the XML Schema Language must: (1) provide for *primitive data typing*, including byte, date, integer, sequence, SQL & Java primitive data types, etc.; (2) define a *type system* that is adequate for import/export from database systems (*e.g.*, relational, object, OLAP); (3) distinguish requirements relating to lexical data representation vs. those governing an underlying information set; and (4) allow creation of *user-defined datatypes*, such as datatypes that are derived from existing datatypes and which may constrain certain of its properties (*e.g.*, range, precision, length, format)."

- Datatypes - HTML Version, [local archive copy]
- XML format, [local archive copy]
- XML schema, [local archive copy]
- XML DTD, [local archive copy]
- Comments to:www-xml-schema-comments@w3.org
- Comments public archive.

XML Schema Definition Language - First Working Draft

- ***XML Schema Part 1: Structures.*** W3C Working Draft 6-May-1999. Edited by David Beech (Oracle), Scott Lawrence (Agranat Systems), Murray Maloney (Commerce One), Noah Mendelsohn (Lotus), and Henry S. Thompson (University of Edinburgh). Part 1 'proposes facilities for associating datatypes with XML element types and attributes; this will allow XML software to do a better job of managing dates, numbers, and other special forms of information.'

  - HTML Version, [local archive copy]
  - XML Version; [local archive copy]
  - Accompanying schema, [local archive copy]
  - Accompanying DTD, [local archive copy]

- Press Release: "World Wide Web Consortium Releases First Working Drafts of XML Schema Specification. W3C Members Collaborate to Improve and Standardize Needed Technology." [local archive copy]
- Send comments to: www-xml-schema-comments@w3.org
- Comments archived

- ***XML Schema Part 2: Datatypes.*** World Wide Web Consortium Working Draft 06-May-1999. Edited by Paul V. Biron (Kaiser Permanente, for Health Level Seven) and Ashok Malhotra (IBM). Part 2 of the specification 'proposes methods for describing the structure and constraining the contents of XML documents.'

  - HTML Version; [local archive copy]
  - Accompanying schema, [local archive copy]
  - Accompanying DTD, [local archive copy]
  - Send comments to: www-xml-schema-comments@w3.org
  - Comments archived

Related XML Schema Proposals

XML-Data (XDR)

- XML-Data. W3C Note 05-Jan-1998. "This paper describes an XML vocabulary for schemas, that is, for defining and documenting object classes. It can be used for classes which as strictly syntactic (for example, XML) or those which indicate concepts and relations among concepts (as used in relational databases, KR graphs and RDF). The former are called 'syntactic schemas;' the latter 'conceptual schemas'."
- XML-Data Reduced. Draft Version 3-July-1998, version 0.21. By Charles Frankston (Microsoft) and Henry S. Thompson (University of Edinburgh). "The XML-Data submission contained many new ideas that an XML schema language could support. This document refines and subsets those ideas down to a more manageable size in order to allow faster progress toward adopting a new schema language for XML. Some of the inconsistencies in the XML-Data submission are cleaned up, and some changes have been made based on comments received since the XML-Data submission was posted. This note is a refinement

of the January 1998 XML-Data submission http://www.w3.org/TR/1998/NOTE-XML-data-0105/." [local archive copy]

- "[Microsoft] XML Schema Developer's Guide" Andrew Layman [XML-DEV 18-May-1999]: "'XML-Data Reduced' (XDR) refers to a trimmed and improved version of the XML-Data schema syntax. The original XML-Data submission can be found at http://www.w3.org/TR/1998/NOTE-XML-data/. Information on XML-Data Reduced is at http://msdn.microsoft.com/xml/XMLGuide/schema-overview.asp'." Note: 'XDR' in this context not to be confused with XDR: External Data Representation Standard (Network Working Group, Request for Comments: 1832, August 1995).
- [February 09, 2001] Microsoft XDR-XSD Converter. An XDR-XSD Converter is available for download from Microsoft's [XML] MSDN Online Development Center. The transformation tool is made available in its current beta state for experimentation purposes only. "This XSLT stylesheet transforms XML Data Reduced (XDR) Schemas as supported in MSXML to XML Schemas -- conformant to the W3C XML Schema Candidate Recommendation of October 25, 2000. MSXML 3.0 itself does not support the W3C XML Schema format. Notes on XDR-XSD Converter: XDR schemas using open content models allow more attribute extensions than the XML Schema resulting from this style sheet. Specifically, under XDR, when `model="open"` attributes from the target namespace may be added to an element, as long as they conform to the validity constraints for that attribute. Attributes from other namespaces may be added to an element, whether or not there are validity constraints for those attributes. It is not possible in XSD to treat attribute validation differently for attributes from the target namespace (`<xsd:anyAttribute namespace="##targetNamespace" processContents="lax"/>`) than for attributes from other namespaces (`<xsd:anyAttribute namespace="##other" processContents="strict"/>`). This is represented in the XML Schema DTD as only allowing one `<xsd:anyAttribute/>` element. The transformed XSD schema will not allow attributes from the target namespace. You may want to adjust the resulting schema to accommodate for this, by either not requiring attributes from the target namespace to be validated, or by adding the allowed attributes explicitly to the content model." The developers welcome feedback on problems encountered or other suggestions that would help them improve the transformation tool.

Document Content Description (DCD)

- DCD. NOTE-dcd-19980731, Submission to the World Wide Web Consortium 31-July-1998. "This document proposes a structural schema facility, Document Content Description (DCD), for specifying rules covering the structure and content of XML documents. The DCD proposal incorporates a subset of the XML-Data Submission and expresses it in a way which is consistent with the ongoing W3C RDF (Resource Description Framework) effort; in particular, DCD is an RDF vocabulary. DCD is intended to define document constraints in an XML syntax; these constraints may be used in the same fashion as traditional XML DTDs. DCD also provides additional properties, such as basic datatypes."

Schema for Object-oriented XML (SOX)

- SOX. Schema for Object-oriented XML. NOTE-SOX-19980930, Submitted to W3C 19980915. "This document proposes a schema facility, Schema for Object-oriented XML (SOX), for defining the structure, content and semantics of XML documents to enable XML validation and higher levels of automated content checking. The SOX proposal is informed by the XML 1.0 [XML] specification as well as the XML-Data submission (XML-Data), the Document Content Description submission (DCD) and the EXPRESS language reference manual (ISO-10303-11). SOX provides an alternative to XML DTDs for modeling markup relationships to enable more efficient software development processes for distributed applications."
- "SOX was created by Commerce One in anticipation of the coming W3C standard XML Schema Language. SOX leverages the object-relationships between data structures to allow for the easy management of a component library, and the use of type relationships within schema-based e-commerce applications. It also provides strong datatyping capabilities, like other XML Schema languages..."
- References:
  -
  - SOX Tutorial. "Covers the basic and advanced features of the SOX language, including namespaces, inheritance, and polymorphism." [cache]
  - About SOX
  - FAQ Document

Document Definition Markup Language (DDML)

- DDML. Document Definition Markup Language (DDML) Specification, Version 1.0. NOTE-ddml-19990119, W3C Note, 19-Jan-1999. "This document proposes Document Definition Markup Language (DDML), a schema language for XML documents. DDML [previously 'XSchema'] encodes the logical (as opposed to physical) content of DTDs in an XML document. This allows schema information to be explored and used with widely available XML tools. DDML is deliberately simple, providing an initial base for implementations. While introducing as few complicating factors as possible, DDML has been designed with future extensions, such as data typing and schema reuse, in mind."

Schematron

See the separate document for information on Schematron. Development takes place on SourceForge. [1]

Datatypes for DTDs (DT4DTD)

- [January 13, 2000] The W3C has acknowledged a submission request from Extensibility, Inc for publication of a NOTE: *Datatypes for DTDs (DT4DTD) 1.0*. Reference: W3C Note 13-January-2000, by Lee Buck (Extensibility), Charles F. Goldfarb (*The XML Handbook*), and Paul Prescod (ISOGEN International). The document abstract: "The presented specification allows legacy systems that may presently be unable to convert their DTD markup declarations to XML Schema, to utilize XML Schema conformant datatypes. With it, DTD creators can specify datatypes for attribute values and data content, thereby providing the foundation for a smoother future transition path. **NOTE:** Free open-source code that supports this specification for both SAX and DOM is available at www.extensibility.com/dt4dtd." [And:] "XML 1.0, using DTDs, provides a strong foundation for validating the syntax of a document and ensuring that all the necessary pieces of information are present (*i.e.* necessary elements are included, inappropriate ones are not, attributes are supplied when required, etc.). DTDs do not, however, offer much help in constraining the value of a particular attribute or element: *a.k.a.* datatypes to those with programming backgrounds. DT4DTD brings this important capability to XML. Specifically it: (1) Provides

compatibility with XML Schema data types; (2) Provides compatibility with XML-Data data types; (3) Provides programmatic extensions for DOM and SAX; (4) Provides an extensible architecture for custom datatypes; (5) Provides runtime support for data typed schemas created in XML Authority... The techniques specified in DT4DTD are already in commercial use in several places, including the Financial Products Markup Language from JPMorgan and PriceWaterhouseCoopers, and versions of the Common Business Library from Commerce One, among others. Design-time support for DT4DTD is also available in leading commercial schema design tools such as XML Authority, which is produced by this submitting member organization." See also the W3C staff comment from Dan Connolly, which reads (in part): "W3C is pleased to receive the DT4DTD submission from Extensibility. During this transitional period of the W3C XML Activity while the XML Schema Working Group develops and deploys their work, this submission provides a valuable mechanism for addressing the lack of data types such as integer, date, etc. in XML 1.0 DTD syntax in a way that is compatible with legacy systems. However, it relies on a global convention for the interpretation of the unqualified names `e-dtype` and `a-dtype`, while use of XML Namespaces would make this unnecessary..." Local archive copy: NOTE, W3C comment, acknowledgement.

- [December 01, 1999] Datatypes for DTDs. A draft specification residing on the Extensibility Web site (and looking suspiciously like a W3C 'NOTE') proposes a mechanism that allows the declaration of datatypes for XML content (PCDATA) and attributes. To wit: *Datatypes for DTDs (DT4DTD)* Version 1.0 (November 1999) is a 'Public Specification' edited by Lee Buck (Extensibility), Charles F. Goldfarb, and Paul Prescod (ISOGEN International). Initially posted as a public document: '10/31/99'. "Abstract: The presented specification allows legacy systems that may presently be unable to convert their DTD markup declarations to XML Schema, to utilize XML Schema conformant datatypes. With it, DTD creators can specify datatypes for attribute values and data content, thereby providing the foundation for a smoother future transition path... Free open-source code that supports this specification for both SAX and DOM is available at www.extensibility.com/dt4dtd." According to one of the authors, the specification represents just "a little convention for getting around the limitations of notations as applied to attributes and contents for datatyping." The markup facility uses NOTATIONs, and relies upon two 'fixed' attributes, `e-dtype` and `a-dtype`. Declaring a

datatype for an element is permitted only if associated element type's content allows data but no sub-elements. The background: "XML 1.0, using DTDs, provides a strong foundation for validating the syntax of a document and ensuring that all the necessary pieces of information are present (*i.e.*, necessary elements are included, inappropriate ones are not, attributes are supplied when required, etc.). DTDs do not, however, offer much help in constraining the value of a particular attribute or element: a.k.a. datatypes to those with programming backgrounds. DT4DTD brings this important capability to XML. Specifically it: (1) Provides compatibility with XML Schema data types; (2) Provides compatibility with XML-Data data types; (3) Provides programmatic extensions for DOM and SAX; (4) Provides an extensible architecture for custom datatypes; (5) Provides runtime support for data typed schemas created in XML Authority. The DT4DTD [package] consists of a two major parts: a) The draft specification, and b) The SDK [in Java]." **Note:(?)** similar datatype declaration mechanisms (appropriate for 'architecture-engine' and 'notation-engine' processing aka 'handwaving') are available in SGML, and particularly in Web SGML, where the added 'data specification declared value' allows arguments to be passed to notation processors.

- DT4DTD Draft Specification, October/November 1999 (local archive copy)

Document Structure Description (DSD)

- [February 24, 2000] A recent communiqué from Anders Møller reports on a new release of the Document Structure Description (DSD) specification in which free source code and Win32 executables are available for download. The Document Structure Description (DSD) is an "XML schema language developed by AT&T Labs Research and BRICS, University of Aarhus. DSDs require no specialized XML/SGML insights. The technology is based on simple, general, and familiar concepts that allow much stronger document descriptions than possible with DTDs or the current XML Schema proposal. DSD provides an alternative to XML DTDs and other XML schema languages. It adds expressive power, increases readability, and contains support for default attributes and contents. Furthermore, it guarantees linear time processing in the size of the application document. The relationship between DSDs and XML Schema is briefly described in a FAQ document."
- [November 19, 1999] Document Structure Description (DSD): An XML Schema Language. A communiqué from Nils Klarlund forwards an

announcement for Document Structure Description (DSD) as "a new and very effective way of describing XML documents." According to the text of the announcement, "This new DSD schema language is result of a research collaboration between AT&T Labs, NJ and BRICS at the University of Aarhus, Denmark. The DSD 1.0 language has been designed by Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. A prototype DSD processor has been implemented. It is freely available for experimentation and further development. The DSD language has arisen out of a need to describe XML documents to Web programmers with an elementary background in computer science. DSDs have also been expressively designed to further W3C sponsored XML technologies such as Cascading Style Sheets (CSS) and XSL Transformations (XSLT). CSS is an essential part of modern HTML, but has so far not been formulated as a general style sheet mechanism for XML that works with any semantic domain. DSDs provide both a generalized semantics for a CSS-like style sheet mechanism and document processing instructions that provide the abstraction benefits of CSS in any XML document. XSLT 1.0 is a programming notation that allows transformations of classes of XML documents into semantic domains like HTML. XSLT programs are easy to write, especially if assumptions can be made about the input documents. The expressive power of DSDs allow declarative and readable specifications of XML documents that are to be subjected to XSLT processing. DSDs require no specialized XML/SGML insights. The technology is based on general and familiar concepts that allow much stronger document descriptions than possible with DTDs or the current XML Schema proposal." The Web site references the DSD 1.0 specification, an overview article, and a DSD description of DSDs.

REgular LAnguage description for XML (RELAX)

See the separate document. [RP]

TREX (Tree Regular Expressions for XML)

Information on "Tree Regular Expressions for XML (TREX)" is provided in a separate document. On 2001-04-24, James Clark posted a clarification confirming that he and Murata-san were working on the unification of TREX and RELAX.

RELAX NG

RELAX Core and TREX (Tree Regular Expressions for XML) are to be unified, since the two are very similar as structure-validation languages. The unified TREX/RELAX language is called RELAX NG [for "Relax Next Generation," pronounced "relaxing"]. This design work is now being conducted within the OASIS TREX Technical Committee, where a (first) specification is expected by July 1, 2001. The OASIS TC has also been renamed 'RELAX NG' [mailing list: 'relax-ng@lists.oasis-open.org'] to reflect the new name of the unified TREX/RELAX language. The RELAX NG development team plans to submit the OASIS specification to ISO, given the importance of ISO standards in Europe. See Minutes for RELAX NG TC 2001-05-17 and RELAX Core as ISO TR.

Examplotron

The Examplotron specification edited by Eric van der Vlist (Dyomedea) uses instance documents "as a lightweight schema language -- eventually adding the information needed to guide a validator in the sample documents. Examplotron may be used either as a validation language by itself, or to improve the generation of schemas expressed using other XML schema languages by providing more information to the schema translators. An XSLT transformation [in `compile.xsl`] transforms examplotron documents into XSLT sheets that validate 'similar' documents."

References:

- Examplotron web site
- Version 0.4 specification [cache]
- Version 0.4 XSLT file cache]
- [June 10, 2003] "Introducing Examplotron: The Fastest Road to Schema." By Uche Ogbuji (Principal Consultant, Fourthought, Inc). From IBM developerWorks, XML zone. June 10, 2003.

Hook: A One-element validation language

Hook: A One-Element Language for Validation of XML Documents based on Partial Order. By Rick Jelliffe, Academia Sinica Computing Centre. "The Hook validation language is a thought experiment in minimalism in XML schema languages. The purpose of such a minimal language would be to provide useful but ultra-terse success/fail validation for basic incoming QA, especially of datagrams. It is like a checksum for a schema. The validation it performs can be characterized as "Does this element have a feasible name, ancestry,

previous-siblings and contents?", there being some tradeoff between the how fully the later criteria are tested..." [cache]

Document Schema Definition Language (DSDL)

Information in a separate document: "Document Schema Definition Language (DSDL)."

[June 12, 2001] Document Schema Definition Language (DSDL) Proposed as ISO New Work Item.

Other XML Schema Language References

[*These references need reorganization - soon, I hope. - rcc*]

- [January 23, 2004] IBM and X-Hive Present XML Schema API as a W3C Member Submission. W3C has acknowledged receipt of a member submission entitled *XML Schema API*. The technology was submitted by from IBM Corporation and X-Hive Corporation B.V. and provides API access to properties of the XML Information Set. Specifically, the document "defines an XML Schema API, a platform- and language-neutral interface that allows programs and scripts to dynamically access and query the post-schema-validation infoset (PSVI) defined in the Normative Appendix C (Outcome Tabulations) of the W3C Recommendation *XML Schema Part 1: Structures*, "C.2 Contributions to the post-schema-validation infoset." The specification is implemented in Apache Xerces2 Java Parser; there is also a C++ binding and implementation for this specification in Apache Xerces C++ Parser." Section 1.2 defines interfaces which allow developers to access the XML Schema components which follow as a consequence of validation and/or assessment; Section 1.3 defines a set of interfaces for accessing the PSVI from an instance document; Section 1.4 defines a set of interfaces for loading XML Schema documents. The W3C Staff comment on the *XML Schema API* notes that the proposal "provides a substantial and useful addition to the DOM API, or to other existing event/pull parsing APIs such SAX/XNI. However, the XML Query and XSL Working Groups have been working on extending the work of XML Schema and adding more properties, and

are working on a new version of XML Schema 1.1; therefore, while the proposal addresses today's needs, it should be noted that future extensions will still be needed to follow additions to the XML Architecture."

- [December 23, 2003] "XML and Information Integration: Conceptual Modeling of XML Schemas." By Bernadette Farias Lóscio, Ana Carolina Salgado, and Luciano do Rêgo Galvão (Centro de Informática, Universidade Federal de Pernambuco, Brasil). In *Proceedings of the Fifth International Workshop on Web Information and Data Management (WIDM 2003)* (November 7-8, 2003). "XML has become the standard format for representing structured and semi-structured data on the Web. To describe the structure and content of XML data, several XML schema languages have been proposed. Although being very useful for validating XML documents, an XML schema is not suitable for tasks requiring knowledge about the semantics of the represented data. For such tasks it is better to use a conceptual schema. This paper presents an extension of the Entity Relationship (ER) model, called X-Entity, for conceptual modeling of XML schemas. We also present the process of converting a schema, defined in the XML Schema language, to an X-Entity schema. The conversion process is based on a set of rules that consider element declarations and type definitions and generates the corresponding conceptual elements. Such representation provides a cleaner description for XML schemas by focusing only on semantically relevant concepts. The X-Entity model has been used in the context of a Web data integration system with the goal of providing a concise and semantic description for local schemas defined in XML Schema... The X-Entity representation provides a cleaner description for XML schemas hiding implementation details and focusing on semantically relevant concepts. The X-Entity model extends the ER model so that one can explicitly represent important features of XML schemas, including: element and subelement relationships, occurrence constraints of elements and attributes and choice groups. Due to space limitations, some X-Entity features were not presented in this paper. Other issues were not considered in our approach, including: hierarchy of elements and attributes, cardinality of group of elements, elements with mixed content and order of elements imposed by a sequence compositor. However, our model can be easily extended with additional features and new rules can be developed for the conversion process. We already implemented a prototype to generate XEntity schemas from XML Schemas..."
- [November 24, 2003] "An Introduction to

Schematron." By Eddie Robertsson. From *XML.com* (November 12, 2003). "The Schematron schema language differs from most other XML schema languages in that it is a rule-based language that uses path expressions instead of grammars. This means that instead of creating a grammar for an XML document, a Schematron schema makes assertions applied to a specific context within the document. If the assertion fails, a diagnostic message that is supplied by the author of the schema can be displayed. One advantages of a rule-based approach is that in many cases modifying the wanted constraint written in plain English can easily create the Schematron rules. In order to implement the path expressions used in the rules in Schematron, XPath is used with various extensions provided by XSLT. Since the path expressions are built on top of XPath and XSLT, it is also trivial to implement Schematron using XSLT, which is shown later in the section Schematron processing. Schematron makes various assertions based on a specific context in a document. Both the assertions and the context make up two of the four layers in Schematron's fixed four-layer hierarchy: phases (top-level), patterns, rules (defines the context), and assertions... This introduction covers only three of these layers (patterns, rules and assertions); these are most important for using embedded Schematron rules in RELAX NG... Version 1.5 of Schematron was released in early 2001 and the next version is currently being developed as an ISO standard. The new version, ISO Schematron, will also be used as one of the validation engines in the DSDL (Document Schema Definition Languages) initiative..." See: "Schematron: XML Structure Validation Language Using Patterns in Trees."

- [August 05, 2003] "WDSL Tales From the Trenches, Part 3." By Johan Peeters. From *O'Reilly WebServices.xml.com* (August 05, 2003). ['Continuing the focus on sound design, we have the third and final installment of Johan Peeters' "WSDL Tales from the Trenches." Peeters concentrates on the importance of modeling the data elements involved in web services, and explains the best strategies for using W3C XML Schema to model this data.'] "I examine the type definitions and element declarations in the types element of a WSDL document. Such types and elements are for use in the abstract messages, the message elements in a WSD. WSDL does not constrain data definitions to W3C XML Schema (WXS). However, alternatives to WXS are not covered in this article: the goal of the series is to provide help and guidance with current real-world problems, and I have not seen any of the alternatives to WXS being used for web services

on a significant scale to date. This may change in the future: while only the WXS implementation is discussed in the WSDL 1.1 spec, it was always the intention of the WSDL designers to provide several options. The WSDL 1.2 draft's appendix on Relax NG brings this closer to realization. Data modeling with WXS is not for the faint-hearted. It presents a lot of pitfalls. This article will point some of these out and helps you avoid them..." On WSDL 1.2, see the announcement "W3C Releases Three Web Services Description Language (WSDL) 1.2 Working Drafts." The non-normative Appendex E ('Examples of Specifications of Extension Elements for Alternative Schema Language Support') in the WSDL Part 1: Core WD includes a section on RELAX NG: "A RELAX NG schema may be used as the schema language for WSDL. It may be embedded or imported; import is preferred. A namespace must be specified; if an imported schema specifies one, then the [actual value] of the `namespace` attribute information item in the `import` element information item must match the specified namespace. RELAX NG provides both type and element definitions which appear in the {type definitions} and {element declarations} properties of [Section 2.1.1] 'Definitions Component' respectively..."

- [July 16, 2003] "Logic Grammars and XML Schema." By C. M. Sperberg-McQueen (World Wide Web Consortium / MIT Laboratory for Computer Science, Cambridge MA). Draft version of paper prepared for Extreme Markup Languages 2003, Montréal. "This document describes some possible applications of logic grammars to schema processing as described in the XML Schema specification. The term logic grammar is used to denote grammars written in logic-programming systems; the best known logic grammars are probably definite-clause grammars (DCGs), which are a built-in part of most Prolog systems. This paper works with definite-clause translation grammars (DCTGs), which employ a similar formalism but which more closely resemble attribute grammars as described by [D. Knuth, 'Semantics of Context-Free Languages,' 1968] and later writers; it is a bit easier to handle complex specifications with DCTGs than with DCGs. Both DCGs and DCTGs can be regarded as syntactic sugar for straight Prolog; before execution, both notations are translated into Prolog clauses in the usual notation... Any schema defines a set of trees, and can thus be modeled more or less plausibly by a grammar. Schemas defined using XML Schema 1.0 impose some constraints which are not conveniently represented by pure context-free grammars, and the process of schema-validity-assessment

defined by the XML Schema 1.0 specification requires implementations to produce information that goes well beyond a yes/no answer to the question 'is this tree a member of the set?' For both of these reasons, it is convenient to use a form of attribute grammar to model a schema; logic grammars are a convenient choice. In [this] paper, I introduce some basic ideas for using logic grammars as a way of animating the XML Schema specification / modeling XML Schema... The paper attempts to make plausible the claim that a similar approach can be used with the XML Schema specification, in order to provide a runnable XML Schema processor with a very close tie to the wording of the XML Schema specification. Separate papers will report on an attempt to make good on the claim by building an XML Schema processor using this approach; this paper will focus on the rationale and basic ideas, omitting many details..." See also the abstract for the Extreme Markup paper [Tuesday, August 5, 2003]: "The XML Schema specification is dense and sometimes hard to follow; some have suggested it would be better to write specifications in formal, executable languages, so that questions could be answered just by running the spec. But programs are themselves often even harder to understand. Representing schemas as logic grammars offers a better approach: logic grammars can mirror the wording of the XML Schema specification, and at the same time provide a runnable implementation of it. Logic grammars are formal grammars written in logic-programming systems; in the implementation described here, logic grammars capture both the general rules of XML Schema and the specific rules of a particular schema." *Note*: the paper is described as an abbreviated version of "Notes on Logic Grammars and XML Schema: A Working Paper Prepared for the W3C XML Schema Working Group"; this latter document (work in progress 2003-07) provides "an introduction to definite-clause grammars and definite-clause translation grammars and to their use as a representation for schemas."

- [June 21, 2003] "Mobile Subset of XML Schema Part 2." ISO Document for information and review. Produced by SC34 Japan for ISO/IEC JTC 1/ SC34/WG1: Information Technology -- Document Description and Processing Languages -- Information Presentation. Project 19757-5 (Project Editor, Martin Bryan). ISO Reference: ISO/IEC JTC 1/SC34 N 0410. April 22, 2003. Excerpts: "We propose to create a compact and reliable subset of W3C XML Schema Part 2 and publish it as an ISO standard. The main target of this subset is mobile devices (such as cellular phones). Mobile devices are expected to use XML

in the near future. Small XML parsers have been developed already. Validators for schema languages are expected to follow, and a prototypical validator for RELAX NG on mobile phones has been developed. Such parsers and validators will hopefully be used for implementing XForms and Web Service on mobile devices. Part 2 of W3C XML Schema provides a set of datatypes and facets. Although it might not be perfect, it is likely to be widely used by many XML applications including mobile applications. We just cannot believe that an incompatible set of general-purpose datatype (*e.g.*, `int`) libraries will be accepted by the market. However, datatypes and facets of W3C XML Schema Part 2 are too complicated for mobile devices. Some specifications such as XForms have already created their own subsets of W3C XML Schema Part 2. However, if different specifications introduce different subsets, incomparability will be significantly spoiled. It would be much nicer if one subset is internationally standardized... [In the] choice of datatypes we omit: (1) datatypes requiring infinite precision; (2) datatypes that do not have obvious mapping to J2ME; (3) archaic datatypes such as IDREFS, ENTITY, ENTITIES, and NOTATION; (4) unsolid datatypes -- `dateTime` and so forth; (5) datatypes such that validity depends on namespace declarations... [In the] choice of facets we omit: [1] the pattern facet, which requires the property list of Unicode characters; [2] whitespace, which does not affect validity but controls PSVI; [3] `totalDigits` and `fractionDigits`. Implementation considerations: We have studied the source code of Jing implementation by James Clark. We believe that if the above restrictions are accepted, an implementation of the remaining datatypes and facets will require less than 20KB as the size of a JAR file." See details for the proposed list of datatypes and factes in 'Table 1: The list of datatypes' and 'Table 2: The list of facets'. [cache]

- [June 19, 2003] Namespace Routing Language (NRL) Supports Multiple Independent Namespaces. James Clark has announced the publication of a Namespace Routing Language (NRL) specification. NRL is "an XML language for combining schemas for multiple namespaces; it allow the schemas that it combines to use arbitrary schema languages." The release includes a tutorial and specification document and a sample implementation in the Jing (RELAX NG Validator in Java) distribution. NRL "is the successor to Clark's Modular Namespaces (MNS) language and is intended to be another step on the path towards Document Schema Definition Languages (DSDL) Part 4." The W3C XML Namespaces Recommendation itself "allows an

XML document to be composed of elements and attributes from multiple independent namespaces: each of these namespaces may have its own schema and the schemas for different namespaces may be in different schema languages. The problem then arises of how the schemas can be composed in order to allow validation of the complete document." The Namespace Routing Language attempts to solve this problem. Among the features and benefits of NRL: it supports schema language coexistence, allows extension of schemas not designed to be extended, makes authoring of extensible schemas easier supports 'transparent' namespaces, allows contextual control of extension, and allows concurrent validation. "For RELAX NG, it can be used to provide some of the namespace-based modularity features that are built-in to XSD. NRL is designed to allow an implementation to stream, and the sample implementation does so. The sample implementation has a SAX-based plug-in architecture that allows new schema languages to be added dynamically. It comes with support for RELAX NG (both XML and compact syntax), W3C XML Schema (via a wrapper around Xerces-J), Schematron, and (recursively) NRL; it can also use any schema language with an implementation that supports the JARV interface."

- [June 10, 2003] "Introducing Examplotron: The Fastest Road to Schema." By Uche Ogbuji (Principal Consultant, Fourthought, Inc). From IBM developerWorks, XML zone. June 10, 2003. ['A zoo of XML schema languages is out there, and although some of the beasts are bigger than others none is as friendly as Examplotron. With Examplotron, your example XML document is your schema, for the most part. It requires you to learn very little new syntax, and most of the core features of XML can be specified by providing representative examples in the source. In this article, Uche Ogbuji introduces Examplotron, providing plenty of examples.'] "At first XML had the Document Type Definition (DTD). XML 1.0 came bundled with the schema technology inherited from SGML. However, numerous XML users complained about DTDs including the fact that they use a different syntax from XML itself. The W3C developed a successor technology to DTD, W3C XML Schema, but some complained that it was too complex, and that it showed every sign of design-by-committee. Separate groups developed schema technologies that became RELAX NG and Schematron. These technologies all have their strengths and weaknesses, and their attendant factions. But for the developer with deadlines to mind, crafting schemata is often too much of an additional burden. Without a doubt, it is always a good idea to develop a schema. If for no other reason, it provides documentation of the

format. But in the real world, the most common course for harassed developers is to develop a sample of the XML format to serve all purposes of a proper schema. But what if the example itself could provide the benefits of a formal schema? In particular, what if the example could be used to validate documents? Eric van der Vlist set out to develop a system that allows example documents to serve as formal schemata, and his invention is Examplotron. In this article, I introduce Examplotron. This system is simple to use, so I encourage you to follow along by downloading Examplotron 0.7 (compile.xsl) and use your favorite XSLT and RELAX NG processors... On a recent project, a client who had many XML formats hired me, through my company Fourthought, to develop schemata for documentation and validation for these XML formats. All they had to start with were sample XML documents for each format. Using Examplotron to generate the production RELAX NG schemata from these sample documents saved me perhaps over a hundred hours of effort, and thus saved them tens of thousands of dollars. I did have to augment Examplotron with document generation and other refinement code; I hope to cover the non-proprietary aspects of this refinement code in a future article. Examplotron produces RELAX NG schemata, but if you must produce W3C XML Schema, all is still well: You can use James Clark's excellent Trang tool to convert RELAX NG to WXS. I know from my overall consulting experience that sample documents are the most common form of schema in the real world, so I expect that Examplotron will be of great help to a lot of folks right away." See Examplotron above.

- [April 30, 2003] "Clean Up Your Schema for SOAP. Updating XML Schemas to be SOAP-Friendly." By Shane Curcuru (Advisory Software Engineer, IBM Research). From IBM developerWorks, Web services. April 29, 2003. ['More and more projects are using XML schemas to define the structure of their data. As your repository of schemas grows, you need tools to manipulate and manage your schemas. The Eclipse XSD Schema Infoset Model has powerful querying and editing capabilities. In this article, Shane Curcuru will show how you can update a schema for use with SOAP by automatically converting attribute uses into element declarations.'] "If you've built a library of schemas, you might want to reuse them for new applications. If you already have a data model for an internal purchase order, as you move towards Web services you may need to update it for use with SOAP. SOAP allows you to transport an xml message across a network; the xml body can also

be constrained with a schema. However a SOAP message typically uses element data for its xml body, not attribute data. You'll explore a program that can automatically update an existing schema document to convert any attribute declaration into roughly 'equivalent' element declarations... Given the complexity of XML schemas, you certainly don't want to use Notepad to edit the .xsd files. A good XML editor is not much of a step up -- while it may organize your elements and attributes nicely, it can't show the many abstract Infoset relationships that are defined in the Schema specification. That's where the Schema Infoset Model comes in; it expresses both the concrete DOM representation of a set of schema documents, and the full abstract Infoset model of a schema. Both of these representations are shown through the programmatic API of the Model as well as in the Model's built-in sample editor for schemas... If you've installed the XSD Schema Infoset Model and Eclipse Modeling Framework (EMF) plugins into Eclipse, you can see the sample editor at work in your Workbench... performing a conceptually simple editing operation on schema documents (turning attributes into elements) can entail a fair amount of work. However the power of the Schema Infoset Model's representation of both the abstract Infoset of a schema and its concrete representation of the schema documents makes this a manageable task. The Model also includes simple tools for loading and saving schema documents to a variety of sources, making it a complete solution for managing your schema repository programmatically. Some users might ask, 'Why not use XSLT or another XML-aware application to edit schema documents?' While XSLT can easily process the concrete model of a set of schema documents, it can't easily see any of the abstract relationships within the overall schema that they represent. For example, suppose that you need to update any enumerated simpleTypes to include a new UNK enumeration value meaning unknown. Of course, you only want to update enumerations that fit this format of using strings of length of three; you would not want to update numeric or other enumerations... This article presupposes an understanding of schemas in XML and how SOAP works. The sample code included in the zip file works standalone or in an Eclipse workbench..."

- [March 24, 2003] Schema Unit Test (SUT) Framework for Testing XML Schema. Gavin Kingsley (Invensys Energy Systems Limited) announced the availability of a SourceForge project Schema Unit Test (SUT) which introduces a framework for testing XML Schema. SUT incorporates the Schematron reference

implementation developed by Rick Jelliffe of the Academia Sinica Computing Centre. Problem statement: "W3C Schema can quickly become complex and difficult to determine if they are validating the correct vocabulary. The addition of embedded Schematron schema only makes this problem worse... The SUT framework has two parts. The first is a namespace and vocabulary for embedding test cases into sample XML documents, designed to highlight what is legal and what is not legal in the vocabulary defined in the schema under test. This aspect is independent of what schema language is used and can in theory be applied to any schema language with automatic validation tools. The second part is a Java implementation using JUnit for testing a W3C Schema with embedded Schematron schema. This implementation reads SUT test suite descriptions written in XML with embedded test cases and then creates a JUnit test suite that can be executed inside JUnit in the usual way. Although SUT is written to use JUnit, no specialise Java or JUnit knowledge is required to run SUT test suites. An example is provided based on the purchase order schema from the W3C primer... A SUT Test Suite is a well-formed XML file containing an example of a file to be validated. Test cases are identified by additional elements in the SUT namespace, `http://www.powerware.com/nz/XMLSchemaUnitTest`. The case element identifies test cases created by adding or removing elements. The attribute element identifies test cases created by adding, removing or changing attributes; detailed descriptions of these elements are available (case; attribute)." SUT has free, open source code.

- [February 21, 2003] "Mapping Between UML and XSD." By David Carlson (Ontogenics Corp). From *XMLmodeling News* Volume One, Issue Two (January 28, 2003). "One of the principal advantages of using UML when designing XML vocabularies is that the model can serve as a specification which is independent of a particular schema language implementation. W3C XML Schema is the most common choice right now, but we hope that business vocabularies (and other non-business technical markup languages) have a long life and will be implemented using alternative new schema languages. To achieve this goal, we need to define a complete and flexible mapping between UML and each implementation language. Given that UML was originally intended for object-oriented analysis and design, the mapping is most straightforward for languages that have an object-oriented flavor... A bi-directional mapping between UML and schemas is specified in the form of a UML Profile. The purpose of a UML profile for this or any other

use is to extend the UML modeling language with constructs unique to an implementation language, analysis method, or application domain. The profile extension mechanism is part of the UML standard; it was expanded in the recent UML version 1.4 and will be further expanded when UML 2.0 is adopted this year. A UML profile (pre version 2.0) is composed of three constructs: stereotypes, tagged value properties, and constraints. A stereotype defines a specialized kind of UML element; for example, the `XSDcomplexType` stereotype defines a specialized kind of UML Class, and `XSDschema` defines a specialized kind of UML package. Tagged values define properties of these stereotyped elements. So the `XSDschema` stereotype includes a targetNamespace property. By assigning this stereotype to a UML package and setting a value for this property, we have augmented the UML modeling language with information used to generate a complete XML Schema document from an abstract vocabulary model. Similar stereotypes and properties are defined for all XML Schema constructs. A profile constraint specifies rules about how and where stereotypes and their tagged values can be used in a model. These rules should include what are often called co-constraints: how the value of one property constrains the values of other properties..."

- [January 23, 2003] Trang Multi-Format Schema Converter Supports DTD to W3C XML Schema Conversion. A posting from James Clark to the XML-DEV List announces a new release of Trang, Clark's Multi-Format Schema Converter based on RELAX NG. The conversion tool supports several schema languages for XML, including RELAX NG (XML syntax), RELAX NG compact syntax, XML 1.0 DTDs, W3C XML Schema. With one exception, Trang will convert between any of these formats (W3C XML Schema is supported for output only, not for input). "Trang is written in Java, and available under a BSD-style license. In this release, [Clark has] added an input module for DTDs based on his DTDinst program; this implies that Trang can now convert directly from DTDs to W3C XML Schema (XSD)." Clark identifies three unique features of Trang: "(1) it can reliably turn parameter entities into the higher-level semantic constructs available in XSD (simple types, groups, attribute groups) -- even in the presence of arbitrarily deep nesting of parameter entity references within parameter entity declarations; (2) it supports namespaces, including DTDs that mix multiple namespaces; (3) it can create good-quality, idiomatic XSD, which takes advantage of features such as substitution groups."

- [January 22, 2003] "Altova Simplifies XML Development Through Enhanced Support for Microsoft .NET, Oracle XML DB, Web Services and Document Publishing in XMLSPY 5 Release. New features further demonstrate XMLSPY 5 is the most comprehensive XML development environment for any XML-enabled software project." - "Altova Inc., producer of XMLSPY, the world's leading XML development tool with over a million registered users, today announced the availability of XMLSPY 5 Release 3. Presently, XML technologies today are being applied to solve a wide spectrum of enterprise computing challenges, including electronic commerce, document publishing, database integration, and web services applications. To maximize software developer productivity in implementing any XML-enabled solution, Altova has added enhanced developer support for various widely used enterprise technologies, thereby accelerating and simplifying XML development. The new version of XMLSPY is available immediately for free trial download. To meet the needs of enterprise developers, XMLSPY 5 builds on the success of previous award winning versions through the addition of several key new features: (1) Improved Support for building Microsoft .NET applications: The XMLSPY 5 Code Generator now supports Microsoft C# code generation to accelerate application development on the Microsoft .NET platform. Use XMLSPY to Model data elements in XML Schema, then XMLSPY can auto-generate C# class (data bindings) corresponding to elements defined in the data model. The generated code uses System.XML, the new Microsoft .NET Application Program Interface (API) for programmatically accessing XML documents. (2) Enhanced support for Oracle XML DB. XMLSPY's new features enable developers to easily perform common operations on data managed by XML DB including: List XML Schemas, Load a Schema from a list, Save New or Delete XML Schema to Oracle XML DB, Register an XML Schema with Oracle XML DB, Execute Query using Oracle9i's DBURI, Browse, Open, Edit and Save XML documents stored in Oracle XML DB via WebDAV. (3) Additional Web Services Support: A new Web Service Description Language (WSDL) Documentation generation utility makes it easier for a Web service developer to document and publish a Web service's interface to business partners, other developers, or to the public. (4) PDF support for Document Publishing: XMLSPY's stylesheet designer now supports visual editing and generation of eXtensible Stylesheet Language Formatting Object (XSL:FO) code, which enables XML content to be rendered into a PDF file. Now, with a single stylesheet

design, developers can preview the output of a stylesheet transformation in either PDF or HTML. (5) Usability Enhancements for Stylesheet Designer: Improved drag/drop functionality in Stylesheet Designer. (6) New Java Integration Support: A new Java API enables easier customization and integration of the XMLSPY development environment for system integrators or Java-based product companies wanting to extend the functionality of XMLSPY. Now, programmers can control and use XMLSPY functionality from their Java-based programs. XMLSPY already supports integration via a COM based interface..."

- [January 21, 2003] "Requirements for XML Schema 1.1." Edited by Charles Campbell, Ashok Malhotra (Microsoft), and Priscilla Walmsley. W3C Working Draft 21-January-2003. Version URL: `http://www.w3.org/TR/2003/WD-xmlschema-11-req-20030121/`. Latest version URL: http://www.w3.org/TR/xmlschema-11-req/. "This document contains a list of requirements and desiderata for version 1.1 of XML Schema... Since the XML Schema Recommendation (Part 0: Primer, Part 1: Structures, and Part 2: Datatypes) was first published in May, 2001, it has gained acceptance as the primary technology for specifying and constraining the structure of XML documents. Users have employed XML Schema for a wide variety of purposes in many, many different situations. In doing so, they have uncovered some errors and requested some clarifications. They have also requested additional functionality. Most of the errors and clarifications are addressed in the published errata and will be integrated into XML Schema 1.0 Second Edition, to be published shortly. Additional functionality and any remaining errors and clarifications will be addressed in XML Schema 1.1 and XML Schema 2.0. This document discusses the requirements for version 1.1 of XML Schema. These issues have been collected from e-mail lists and minutes of telcons and meetings, as well as from the various issues lists that the XML Schema Working Group has created during its lifetime. Links are provided for further information. The items in this document are divided into three categories: (1) A requirement must be met in XML Schema 1.1; (2) A desideratum should be met in XML Schema 1.1; (3) An opportunistic desideratum may be met in XML Schema 1.1..."
- [January 17, 2003] "From Model to Markup: XML Representation of Product Data." By Joshua Lubell (US National Institute of Standards and Technology, Manufacturing Systems Integration Division). Paper presented December 2002 at the XML 2002 Conference, Baltimore, MD, USA. With

Appendix (Complete PDM Example with EXPRESS Schema, RELAX NG Schema [Compact Syntax], RELAX NG Schema [XML Syntax], W3C XML Schema, Flat XML Data, Hierarchical XML Data). "Business-to-consumer and business-to-business applications based on the Extensible Markup Language (XML) tend to lack a rigorous description of product data, the information generated about a product during its design, manufacture, use, maintenance, and disposal. ISO 10303 defines a robust and time-tested methodology for describing product data throughout the lifecycle of the product. I discuss some of the issues that arise in designing an XML exchange mechanism for product data and compare different XML implementation methods currently being developed for STEP... ISO 10303, also informally known as the Standard for the Exchange of Product model data (STEP), is a family of standards defining a robust and time-tested methodology for describing product data throughout the lifecycle of the product. STEP is widely used in Computer Aided Design (CAD) and Product Data Management (PDM) systems. Major aerospace and automotive companies have proven the value of STEP through production implementations. But STEP is not an XML vocabulary. Product data models in STEP are specified in EXPRESS (ISO 10303-11), a modeling language combining ideas from the entity-attribute-relationship family of modeling languages with object modeling concepts. To take advantage of XML's popularity and flexibility, and to accelerate STEP's adoption and deployment, the ISO group responsible for STEP is developing methods for mapping EXPRESS schemas and data to XML. The rest of this paper discusses some of the issues that arise in designing an XML exchange mechanism for product data. It also compares two implementation approaches: a direct mapping from EXPRESS to XML syntax rules, and an indirect mapping by way of the Unified Modeling Language (UML)... *Observations*: Developers of mappings from product data models to XML have learned some valuable lessons. The most important lesson is that, although XML makes for an excellent data exchange syntax, XML is not well suited for information modeling. After all, if XML were a good modeling language, it would be much easier to develop a robust mapping from EXPRESS to XML. UML, on the other hand, is a good modeling language and, therefore, it should be easier to map EXPRESS to UML than to XML. Although mapping EXPRESS to UML does not make the difficulties of representing product models in XML go away, it offloads a significant part of the effort from the small and resource-strapped STEP developer community to the much larger and

resource-rich UML developer community. Another lesson learned is that the choice of XML schema language used in the mapping is not very important, as long as the schema language supports context-sensitive element types and provides a library of simple data types for items such as real numbers, integers, Booleans, enumeration types, and so on. Therefore, both the W3C XML Schema definition language and RELAX NG are good XML schema language choices. The DTD is a poor schema language choice for the reasons discussed earlier..." See Lubell's RELAX NG List Note of 2003-01-17: "Although David Carlson's "Modeling XML Applications with UML" (Addison Wesley) mentions WXS but not RELAX NG, his hyperModel tool can generate both WXS and RELAX NG schemas from XMI. Carlson used to have a web-accessible form where you could upload an XMI file and process it using hyperModel, but the website wasn't working the last time I tried to use it. He has also implemented hyperModel as a plug-in to Eclipse and is selling this as a commercial product. I haven't come across any documentation for Carlson's UML-to-RELAX NG mapping. The RELAX NG in my XML 2002 paper "From Model to Markup" was created using the web-accessible hyperModel tool, although I had to tweak the output in order to handle bidirectional UML associations and to use interleave properly..." See: "Conceptual Modeling and Markup Languages."

- [January 17, 2003] "Transforming XML Schemas." By Eric Gropp. From XML.com. January 15, 2003. ['Eric Gropp shows how XSLT can be used to transform W3C XML Schemas to create, among other things, HTML input forms, generate query interfaces, and documentation of data structures and interfaces.'] A W3C XML Schema (WXS) document contains valuable information that can be used throughout a system or application, but the complexity that WXS allows can make this difficult in practice. XSLT, however, can concisely and efficiently manipulate WXS documents in order to perform a number of tasks, including creating HTML input forms, generating query interfaces, documenting data structures and interfaces, and controlling a variety of user interface elements. As an example, this article describes an XSLT document which creates an XHTML form based on the WXS definition of a complex element. For brevity and clarity, the article omits several WXS and XHTML form aspects, including attribute definitions, keys, imported/included schemas, and qualified name issues. How these additional features are implemented can depend greatly on your use of WXS and on your application. However, building a

stylesheet that handles every possible WXS feature can be quite an effort and may often be unnecessary. Much of the information -- occurrence constraints, data types, special restrictions, and enumerations -- needed to build an XHTML form is already contained in a WXS document. Missing bits such as label text and write restrictions can be added into WXS's `<annotation>` element. The stylesheet will perform four distinct tasks: (1) Find the definition of the target complex element that we want; (2) Build a form element for the target element; (3) Find the definitions of the target element's valid children; (4) Build an input element for each of the simple child elements. In order to do this, the stylesheet will apply different template rules to similar WXS elements depending on the task at hand. To make this possible, the stylesheet will use a separate mode for each task... This article will preface the description of a set of template rules with a model that is based on UML Activity Diagrams. In these diagrams template rules are shown as states; modes are shown as composite states... Using WXS as the common resource for data typing in your application can have big payoffs. By allowing components and interfaces to automatically reflect changes to an application's data model, you can greatly increase the reusability and flexibility of a system. XSLT is a useful, largely platform-independent, and highly portable tool for making this possible." See also *XML Schema: The W3C's Object-Oriented Descriptions for XML*, by Eric van der Vlist.

- [January 10, 2003] W3C Working Draft Specification for XML Schema Component Designators. The W3C XML Schema Working Group has released an initial working draft for *XML Schema: Component Designators*. The specification defines a system for designating XML Schema components. The document addresses a range of problematic issues in the use of a `QName` to designate schema components as defined in the W3C XML Schema Recommendation. "The schema-as-a-whole schema component may represent the amalgamation of several distinct schema documents, or none at all. It may be associated with any number of target namespaces, including none at all. It may have been obtained for a particular schema assessment episode by deferencing URIs given in schemaLocation attributes, or by an association with the target namespace or by some other application-specific means. In short, there are substantial technical challenges to defining a reliable designator for the schema-as-a-whole, particularly if that designator is expected to serve as a starting point for the other components encompassed by that schema.

The editors propose to divide the problem of constructing schema component designators into two parts: defining a designator for an assembled schema, and defining a designator for a schema component, understood relative to a designated schema."

- [December 26, 2002] "A Data Model for Strongly Typed XML." By Dare Obasanjo. From XML.com. December 18, 2002. ['Dare Obasanjo has been searching for an appropriate data model for typed XML. In his article he examines the Post Schema-Validation Infoset, but settles on the XPath/ XQuery data model as the best candidate for applications that want to use a strongly typed data model for XML.'] "In many XML applications, the producers and consumers of XML documents are aware of the datatypes within those documents. Such applications can benefit from manipulating XML via a data model that presents a strongly typed view of the document. Although a number of abstractions exist for manipulating XML -- the XML DOM, XML infoset, and XPath data model -- none of these views of XML take into account usage scenarios involving strongly typed XML. Many developers utilize XML in situations where type information is known at design or compile time, including interacting with relational databases and strongly typed programming languages like Java and C#. Thus, there is a significant proportion of the XML developer community which would benefit from a data model that encouraged looking at XML as typed data. This article is about my search for and discovery of this data model. The W3C XML Information Set recommendation describes an abstract representation of an XML document. The XML Infoset is primarily meant to act as a set of definitions used by XML technologies to formally describe what parts of an XML document they operate on. Several W3C XML technologies are described in terms of the XML Infoset, including SOAP 1.2, XML Schema, and XQuery. An XML document's information set consists of a number of information items. An information item is an abstract representation of a component of an XML document: such as an element, attribute or processing instruction. Each information item has a set of associated named properties. Each property is either a collection of related information items or data about the information item; the [children] property of an element information item is an example of the former, while the [base URI] of a document information item is an example of the latter. An XML document's information set must contain a document information item from which all other information items belonging to the document can be accessed. The XML Infoset is a tree-based hierarchical representation of an XML document...

The XQuery and XPath 2.0 Data Model is still a working draft; some of its details may change before it becomes a W3C recommendation. However, the core ideas behind the data model which this article explores are unlikely to change. This article is based on the November 15th draft of the working draft. The XQuery and XPath 2.0 data model presents itself as a viable data model for processing XML in strongly typed usage scenarios. The loose coupling to the W3C XML Schema type system is especially beneficial because it both provides an interoperable set of types, yet does not limit one to solely those types. The XQuery and XPath 2.0 data model stands out as the most credible data model for dealing with XML in strongly typed scenarios. Given that the XQuery and XPath 2.0 data is based on the XML Infoset, and also builds upon the past experience with XPath 1.0, it's the best candidate for the Data Model for XML..."

- [December 12, 2002] "Datatypes for XML Topic Maps (XTM): Using XML Schema Datatypes." By Murray Altheim (Knowledge Media Institute, The Open University, Milton Keynes, Bucks, UK). Draft version 1.4 2002/12/12. Topic map available in XML format, also zipped. Reference posted to the OASIS Topic Maps Published Subjects Technical Committee list. Abstract: "The W3C Recommendation XML Schema Part 2: Datatypes ('XSD') provides a specification of datatypes and their facets, and forms the semantic basis of this document, which establishes Published Subject Indicators (PSIs) for each XML Schema datatype and facet. A PSI is a (relatively) stable URL used as a canonical identifier for a subject, particularly within an XML Topic Map (XTM) document, though application of PSIs is not limited to XTM. This document does not alter any XML Schema datatype definition; for definitions of each datatype..." Author's note: "I'm pleased to announce a first draft of something that's been in the works for over a year... The ability to constrain or 'type' topic characteristics is something necessary within the topic map community. Constraints on topic map structures may be provided by various forms of schema facilities, such as TMCL, but a simple datatyping feature is still something sorely missing. There are some examples provided in the document... I welcome comments or suggestions towards establishing 'best practices' for use of these PSIs, as well as comments on the structures within the provided topic map. I am willing to add visualizations of various parts of the topic map if that is considered helpful." The purpose of the Topic Maps Published Subjects Technical Committee is "to promote the use of Published Subjects by specifying requirements, recommendations, and

best practices, for their definition, management and use. Public Subject was defined by ISO 13250 Topic Maps standard and further refined as Published Subject in the XML Topic Maps (XTM) 1.0 Specification." See: "(XML) Topic Maps."

- [November 30, 2002] "Modeling Biz Docs in XML." By Jon Udell. In *InfoWorld* (November 29, 2002). ['Unlocking Office 11's XML features means coming to grips with its data definition language, XML Schema. That won't be easy, but the sooner we start, the better. The future of Web services depends on our ability to model business documents in XML. Yes, XML Schema is complex, but some of the issues are more general. Even experts disagree on the best practices for object-oriented data modeling. Office 11 creates an environment in which we can start to codify those best practices as they apply to ordinary business documents.'] "The good news is that Office 11 supports XML Schema. The bad news is that XML Schema has been described even by XML experts as 'confusing,' 'impenetrable,' 'fuzzy,' and 'as user-friendly as a stick in the eye.' A successor to the SGML/XML DTD (Standard Generalized Markup Language/ XML document type definition), XML Schema is a language for writing rules that constrain the kinds of elements that can appear in documents and the ways in which they can be sequenced, grouped, and nested. XML Schema is still a relatively new specification. The W3C Recommendation for XML Schema was published in May 2001. XML parsers that support XML Schema haven't done so for very long, and there is not yet much experience using it. Most people who are adept at defining document structure learned how to do so by writing DTDs. Some of the allergic reaction to XML Schema can, therefore, be chalked up to normal reluctance to learn new skills... Upgrading the word processors and spreadsheets on those government computers to versions that not only can read and write XML, but, more crucially, can enforce rules about datatypes and structures, is part of the solution. Assuming, of course, that such rules can be written, deployed, and unobtrusively applied and maintained over time. 'Therein,' observes Windley, 'lies the rub.' There is very little extant knowledge about how to model unstructured and semistructured data in XML. Unlike SGML, the XML DTD was always optional, because the framers of XML knew there was enormous value in documents that were merely well-formed, even if not valid with respect to a DTD. RSS (Rich Site Summary), for example, the wildly popular XML format for content syndication, has no DTD or schema... One possibility is to infer schemas from example documents. Tools can do this, but so far, not with much sophistication. Microsoft, for example, offers a .Net namespace

(`Microsoft.XsdInference`) that will infer a schema from an XML document, and even refine that schema based on further examples. The results make a useful starting point, and inferencing is a promising technology that can and should evolve, but the fact is that modeling XML data is a complex subject that even the best human experts have yet to codify. XML Schema delivers a much richer set of modeling tools than were available to DTD authors. Learning to use them well is going to be a challenge..." See also "Microsoft Office 11 and XDocs."

- [November 26, 2002] "XML Spy Tops as XML Editor." By Timothy Dyck. In *eWEEK* (November 25, 2002). "Altova GMBH's XML Spy has long been a strong player in the XML space, and Version 5 of the XML editor raises the bar even higher. Of all the XML editors eWEEK Labs has seen -- and we've seen a lot -- the $990 XML Spy 5.0 Enterprise Edition provides the best overall combination of editing power and usability, along with wide database and programming language integration support. This earns the product an eWEEK Labs Analyst's Choice award. XML Spy's user interface -- particularly its graphical schema editing tools and grid-based XML data editor -- keeps impressing us with its versatility, intuitiveness and power. For quick, ad hoc XML transformations, such as converting a series of attributes into elements, XML Spy is a perfect tool. The Enterprise Edition of XML Spy is new to the product line. It includes cutting-edge HTML-to-XML conversion capabilities; Java and C++ code generation; and Web services features, including a Simple Object Access Protocol debugger and a graphical WSDL (Web Services Description Language) file editor. The $399 Professional Edition of XML Spy does not have these features but does include the XML and XML Schema editing features, database import and export capabilities, and the XSLT (Extensible Stylesheet Language Transformations) debugger found in 5.0... The XSLT debugger in the new XML Spy line is important not only because it will be highly useful to developers, but also because it was one feature the competition had that XML Spy didn't. Excelon Corp.'s Stylus Studio 4.5, for example, has a very effective XSLT debugger... Also significant in XML Spy 5.0 is a new feature that helps automate the conversion of an HTML-based site to one that is based on XML technologies. XML Spy accomplishes this by transforming XML source data through Extensible Stylesheet Language into HTML... Plus: HTML-to-XML conversion capabilities; XSLT processor and debugger; graphical WSDL editor; Java and C++ code generators for XML data structures; Web services debugger; powerful XML editing features.

Minus: Lacks support for DB2 and Sybase Adaptive Server Enterprise XML database extensions."

- [November 22, 2002] "W3C XML Schema Design Patterns: Avoiding Complexity." By Dare Obasanjo. From XML.com. November 20, 2002. ['A year or so ago XML.com published an article called "W3C XML Schema Made Simple," which suggested, somewhat controversially, that you should avoid areas of the W3C XML Schema specification in order to keep things manageable. This week our main feature is both a companion piece and counterpoint to the "Made Simple" article. In "W3C XML Schema Design Patterns: Avoiding Complexity" Dare Obasanjo suggests that most of W3C XML Schema is indeed useful, but highlights areas the schema author should handle with care.'] "Over the course of the past year, during which I've worked closely with W3C XML Schema (WXS), I've observed many schema authors struggle with various aspects of the language. Given the size and relative complexity of the WXS [W3C XML Schema] recommendation, it seems that many schema authors would be best served by understanding and utilizing an effective subset instead of attempting to comprehend all of its esoterica. There have been a few public attempts to define an effective subset of W3C XML Schema for general usage, most notable have been W3C XML Schema Made Simple by Kohsuke Kawaguchi and the X12 Reference Model for XML Design by the Accredited Standards Committee (ASC) X12. However, both documents are extremely conservative and advise against useful features of WXS without adequately describing the cost of doing so. This article is primarily a counterpoint to Kohsuke's and considers each of his original guidelines; the goal is to provide a set of solid guidelines about what you should do and shouldn't do when working with WXS... The Guidelines: I've altered some of Kohsuke's original guidelines [...] I propose some additional guidelines as well: Do favor key/keyref/unique over ID/IDREF for identity constraints; Do not use default or fixed values especially for types of xs: QName; Do not use type or group redefinition; Do use restriction and extension of simple types; Do use extension of complex types; Do carefully use restriction of complex types; Do carefully use abstract types; Do use elementFormDefault set to qualified and attributeFormDefault set to unqualified; Do use wildcards to provide well defined points of extensibility... The WXS recommendation is a complex specification because it attempts to solve complex problems. One can reduce its burdens by utilizing its simpler aspects. Schema authors should ensure that their

schemas validate in multiple schema processors. Schemas are an important facilitator of interoperability. It's foolish to depend on the nuances of a specific implementation and inadvertently give up this interoperability..."

- [November 15, 2002] "Normalizing XML, Part 1." By Will Provost. From XML.com. November 13, 2002. ['Will Provost's Schema Clinic series on XML.com has so far taken an object-oriented view of W3C XML Schema design. This month, Will has written the first of a two-part series that examines the relational aspects of schema design. The series examines guidelines that achieve the goal of normalization -- the principles guiding database design -- using the mechanisms provided by W3C XML Schema.'] "The goal is to see what relational concepts we can usefully apply to XML. Can the normal forms that guide database design be applied meaningfully to XML document design? Note that we're not talking about mapping relational data to XML. Instead, we assume that XML is the native language for data expression, and attempt to apply the concepts of normalization to schema design. The discussion is organized loosely around the progression of normal forms, first to fifth. As we'll see, these forms won't apply precisely to XML, but we can adhere to the law's spirit, if not its letter. It's possible to develop guidelines for designing W3C XML Schema (WXS) that achieve the *goals* of normalization: (1) Eliminate ambiguity in data expression; (2) Minimize redundancy -- some would say, 'eliminate all redundancy'; (3) Facilitate preservation of data consistency; (4) Allow for rational maintenance of data. In this first of two parts, we'll consider the first through third normal forms, and observe that while there are important differences between the XML and relational models, much of the thinking that commonly goes into RDB design can be applied to WXS design as well. ... the key concept of reducing redundancy through key association is alive and well in W3C XML Schema design. While I'd love to finish on this bright note, I must report that there are devils inhabiting the details. In part two of this article, I'll point them out and discuss the implications for WXS design, as well as addressing the subtler fourth and fifth normal forms..."
- [November 14, 2002] "Web Services Development: Jean Paoli on XML in Office 11." By Jon Udell. In *InfoWorld* (November 14, 2002). ['Next week's issue of InfoWorld includes an article on the new XML capabilities of Office 11. While researching the story, I interviewed the architect of XML in Office 11, Microsoft's Jean Paoli, one of the primary co-creators of XML. Here are some of his remarks == excerpts from Paoli'] "... The goal is to unleash the Excel functionality

on generic schema, on customer-defined schema. Who knows how to create a data model better than the financial or health care company who uses the data? Until now, it was very difficult to find a tool which lets you pour the data belonging to any arbitrary schema, and then, for example, chart that data... All our tools are XML editors now: Word, Excel, XDocs. But we shouldn't think about XML editors, we should think about the task at hand. If I want to create documents with a lot of text, that's Word. With XDocs, the task is to gather information in structured form. And with Excel, it's to analyze information. We have this great toolbox which enables you to analyze data. We can do pie charts, pivot tables, I don't know how many years of development of functionality for analyzing data. So we said, now we are going to feed Excel all the XML files that you can find in nature... To create the schema for your spreadsheet, first look at the information which is captured in that spreadsheet. Give names to the data. The data is about the user's name and e-mail address, for example. I don't want to call it cell 1, cell 2, or F1 or F11. The whole thing about XML is to give names to things which are in general not named... The goal is to unleash the Excel functionality on generic schema, on customer-defined schema. Who knows how to create a data model better than the financial or health care company who uses the data? Until now, it was very difficult to find a tool which lets you pour the data belonging to any arbitrary schema, and then, for example, chart that data..." Udell says: "Modeling XML data using DTD (Document Type Description) or, more recently, XML Schema, has been a fairly arcane discipline. Practitioners have included publishers seeking to repurpose content and Web services developers writing WSDL files for which XML Schema serves as the type definition language. But enterprise data managers have not, in general, seen much reason to model lots of data using XML Schema. With Office 11, Microsoft aims to rewrite the rules in a dramatic way. If every enterprise desktop can consume, process, and emit schema-valid XML data, the modeling of that data becomes a huge strategic opportunity. And the people who can do that modeling effectively become very valuable..." See "Microsoft 'XDocs' Office Product Supports Custom-Defined XML Schemas."

- [November 12, 2002] "Place XML Message Design Ahead of Schema Planning to Improve Web Service Interoperability." By Yasser Shohoud. In *MSDN Magazine* Volume 17, Number 12 (December 2002). ['Web Services are all about exchanging data in the form of XML messages. If you were about to design a database schema, you probably wouldn't let your

tool do it for you. You'd hand-tool it yourself to ensure maximum efficiency. In this article, the author maintains that designing a Web Service should be no different. You should know what kind of data will be returned by Web Service requests and use the structure of that data to design the most efficient message format. Here you'll learn how to make that determination and how to build your Web Service around the message structure.'] "When you build a data-centric application, how do you create the database schema? Do you begin by creating classes and then let your IDE or tools create the database schema for you, or do you design the database schema yourself, taking into account normalization, referential integrity, and performance optimizations? Chances are you design and create the database schema yourself. Even if you use a visual schema designer rather than data definition language (DDL) statements, you are still taking control of the database schema design. Web Services are all about supplying the right data at the right time. When a client calls a Web Service, an XML data message is sent over the wire and a response is returned to the client. When you program the Web Service and its clients, you are really programming against these messages. The data in these messages is ultimately what the application cares about. So why would you create a Web Service beginning with the classes and methods and let the tools create the message schemas for you? You should design the data (message) schema and implement the Web Service to fit this design, like you would when designing a database schema... Web Services are all about applications exchanging data over the Web in the form of XML messages, so building a Web Service requires careful design of these messages using XML Schema and WSDL. When you begin with message design rather than method design the kind of data your Web Service expects to receive and return is made clear. By designing messages using XSD and WSDL, you create a formal interface definition that Web Service developers can implement and client developers can program against simultaneously. Next time you begin a Web Service project, begin by designing the messages format using the Visual Studio XML Schema designer..."

- [November 08, 2002] SchemaViewer 1.0. Francis Kilkelly has announced the availability of a SchemaViewer 1.0 tool which "virtually eliminates tedious browsing of XML Schema documents by representing them as a easily navigatable hierarchical tree. The application is a Swing-based GUI. Features: (1) The tool allows you to quickly and easily browse the contents of any W3C-compliant XML Schema. (2) It displays any

XML Schema as a hierarchical tree comprising of elements encountered within the schema/s. (3) If an element has 'type', 'ref' or 'base' attributes then the referenced element will appear as a child of the current element in the tree. (4) If the XML Schema has any import or include statements this tool will include the contents of the corresponding XML Schema. (5) This tool also allows you to view XML Schemas embedded within WSDL (Web Services Definition Language) documents." Requires Java Run-Time Environment 1.3.1 or higher. See related resources in W3C list of XML Schema tools.

- [November 04, 2002] Architag XML Editor XRay Supports W3C XML Schema. A posting from Mae Ozkan [2002-11-01] reports that Architag's XML Editor XRay now supports XML Schema (XSD) with XRay version 2.0. "Full W3C XML Schema (also called XSD) support is built into version 2 of XRay. XSD schemas are automatically parsed according to the W3C specification to assure compliance. Then, a schema is available, using XML Namespaces, to validate other XML documents within XRay. The XRay editing engine offers a real-time validator. Parsing errors are shown in real time as you type your XML tags and content. The real-time editing functionality helps new users learn XML quickly because they get instant feedback. Web Services Description Language (WSDL) is fully supported in XRay 2.0. This includes all parts of WSDL documents, including intelligent parsing of schemas within the WSDL file. XRay has built-in XSLT processing that, like the XML engine, provides real-time transformation of XML structures, including HTML. There is also a built-in HTML viewer for quick development of XML-based Web pages..." See the screen shots. XRay 2.0 is available for free download from http://architag.com/xray.
- [October 31, 2002] "Analyze Schemas with the XML Schema Infoset Model." By Shane Curcuru. From DevX XML Zone. October 2002. ['IBM's new XML Schema Infoset Model provides a complete modeling of schemas themselves, including the concrete representations as well as the abstract relationships within a schema or a set of schemas. Learn how to use this powerful library to perform complex queries on your own schemas.'] "As the use of schemas grows, so does the need for tools to manipulate those schemas. IBM's new XML Schema Infoset Model provides a complete modeling of schemas themselves, including the concrete representations as well as the abstract relationships within a schema or set of schemas. This library easily queries the model of a schema for detailed information. You can also use it to update the schema to fix any problems found and write the schema back out. Although there are a

number of parsers and tools that use schemas to validate or analyze XML documents, tools that allow querying and advanced manipulation of schema documents themselves are still being built. The XML Schema Infoset Model (AKA the Java packages `org.eclipse.xsd.*`, or just 'the library') provides a rich API library that models schemas -- both their concrete representations (perhaps in a `schema.xsd` file) and the abstract concepts in a schema as defined by the specification. As anyone who has read the schema specs knows, they are quite detailed. The XML Schema Infoset Model strives to expose all the Infoset details within any schema. This allows you to efficiently manage your schema collection, and empower higher-level schema tools such as schema-aware parsers and transformers... The XML Schema Infoset Model also includes the UML diagrams used in building the library interfaces themselves; these diagrams show the relationships between the library objects, which very closely mimic the concepts in the schema specifications..." Note: The IBM XML Schema Infoset Model "is a reference library for use with any code that examines, creates, or modifies XML Schemas (standalone or as part of other artifacts, such as XForms or WSDL documents." On October 23, 2002 IBM released a downloadable Version 1.0.1 stable build (20021023_1900TL); see the Developer FAQ and complete documentation. The earlier news item: "IBM Publishes XML Schema Infoset API Requirements and Development Code."

- [October 31, 2002] "Create Flexible and Extensible XML Schemas. Building XML Schemas in an Object-Oriented Framework." By Ayesha Malik (Senior Consultant, Object Machines). From IBM developerWorks, XML zone. October 2002. ['XML schemas offer a powerful set of tools for constraining and formalizing the vocabulary and grammar of XML documents. With XML rapidly emerging as the data transport format of the future, it is clear that the structure of the XML, as outlined by schemas, must be created and stored in an organized manner. Developers experienced in object-oriented design know that a flexible architecture ensures consistency throughout the system and helps to accommodate growth and change. This instructional article uses an object-oriented framework to show you how to design XML schemas that are extensible, flexible, and modular.'] "When leveraging established patterns of object-oriented programming in constructing XML schemas, I use the three main principles of object-oriented design: encapsulation, inheritance, and polymorphism. To help discuss object-oriented frameworks in this context, I use an example of a fictitious company, Bond

Publishing... Design patterns for decoupling: Recently, some design patterns have emerged that address decoupling and cohesiveness in XML schemas. We have already discussed how to create reusable components. Now, you'll learn how to vary the granularity of datatypes. This is similar to trying to answer the question 'How can I refactor my code and how much refactoring is appropriate for a given situation?' There are currently three design patterns that represent three levels of granularity when creating components... Object-oriented programming places a great deal of emphasis on packaging classes according to their services. The package structure organizes the code and facilitates modularity and maintenance. You can achieve similar benefits by organizing your XML schemas according to their functions... If your system is going to use XML to transport data information, either internally or externally, then you should seriously consider how to properly design your XML schemas. In this article, you have seen how to create schemas that use inheritance, encapsulation and polymorphism, and even had a glimpse of emerging design patterns in XML schema design. Leveraging these object-oriented frameworks helps you design XML schemas that are modular and extensible, maintain data integrity, and can be easily integrated with other XML protocols..."

- [October 24, 2002] Microsoft XSD Inference Tool Creates Schemas from XML Instances. A posting from Dare Obasanjo announces the availability of a Microsoft XSD Inference utility. The Beta 1 XSD Inference Tool "is used to create an XML Schema definition language (XSD) schema from an XML instance document. The input must be a well-formed XML instance document, and not an XML fragment. The output is an XML schema that can validate the instance document. When provided with well-formed XML file, the utility generates an XSD that can be used to validate that XML file. You can also refine the XSD generated by providing the tool more well-formed XML files." An interface to the tool is available online, and a binaries may be downloaded for use with Microsoft .Net Frameworks. For the online version, the total size of the file must not exceed 1 MB. Related utilities from the Microsoft 'GotDotNet' XML Tools Team include the Microsoft XML Diff and Patch tool and an XSD Schema Validator. [*Full context*]
- [October 04, 2002] A posting from Henry S. Thompson announces a "Major New Release of XSV" which reorganizes the code to make it PPC (Python Polically Correct) and adds new functionality. XSV (XML Schema Validator) is an open source GPLed work-in-progress attempt at a

conformant schema-aware processor, as defined by *XML Schema Part 1: Structures*, May 2, 2001 (REC) version. Henry says: "New functionality includes command-line settable optional invocation control of top-level element name and/or type, partial support for the 'pattern' facet. The status page provides more details, including information on the Win32 installer and running the sources -- made easier now, because there are RPMs for the underlying PyLTXML stuff... Since this is a major re-org, there's obviously a bigger chance than usual of bugs lurking, despite moderately careful regression testing -- please let me know ASAP of anything that stops working..."

- [October 03, 2002] "Working with a Metaschema." By Will Provost. From XML.com (October 02, 2002). ['Document schemas can be useful for a lot more than their primary purpose of validating document instances. For some time now, it has been popular to use a schema to construct parts of a processing application. Our main feature this week, the latest in Will Provost's XML Schema Clinic series, focuses on how schemas can be used in application construction. In particular, Will looks at how the schema for W3C XML Schemas themselves can be adapted to produce "metaschemas," allowing your applications to either restrict or extend the functionality of W3C XML Schema.'] "W3C XML Schema [here: WXS] provides a structural template that describes in detail each type and relationship: just the information an application would need, say, to build a new instance document from a data stream or to create an intuitive GUI for data entry. Given the tremendous complexity of WXS, however, applications which consume schemas face a daunting processing challenge. Often the full power of the language is neither needed nor wanted, as modeling requirements may be relatively simple, and developers don't want to be responsible for every possible wrinkle in a schema. If only we could constrain a candidate schema to use just a subset of the full WXS vocabulary... Oh, wait, we can. 'WXS vocabulary' is the tip-off: a schema is just an XML document, after all, and it can be validated like any other. What we need, in other words, is a schema for our schema. In this article we'll investigate the uses of metaschemas and the techniques for creating them. This will bring us in close contact with the existing WXS metamodel, an interesting study in and of itself. We'll consider several strategies for bending this metamodel to our application's purposes, and we'll see which strategies best suit which requirements. To tip the hand a bit, the prize will go to the WXS redefine component as a way of redefining parts of the WXS metamodel itself... This system isn't perfect. There are many

ways in which I'd like to leverage the WXS metamodel that are either closed to me or just too complicated to be worth the trouble. This isn't a shortcoming in WXS, as I see it; if the type model were as pliable as I'd like it to be, it just wouldn't be W3C XML Schema and wouldn't have the tremendous descriptive power and precision that I also want. Where they are feasible, redefinitions of schema components offer an elegant way to tailor the WXS model to the needs of an application. XPath/XSLT validation can provide another option, but it's important to see past logistics and remember that the WXS metamodel is as stiff as it is for a reason. If you find yourself demanding features in your application's candidate schema that make them malformed under WXS proper, or changing so many things that the metamodel is unrecognizable, you should probably be building your metamodel from scratch or working from a different starting point." See also *XML Schema: The W3C's Object-Oriented Descriptions for XML,* by Eric van der Vlist.

- [October 01, 2002] "Guide to the XML Schema Specifications." Techquila's Topicmap-powered browser for the W3C XML Schema Specifications. By Kal Ahmed (Techquila). October 01, 2002. "The W3C XML Schema standards are often accused of being over-complex and difficult to read. In an attempt to assist those trying to find their way around the W3C specifications, I have created a multi-modal topic map of the specifications. In this topic map you will find indexes of the terms used by the specifications and the main concepts of XML Schema. The topic maps are primarily created automatically using MDF to process the XML Schema specifications and the schema for XML Schema. The topic maps are then integrated by merging them with a hand-crafted topic map created using TMTab. A static HTML site has been created from the topic maps and can be browsed [online]. The application that produces this HTML output is based on TM4J and Jakarta Velocity. For more information about the creation and publication of topic maps using open-source and free software; or to get the topic map files themselves, please contact Kal Ahmed directly..." [From the posting: "I've spent a little time creating topic maps from the XML Schema family of specifications. The HTML-ized result is now [online]... This is a first stab at trying to topicmap the specs so comments on both presentation and content would be very welcome. For the topic map nerds, the XTM files are available; send me an email if you would like them. My thanks to Jeni Tennison who provided some really helpful hints in getting me started on this project (though all the mistakes are mine!)."]
- [September 30, 2002] "Data Interchange

Standards Association Develops New Software Tool to Ease XML Schema Documentation." - "The Data Interchange Standards Association (DISA) released a report on the Componetizer, a new software tool built by DISA that automatically identifies and documents data items in the electronic rules for XML documents, called XML Schemas. The eXtensible Markup Language (XML) is a high-powered standard format, developed by the World Wide Web Consortium or W3C, for exchanging business messages and other structured data over the Internet. The Componetizer dramatically reduces the time and effort needed to document XML Schemas, an often-laborious process in the development of standard business messages written in XML. While XML Schemas can be powerful, they are written in XML syntax, which is machine-readable but sometimes difficult to understand by humans. The Componetizer scans XML Schemas, identifies the individual data items in those schemas, and then arrays the Schema components in easy-to-read tables viewable with any Web browser... The Componetizer works with XML Schemas meeting the requirements of the W3C's XML Schema 1.0 Recommendation of May 2, 2001. DISA has used the Componetizer to generate documentation for some of its current industry affiliates, and plans to expand the outputs to other visual display and database formats..." See "Componetizer: A Tool for Extracting and Documenting XML Schema Components," by Marcel Jemio and Alan Kotok.

- [September 16, 2002] Altova Introduces New XML Product Line for Design and Development. Altova Inc. has released a new XML product line consisting of three easy-to-use software tools designed to facilitate and advance the adoption of XML technologies. The XMLSPY 5 tool "builds on the previous XMLSPY version by adding XSLT debugging, WSDL editing, Java/C++ code generation, HTML Importing, and Tamino Integration. Altova's AUTHENTIC 5 is a standards-based browser enabled document editor; it allows business users to seamlessly capture thoughts and ideas directly in XML content for storage in any content management system, database, or XML repository, for later retrieval or transformation, unlocking corporate knowledge. The STYLEVISION 5 XML tool supports web developers by providing powerful conversion utilities for painless migration of traditional HTML websites to true XML-based sites; it consists of XSLT stylesheets, XML Schema/DTD, and XML content." Each of the tools is available from the Altova website for free trial download and evaluation. [*Full context*]
- [September 05, 2002] IBM alphaWorks Updates

XML Schema Quality Checker. A posting from Achille Fokoué (XML/XSL Transformational Systems, IBM T.J. Watson Research Center) announces the release of the IBM XML Schema Quality Checker version 2.1.1 from IBM Alphaworks. XML Schema Quality Checker (SQC) "is a program which takes as input documents containing XML Schemas written in the W3C XML schema language and diagnoses improper uses of the schema language. Where the appropriate action to correct the schema is not obvious, the diagnostic message may include a suggestion about how to make the fix. For Schemas which are composed of numerous schema documents connected via `<include>`, `<import>` or `<redefine>` element information items, a full schema-wide checking is performed. The tool can also be run in batch mode to quality check multiple XML schemas in a single run. SQC may be installed as an Eclipse or WSAD Plugin." Changes in version 2.1.1 include: (1) improved error detection; (2) implementation of fixes based upon the W3C 'XML Schema 1.0 Specification Errata' document; (3) SchemaQualityChecker is now using Apache's XERCES-J version 2.1; (4) Additional command line options; (5) Eclipse progress meter. [*Full context*]

- [August 30, 2002] "Validation by Instance." By Michael Fitzgerald. From XML.com. (August 28, 2002). ['Michael Fitzgerald shows a convenient way to write schemas for validating XML documents. Rather than modeling the schema from scratch, Michael shows how to derive schemas (DTDs, RELAX NG, and W3C XML Schema) from instance documents.'] "Most people these days develop XML documents and schema with a visual editor of some sort, perhaps Altova's XML Spy, Tibco's TurboXML, xmlHack from SysOnyx, or Oxygen. Some even use several editors on a single project, depending on the strengths of the software. Others prefer to work closer to the bone. I usually develop my schema and instances by hand, using the vi editor, along with other Unix utilities (actually, I use Cygwin on a Windows 2000 box). I don't want to make more work for myself, but I prefer to use free, open source tools that allow me to make low-level changes that suit my needs. If you prefer to work this way, you should enjoy this piece. In this article, I will explore how you can translate an XML document into a Document Type Definition (DTD), a RELAX NG schema, and then into an W3C XML Schema (WXS) schema, in that order. I'll do this with the aid of several open source tools, and I'll also cover a way to validate the original XML instance against the various schemas. [1] Translating the DTD to RELAX NG: James Clark's DTDinst is a Java tool that

translates a DTD either into its own XML vocabulary or into a schema in RELAX NG's XML syntax. After downloading and installing dtdinst.jar, you can issue the following command to translate a DTD into RELAX NG: [2] Translating an XML Document into a DTD: To translate the XML document into a DTD, I'll use Michael Kay's DTDGenerator. Originally, DTDGenerator was part of the Saxon XSLT processor, but now it is separate. At just 17kb, it's a pretty small download. DTDGenerator does a fair amount of work for you, but it doesn't produce parameter entities, notation declarations, or entity declarations. It's also not namespace-aware, but DTDs aren't inherently aware of namespaces or qualified names anyway. [3] Translating RELAX NG to XML Schema: Trang is a another tool written by James Clark. It can take as input a schema written in RELAX NG XML and compact syntax; it can produce RELAX NG XML, RELAX NG compact syntax, DTD, and WXS as output. After downloading Trang (which includes a JAR file for Jing, a RELAX NG validator), unzipping and installing it, you can convert the RELAX NG schema back to a DTD new-event.dtd ... If you work on the Windows platform, I have also written a set of batch files that will perform all the translations (from instance, to DTD, to RELAX NG, and finally to W3C XML Schema) and then validate against them in one simple step... Using the tools I've described here, you can perform the conversions and validate against the resulting schemas in a matter of seconds. You may still prefer to use a visual editor, but I believe that learning and using these tools can save you time and money..."

- [August 08, 2002] "UML For W3C XML Schema Design." By Will Provost. From XML.com. August 07, 2002. ['Will Provost offers a UML profile for W3C XML Schema'] "Even with the clear advantages it offers over the fast-receding DTD grammar, W3C XML Schema (WXS) cannot be praised for its concision. Indeed, in discussions of XML vocabulary design, the DTD notation is often thrown up on a whiteboard solely for its ability to quickly and completely communicate an idea; the corresponding WXS notation would be laughably awkward, even when WXS will be the implementation language. Thus, UML, a graphical design notation, is all the more attractive for WXS design. UML is meant for greater things than simple description of data structures. Still the UML metamodel can support Schema design quite well, for wire-serializable types, persistence schema, and many other XML applications. UML and XML are likely to come in frequent professional contact; it would be nice if they could get along. The highest possible degree of

integration of code-design and XML-design processes should be sought. Application of UML to just about any type model requires an extension profile. There are many possible profiles and mappings between UML and XML, not all of which address the same goals. The XML Metadata Interchange and XMI Production for W3C XML Schema specifications, from the OMG, offer a standard mapping from UML/MOF to WXS for the purpose of exchanging models between UML tools. The model in question may not even be intended for XML production. WXS simply serves as a reliable XML expression of metadata for consumption in some other tool or locale. My purpose here is to discuss issues in mapping between these two metamodels and to advance a UML profile that will support complete expression of an WXS information set... The major distinction is that XMI puts UML first, so to speak, in some cases settling for a mapping that fails to capture some useful WXS construct, so long as the UML model is well expressed. My aim is to put WXS first and to develop a UML profile for use specifically in WXS design: (1) The profile should capture every detail of an XML vocabulary that an WXS could express. (2) It should support two-way generation of WXS documents. I suggest a few stereotypes and tags, many of which dovetail with the XMI-Schema mapping. I discuss specific notation issues as the story unfolds, and highlight the necessary stereotypes and tags... David Carlson [*Modeling XML Applications with UML: Practical e-Business Applications*] has also done some excellent work in this area, and has proposed an extension profile for this purpose. I disagree with him on at least one major point of modeling and one minor point of notation, but much of what is developed here lines up well with Carlson's profile..." See references in: (1) "Conceptual Modeling and Markup Languages"; (2) "XML and 'The Semantic Web'."

- [August 05, 2002] "Not My Type: Sizing Up W3C XML Schema Primitives." By Amelia Lewis. From XML.com. July 31, 2002. ['Continuing our occasional series of opinion pieces from members of the XML community, Amy Lewis takes a hard look at W3C XML Schema datatypes.'] "Since the application of XML to data representation first gained public visibility, there has been a movement to enhance its type system beyond that originally provided by DTD. Several attempts were made (SOX, XML Data and XML Data Reduced, Datatypes for DTDs, and others) before the W3C handed the problem to the XML Schema Working Group. What is the goal of data type definitions for XML? For one thing, it establishes "strong typing" in XML in a fashion that corresponds with strong typing in programming languages. Various

commercial interests have been vocal supporters of strong typing in XML because they see typed generic data representation as their best hope for interoperability and increased automation. With typing in schemas extended into the textual content of simple types, and not just the structural content of complex types, businesses can enforce contracts for data exchange. In other words, strong typing enables electronic commerce. To phrase it a little differently, the data types defined in DTDs were considered inadequate to support the requirements of electronic commerce or, more generally, of commercially reliable electronic information exchange. The publication of W3C XML Schema (or WXS), in which one half of the specification was devoted to the definition of a type library (part two), seemed to resolve the problem. Certainly, with forty-four built-in data types, nineteen of them primitive, it seemed at first glance to cover the field. The increasing visibility of WXS and the efforts to layer additional specifications on top of it -- XML Query, the PSVI, data types in XPath 2.0, typing in web services -- have begun to raise serious questions about WXS part two, even among proponents of strong types, including the author of this article. There are two fundamental problems with WXS datatyping. The first is its design: it's not a type system -- there is no system -- and not even a type collection. Rather, it's a collection of collections of types with no coherent or consistent set of interrelations. The second problem is a single sentence in the specification: 'Primitive datatypes can only be added by revisions to this specification'. This sentence exists because of the design problem; lacking a concept for what a primitive data type is, the only way to define new types is by appeal to authority. The data type library is wholly inextensible, internally inconsistent, bloated in and incomplete for most application domains..." 'Related Reading' from O'Reilly includes *XML Schema: The W3C's Object-Oriented Descriptions for XML*, by Eric van der Vlist.

- [August 03, 2002] "XML to Relational Conversion using Theory of Regular Tree Grammars." By Murali Mani and Dongwon Lee. In Proceedings of the VLDB Workshop on Efficiency and Effectiveness of XML Tools, and Techniques (EEXTT), Hong Kong, China, August 2002. 12 pages, with 17 references. "In this paper, we study the different steps of translation from XML to relational models, while maintaining semantic constraints. Our work is based on the theory of regular tree grammars, which provides a useful formal framework for understanding various aspects of XML schema languages. We first study two normal form representations for regular tree grammars. The first normal form representation,

called NF1, is used in the two scenarios: (a) Several document validation algorithms use the NF1 representation as the first step in the validation process for efficiency reasons, and (b) NF1 representation can be used to check whether a given schema satisfies the structural constraints imposed by the schema language. The second normal form representation, called NF2, forms the basis for conversion of a set of type definitions in a schema language L1 that supports union types (*e.g.*, XML-Schema), to a schema language L2 that does not support union types (*e.g.*, SQL), and is used as the first step in our XML to relational conversion algorithm..." General references: "XML Schemas." [cache]

- [August 03, 2002] "NeT and CoT: Translating Relational Schemas to XML Schemas Using Semantic Constraints." By Dongwon Lee, Murali Mani, Frank Chiu, and Wesley. W. Chu. Paper prepared for VLDB 2002 (28th International Conference on Very Large Data Bases). "The paper introduces two algorithms, called NeT and CoT, to translate relational schemas to XML schemas using various semantic constraints are presented. The XML schema representation we use is a language-independent formalism named XSchema, that is both precise and concise. A given XSchema can be mapped to a schema in any of the existing XML schema language proposals. Our proposed algorithms have the following characteristics: (1) NeT derives a nested structure from a flat relational model by repeatedly applying the nest operator on each table so that the resulting XML schema becomes hierarchical, and (2) CoT considers not only the structure of relational schemas, but also semantic constraints such as inclusion dependencies during the translation - it takes as input a relational schema where multiple tables are interconnected through inclusion dependencies and converts it into a 'good' XSchema. To validate our proposals, we present experimental results using both real schemas from the UCI repository and synthetic schemas from TPC-H." See similarly "NeT and CoT: Inferring XML Schemas from the Relational World", by Dongwon Lee, Murali Mani, Frank Chiu, and Wesley. W. Chu; in *Proceedings of ICDE 2002*, San Jose, California, February 2002. General references: "XML Schemas." [source]
- [July 27, 2002] "Data Modeling using XML Schemas." By Murali Mani (Computer Science Dept, UCLA). Presentation to be given Wednesday, August 7, 2002 as a Nocturne at the Extreme Markup Languages Conference 2002. "XML appears to have the potential to make significant impact on database applications, and XML is already being used in several database

applications. One of the main reasons for this is the 'superiority' of XML schemas for data modeling - recursion and union types are easily specified using XML schemas. In order to do data modeling effectively, we should be study it systematically. A data model has three constituents to it - structural specification, constraint specification, and operators for manipulating and retrieving the data. Regular tree grammar theory has established itself as the basis for structural specification for XML schemas. Constraint specification is still being studied, and we have approaches such as 'path-based constraint specification' and 'type-based constraint specification', with strong indications of type-based constraint specification as a very suitable candidate. Operators are available as part of XPath, XSLT, XQuery etc. In this talk, we would like to mention about the two ways of specifying contraints - path-based and type-based. Then we would like to describe how we can specify entities and relationships using regular tree grammar theory, and with type-based constraint specification. Furthermore, we would like to talk about an issue which is attracting attention of late -- subtyping required for XML processing. There are two techniques for subtyping in XML Schemas -- explicit as in W3C XML Schema or implicit as in XDuce. The main results here are: (a) The two subtyping schemes are incompatible with each other, and (b) There are open and interesting issues in doing implicit subtyping..." Extreme 2002 Conference will be held August 4 - 9, 2002 in Montréal.

- [July 23, 2002] A posting from James Clark announces an update for RELAX NG resources, availble from the Thai Open Source Software Center. From the posting: "I've updated jing, trang and dtdinst. Trang now has experimental support for generating W3C XML Schema. DTDinst has a new option `-i` for inlining attribute list declarations; this makes its generated output work better as input for generating W3C XML Schema. Jing has a couple of minor bug fixes... There are still lots of things I want to add to the trang XSD output module. Feedback on what improvements are most needed is welcome... The XML Schema support (provisional) has several limitations..." See "RELAX NG."
- [July 20, 2002] "The Essence of XML." By Jérôme Siméon (Bell Laboratories) and Philip Wadler (Avaya Labs). Invited paper prepared for presentation at the Sixth International Symposium on Functional and Logic Programming (FLOPS 2002), University of Aizu, Aizu, Japan, September 15-17, 2002. 14 pages, with 22 references. Referenced on Philip Wadler's XML page. "The

World-Wide Web Consortium (W3C) promotes XML and related standards, including XML Schema, XQuery, and XPath. This paper describes a formalization XML Schema. A formal semantics based on these ideas is part of the official XQuery and XPath specification, one of the first uses of formal methods by a standards body. XML Schema features both named and structural types, with structure based on tree grammars. While structural types and matching have been studied in other work (notably XDuce, Relax NG, and a previous formalization of XML Schema), this is the first work to study the relation between named types and structural types, and the relation between matching and validation. The dichotomy between names and structures is not quite so stark as at first it might appear. Many languages use combinations of named and structural typing. For instance, in ML record types are purely structural, but two types declared with 'datatype' are distinct, even if they have the same structure. Further, relations between names always imply corresponding relations between structures. For instance, in Java if one class is declared to extend another then the first class always has a structure that extends the second. Conversely, structural relations depend upon names. For instance, names are used to identify the fields of a record... Our aim is to model XML and Schema as they exist -- we do not claim that these are the best possible designs. Indeed, we would argue that XML and Schema have several shortcomings. First, we would argue that a data representation should explicitly distinguish, say, integers from strings, rather than to infer which is which by validation against a Schema. (This is one of the many ways in which Lisp S-expressions are superior to XML.) Second, while derivation by extension in Schema superficially resembles subclassing in object-oriented programming, in fact there are profound differences. In languages such as Java, one can typecheck code for a class without knowing all subclasses of that class (this supports separate compilation). But in XML Schema, one cannot validate against a type without knowing all types that derive by extension from that type (and hence separate compilation is problematic). Nonetheless, XML and Schema are widely used standards, and there is value in modeling these standards. In particular, such models may: (i) improve our understanding of exactly what is mandated by the standard, (ii) help implementors create conforming implementations, and (iii) suggest how to improve the standards..." See also the previous/preliminary version. [cache]

- [July 05, 2002] "W3C XML Schema Design Patterns: Dealing With Change." By Dare Obasanjo. From XML.com. July 03, 2002. ['One of the key challenges in XML systems is how to cope

with change in documents. Business requirements and interactions change, and your documents need to change with them while retaining backwards compatibility. Our feature article this week, the first in a series from Dare Obasanjo, covers design patterns for W3C XML Schema that support the evolution of your XML documents over time.'] "W3C XML Schema is one to specify the structure of and constraints on XML documents. As usage of W3C XML Schema has grown, certain usage patterns have become common and this article, the first in a series, will tackle various aspects of the creation and usage of W3C XML Schema. This article will focus on techniques for building schemas which are flexible and which allow for change in underlying data, the schema, or both in a modular manner. Designing schemas that support data evolution is beneficial in situations where the structure of XML instances may change but still must be validated against the original schema. For example, several entities may share XML documents, the format of which changes over time, but some entities may not receive updated schemas. Or when you must ensure that older versions of an XML document can be validated by newer versions of the schema. Or, perhaps, multiple entities share XML documents that have a similar structure but in which significant domain specific differences. The `address.xsd` example in the W3C XML Schema Primer describes a situation in which a generic address format exists that can be extended to encompass localized address formats..."

- [June 26, 2002] "DSDL Examined." By Leigh Dodds. From XML.com. June 26, 2002. ['In his final column Leigh looks at DSDL, the ISO activity to standardise XML document validation.'] "The core of DSDL will be the Interoperability Framework (Part 1): the glue that binds together the other modules. This week Eric van der Vlist, who is the appointed editor of this section, and Rick Jelliffe have separately produced proposals that aim to explore these kind of framework structures in more detail. The two proposals, neither of which have any formal standing, take very different approaches to the same problem. Van der Vlist's XML Validation Interoperability Framework (XVIF) takes the approach of embedding validation and transformation pipelines within another vocabulary. The specification and online demonstrator both show how this could be achieved by embedding the pipelines within a schema language, but in principle the XVIF is language-neutral so could be embedded within an XSLT transformation for example. XVIF elements just rely on their container to provide the context node on which they will interact. The embedded pipelines may generate other nodes or a simple

boolean validation flag. Van der Vlist has produced a prototype that supports using pipelines containing XPath expressions, XSLT transformations, and manipulating content with simple regular expressions, or using Regular Fragmentations. In contrast, Rick Jelliffe's proposal, 'Schemachine' is closer to other pipeline frameworks such as XPipe and Cocoon in that the pipelines are defined by a separate vocabulary. In fact Jelliffe notes that the proposal borrows a lot from XPipe and Schematron in that it has a number of similar elements and structures, *e.g.*, phases. Schemachine divides pipeline elements up into particular roles such as Selectors (*e.g.*, XPath expressions), Tokenizers (*e.g.*, Regular Fragmentations) and Validators (*e.g.*, RELAX NG, Schematron). Jelliffe differentiated XVIF and Schemachine as 'innies and outies'. Technology aside, the important aspect of these proposals is the intent: publicly exploring strawman proposals and implementations to gather feedback before considering standardization. That's a path which seems not only likely to produce viable results, but may actually deliver useful tools that others benefit from in the shorter term..."

- [June 26, 2002] "Enforcing Association Cardinality." By Will Provost. From XML.com. June 26, 2002. ['Our main feature this week is the first in a new, ongoing, series focusing on W3C XML Schema, called "XML Schema Clinic." Will Provost will be examining issues in schema design and XML data modeling. In this first installment, Will discusses using W3C XML Schema to control the cardinality of associations between elements in a document type.'] "If you're like me, XML document design brings out your darker side. We like a schema whose heart is stone -- a schema that's just itching to call a candidate document on the carpet for the slightest nonconformity. None of this any-element-will-do namespace validation for us. We enjoy the dirty work: schemas, we think, are best built to be aggressive in ferreting out the little mistakes in the information set that could later confuse the more sheltered and constructive-minded XML application. In this article, we'll practice a bit of that merciless science. Specifically, we'll look at ways to control the cardinality of associations between XML elements. The basic implementation, which we'll review momentarily, of an association between two types is simple enough but is only sufficient for many-to-one relationships. What if multiple references, or zero references, to an item are unacceptable? What if a referenced item may or may not be present? These variations will require other techniques, and these are essential for the truly draconian schema author. This article will use a simple UML notation

to illustrate patterns and examples. Knowledge of both XML Schema 1.0 (Part 1 in particular) and UML is assumed, although in developing our notation we'll have a chance to review a little of both... The Unified Modeling Language (UML) provides a basis for a simple notation which will serve our needs in identifying rudimentary design patterns and in illustrating specific examples. Note that many UML-to-Schema mappings are possible; see the 'XMI Production for XML Schema' specification from the OMG for one much more formal option..."

- [June 24, 2002] IBM Clio Tool Supports Mapping Between Relational Data and XML Schemas. Clio is a Computer Science Research project at IBM's Almaden Research Lab. Its developers are designing methods to specify the transformation of legacy data to make it fit for new uses. Clio addresses the challenge of "merging and coalescing data from multiple and diverse sources into different data formats. In particular, it addresses schema matching (the process of matching elements of a source schema with elements of a target schema) and schema mapping (the process of creating a query that maps between two disparate schemas), which lie at the heart of data integration systems. Clio is a tool for generating mappings (queries) between relational and XML Schemas. The user is presented with the structure and constraints of two schemas and is asked to draw correspondences between the parts of the schemas that represent the same real world entity. Correspondences can also be inferred by Clio and verified by the user. Given the two schemas and the set of correspondences between them, clio can generates the (SQL, XSLT, or XQueries) queries that drive the translation of data conforming to the first (source) schema to data conforming to the the second (target) schema." [*Full context*]
- [June 21, 2002] "Can XML Be The Same After W3C XML Schema?" By Eric van der Vlist. From XML.com. June 19, 2002. ['Eric van der Vlist has just finished writing a book on W3C XML Schemas for O'Reilly, an experience that has given him a unique perspective on schema languages and W3C XML Schema in particular. Here Eric reflects on the nature of W3C XML Schema and how it could affect XML as a whole'] "The first question to ask is why is W3C XML Schema different? Why am I asking this question about W3C XML Schema and not about DTDs, Schematron, or RELAX NG? The short answer is datatypes and object orientation. These two aspects of W3C XML Schema are tightly coupled. Datatypes are to W3C XML Schema what classes are to object oriented programming languages.

Both promote a categorization of information into classes and subclasses, analogous to the taxonomies biologists use to classify species. Although this process of classification or derivation seems natural, it is not universal and is much less visible in other schema languages. To reuse the metaphor of species, a rule based language such as Schematron does not attempt to put a sticker on a species, but rather set of rules defining if an animal belongs to a set of 'valid' animals ('the set of animals having four legs and able to run at least 50 km/h'). Grammar based languages, including RELAX NG and DTD, describe the patterns used to build the animal ('an animal made of a body, a neck, a head, a tail, and four legs')... My reward for digging into the W3C XML Schema Recommendation has been to discover an unexpected pearl far away from the limelight: W3C XML Schema is exceedingly good at associating metadata with elements or attributes...Schematron is about rules, and RELAX NG is about patterns; neither of them describes elements or attributes as such. Schematron can define rules to be checked in the context of an element, and RELAX NG can describe a pattern containing a single element, but W3C XML Schema is the only one which can describe elements and attributes. As long as validation is your primary concern, this may not make much difference, but for attaching metadata to elements and attributes, a language which describes elements and attributes seems to be a better fit..."

- [June 17, 2002] "XInterfaces: A New Schema Language for XML." Final thesis by Oliver Nölle. Programming Languages Group, Institute for Computer Science, University of Freiburg, Germany. June 12, 2002. Thesis supervisor, Prof. Dr. Peter Thiemann. 106 pages, with 45 references. Abstract: "A new schema language for XML is proposed to enhance the interoperability of applications sharing a common dataset. An XML document is considered as a semi-structured database, which evolves over time and is used by different applications. An XInterface defines a view of an XML document by imposing constraints on structure and type of selected parts. These constraints are not grammar-based but specify an open-content model, allowing additional elements and attributes to be present anywhere in the document. This enables each application to define and validate its own view on the document, with data being shared between applications or specifically added by one application. XInterfaces feature an explicit type hierarchy, enabling easy extension of existing schemas and documents while guaranteeing conformance of the extended documents to the existing views. This allows data

evolution without breaking compatibility of existing applications. Because different applications share one document, access mechanisms are described that guarantee the validity of the document for all applications after modifications. As a proof of concept, a tool was implemented that maps XInterfaces to a class framework in Java, allowing convenient access to those parts of an XML document that are described by XInterfaces." Notes from Oliver Nölle in a posting dated June 16, 2002: "As a final thesis project I created a new schema language named 'XInterfaces'. It is a very simple language, similar to XML Schema in syntax and similar to Schematron in semantics, featuring an explicit type hierarchy which allows multiple inheritance. Although it is a very simple concept, I think it is a very useful one. I'm announcing it here as this language has some features that XML Schema currently does not have, but which I found very useful in certain scenarios (in particular, multiple inheritance and open content model). Maybe later versions of XML Schema are moving towards implementing these features and XInterfaces can be an inspiration. If you are interested, you can find my thesis, a sample validator implementation and related material on the XInterfaces home page..." See also: (1) A sample XInterface type definition which could be applied to this instance document; (2) XInterface type definition for XInterface type definitions; (3) XML Schema schema that defines the syntax of XInterface type definitions; (4) sample implementation of an XInterface schema validator. [cache]

- [June 04, 2002] "Stylesheet for Extracting Schematron Information from a RELAX-NG Schema." A posting from Eddie Robertsson references a new XSLT stylesheet: "... As long as your application isn't highly time critical you can always embedd Schematron rules within a RELAX-NG schema. Sun's MSV [Multi-Schema XML Validator] will support Schematron like rules in RELAX-NG schemas and I've just finished an XSLT stylesheet 'RNG2Schtrn.xsl' that will extract Schematron rules from a RELAX-NG schema and create a Schematron schema that can be used for validation. It works similar to the XSD2Schtrn.xsl stylesheet that extracts Schematron rules from a W3C XML Schema with one difference: Schematron rules in a RELAX-NG schema can appear anywhere between elements in the RELAX-NG namespace (in W3C XML Schema they have to be declared inside the xs:appinfo element) Like the XSD2Schtrn.xsl stylesheet the RNG2Schtrn.xsl stylesheet will also extract Schematron rules that are included in RELAX-NG modules from the main RNG schema by either

<include> or <externalRef>. I'm currently modifying my previous article on how to embedd Schematron rules in W3C XML Schema to add a new section on how this is done in RELAX-NG and it will be available shortly. If you're interested in embedding Schematron rules in a RELAX-NG schema you can still have a look at the original article ['Combining the power of W3C XML Schema and Schematron'] because the technique used for both W3C XML Schema and RELAX-NG is exactly the same..."

- [June 04, 2002] "Guidelines for the Use of XML within IETF Protocols." Updates the draft "Guidelines For The Use of XML in IETF Protocols" ['draft-hollenbeck-ietf-xml-guidelines-00.txt', April 5, 2002]. By Scott Hollenbeck (VeriSign, Inc.), Marshall T. Rose (Dover Beach Consulting, Inc.), and Larry Masinter (Adobe Systems Incorporated; WWW). Reference: 'draft-hollenbeck-ietf-xml-guidelines-04.txt'. June 4, 2002, expires: December 3, 2002. 33 pages. "The Extensible Markup Language (XML) is a framework for structuring data. While it evolved from SGML -- a markup language primarily focused on structuring documents -- XML has evolved to be a widely- used mechanism for representing structured data. There are a wide variety of Internet protocols being developed; many have need for a representation for structured data relevant to their application. There has been much interest in the use of XML as a representation method. This document describes basic XML concepts, analyzes various alternatives in the use of XML, and provides guidelines for the use of XML within IETF standards-track protocols... It is the goal of the authors that this draft (when completed and then approved by the IESG) be published as a Best Current Practice (BCP)..." Document also available in XML and plain text formats. See also the archives of the 'ietf-xml-use' mailing list, which supports a general discussion on how and when to use XML in IETF protocols. A related posting by James Clark "RELAX NG and W3C XML Schema" in response to section 4.6 of the draft ("... XML Schema should be used as the formalism in the absence of clearly stated reasons to choose another...") led to an XML-DEV thread "XML Schema considered harmful?" See comments from James Clark and Rick Jelliffe. [cache, text]
- [June 04, 2002] "Analyzing XML Schemas With the Schema Infoset Model. Easily Perform Complex Queries on Your Schemas With This Model." By Shane Curcuru (Advisory Software Engineer, IBM). From IBM developerWorks, XML

Zone. June 2002. ['As the use of schemas grows, the need for tools to manipulate schemas grows. The new Schema Infoset Model provides a complete modeling of schemas themselves, including the concrete representations as well as the abstract relationships within a schema or a set of schemas. This article will show some of the power of this library to easily query the model of a schema for detailed information about it; we could also update the schema to fix any problems found and write the schema back out.'] "Although there are a number of parsers and tools that use schemas to validate or analyze XML documents, tools that allow querying and advanced manipulation of schema documents themselves are still being built. The Schema Infoset Model (AKA the IBM Java Library for Schema Components) provides a rich API library that models schemas -- both their concrete representations (perhaps in a schema.xsd file) and the abstract concepts in a schema as defined by the specification. As anyone who has read the schema specs knows, they're quite detailed, and this model strives to expose all the details within any schema. This will then allow you to efficiently manage your schema collection, and empower higher level schema tools -- perhaps schema-aware parsers and transformers... While you can use XSLT or XPath to query a schema's concrete representation in an .xsd file or inside some other .xml content, it is much more difficult to discover the type derivations and interrelationships that schema components actually have. Since the Schema Infoset Model library models both the concrete representation and the abstract concept of the schema, it can easily be used to collect details about its components, even when the schema may have deep type hierarchies or be defined in multiple schema files... Although this is a contrived example, it does show how the library's detailed representation of a schema makes it easy to find exactly the parts of a schema you need. The library provides setter methods for the properties of schema components, so it is easy to update your sample to automatically fix any found types by adding any missing facets. And since the library models the concrete representation of the schema as well, you can write your updated schema back out to an `.xsd` file..." See "IBM Publishes XML Schema Infoset API Requirements and Development Code" and the announcement.

- [May 24, 2002] IBM Publishes XML Schema Infoset API Requirements and Development Code. A posting from Bob Schloss describes the public availability of a requirements document for an XML Schema Infoset API and code being written to produce a reference implementation for

the schema components API. The requirements document outlines "the design principles, scope, and requirements for a XML Schema Infoset and API; it includes requirements as they relate to development time and runtime software which: (1) constructs, examines or modifies schema components; (2) examines the Post Schema Validation Infoset; (3) makes use of schema components in conjunction with components that represent the infoset of other namespaces (such as WSDL or XForms). It includes requirements concerning the data model, external requirements, and coordination. The API could be used by programs such as: editors of XML instance documents which provide guidance based on a schema; tools that examine pairs of schemas; mapping tools that support non-XML data sources at one end and schema-described XML at the other; tools to visualize, create, modify and extend XML Schemas." The IBM development team is building a reference implementation for the API which is expected to be "very complete -- not simply read-only, but able to handle any XML Schema, no matter how complex." This work, including source code, UML, example usage code, and documentation is available online. [*Full context*]

- [May 17, 2002] "XML Schema Languages." By Eric van der Vlist. May 2002. Prepared for the XML Europe 2002 Tutorials. The tutorial follows the classification of XML schema languages proposed by the ISO DSDL Working Group at http://dsdl.org. (1) Introduction; (2) Rule based languages [XSLT and Schematron[; (3) Grammar based languages [RELAX NG]; (4) Object Oriented languages [W3C XML Schema]. See source from the DSDL web site.
- [May 17, 2002] "Eric van der Vlist on W3C XML Schema." By [O'Reilly Staff]. From XML.com. May 15, 2002. ['An interview with the author of O'Reilly's "XML Schema" book.'] "Eric van der Vlist, a regular contributor to XML.com, has just completed writing *XML Schema: The W3C's Object-Oriented Descriptions for XML*for O'Reilly, to be published in June 2002. In this interview he explains the importance of XML schema languages, and his motivations for writing the book. "[I have chosen this subject] "because I think that the XML schema languages in general, and W3C XML Schema in particular, are the hot topics of the moment: being at the same time essential and potentially dangerous for XML. I thought that an objective book needed to be written, which would be a kind of map to W3C XML Schema, showing clearly not only the features and goodies but also the pitfalls of this specification... the lack of XML schema languages is simply not economically acceptable! An

application must expect that the XML documents used as input follow some kind of structure, in order to be able to understand them. Formalizing this structure as 'XML schemas' enables all kind of productivity, quality and performance improvements by automating tasks such as validation, code generation, data binding, documentation and query optimization... The XML DTD was specified in the XML 1.0 recommendation, published before Namespaces in XML 1.0. The XML DTD ignores the notion of namespace and lacks the flexibility necessary to support them in a simple way. The XML DTD is also a descendant of the SGML DTD, which had been designed for document-oriented applications, and lacks a complete type system -- a requirement for data oriented applications. The W3C had the choice between updating the specification of the DTD or creating a new specification; it chose to start anew. I guess that the interoperability issues linked with any modification of the XML 1.0 recommendation have influenced this decision: it is often easier to create a new standard than to update an existing one, especially when it's a successful one! [...] DSDL proposes a classification of schema languages in three categories: (1) Rule based languages (such as Schematron), defining the rules to be followed by a class of XML documents. (2) Grammar based languages (such as RELAX NG), defining the structure of a class of XML documents as a grammar or a set of patterns. (3) Object oriented languages (disclaimer: I am the editor of this section of the DSDL work), describing a class of XML documents as object oriented structures facilitating the mapping between XML documents and object oriented applications. This classification shows that the XML schema languages are very different and could be considered more complementary than competing. If we had to define these schema languages from scratch today, with the experience we have acquired and putting aside any political considerations, I think that we could even define them as layers: a rule based language would be the foundation of a grammar based language, on top of which an object oriented language could be defined..." Note: Eric van der Vlist's book "explains XML Schema foundations, a variety of different styles for writing schemas, simple and complex types, datatypes and facets, keys, extensibility, documentation, design choices, best practices, and limitations; complete with references, a glossary, and examples throughout."

- [April 17, 2002] "Schema Centric XML Canonicalization." By Selim Aissi (Intel), Bob Atkinson (Microsoft), and Maryann Hondo (IBM). Published by UDDI.org. "Copyright (c) 2000-2002

by Accenture, Ariba, Inc., Commerce One, Inc., Compaq Computer Corporation, Fujitsu Limited, Hewlett-Packard Company, i2 Technologies, Inc., Intel Corporation, International Business Machines Corporation, Oracle Corporation, SAP AG, Sun Microsystems, Inc., VeriSign, Inc., and / or Microsoft Corporation." Version 1.0. Working Draft 13-February-2002. An editors' working draft copy circulated for general review, comment, and feedback. Version URL: http://www.uddi.org/pubs/SchemaCentricCanonicalization-20020213.htm. Latest version URL: http://www.uddi.org/pubs/SchemaCentricCanonicalization.htm. "Existing XML canonicalization algorithms such as Canonical XML and Exclusive XML Canonicalization suffer from several limitations and design artifacts (enumerated herein) which significantly limit their utility in many XML applications, particularly those which validate and process XML data according to the rules of and flexibilities afforded by XML Schema. The Schema Centric Canonicalization algorithm addresses these concerns... The Schema Centric Canonicalization algorithm is intended to be complementary in a hand-in-glove manner to the processing of XML documents as carried out by the assessment of schema validity by XML Schema, canonicalizing its input XML instance with respect to *all* those representational liberties which are permitted thereunder. Moreover, the specification of Schema Centric Canonicalization heavily exploits the details and specification of the XML Schema validity-assessment algorithm itself. In XML Schema, the analysis of an XML instance document requires that the document be modeled at the abstract level of an information set as defined in the XML Information Set recommendation. Briefly, an XML document's information set consists of a number of information items connected in a graph. An information item is an abstract description of some part of an XML document: each information item has a set of associated named properties... Properties on each of these items, for example the [children] property of element information items, connect together items of different types in an intuitive and straightforward way. The representation of an XML document as an infoset lies in contrast to its representation as a node-set as defined in XPath. The two notions are conceptually quite similar, but they are not isomorphic. For a given node-set it is possible to construct a semantically equivalent infoset without loss of information; however, the converse is not generally possible. It is the infoset abstraction which is the foundation of XML Schema, and it is therefore the infoset abstraction we use here as the foundation on which to construct Schema

Centric Canonicalization algorithm. The Schema Centric Canonicalization algorithm consists of a series of steps: creation of the input as an infoset, character model normalization, processing by XML-Schema assessment, additional infoset transformation, and serialization..." [cache]

- [April 12, 2002] "Beyond W3C XML Schema." By Will Provost. From XML.com. April 10, 2002. ['XSLT has proven to be a very successful technology, and has moved beyond the relatively narrow scope that drove its design. Even James Clark, XSLT's primary designer, never imagined the many uses XSLT would find. In "Beyond W3C XML Schema" Will Provost demonstrates that there are some aspects of document validation beyond the capabilities of W3C XML Schema. He describes the use of XPath with XSLT for reaching into documents and checking those constraints that can't be enforced with a schema.'] "The XML developer who needs to validate documents as part of application flow may choose to begin by writing W3C XML Schema for those documents. This is natural enough, but W3C XML Schema is only one part of the validation story. In this article, we will discover a multiple-stage validation process that begins with schema validation, but also uses XPath and XSLT to assert constraints on document content that are too complex or otherwise inappropriate for W3C XML Schema. We can think of a schema as both expressive and prescriptive: it describes the intended structure and interpretation of a type of document, and in the same breath it spells out constraints on legal content. There is a bias toward the expressive, though: W3C XML Schema emphasizes "content models", which are good at defining document structure but insufficient to describe many constraint patterns. This is where XPath and XSLT come in: we'll see that a transformation-based approach will let us assert many useful constraints and is in many ways a better fit to the validation problem. (In fact, one might define schema validation as no more than a special kind of transformation; see the paper of van der Vlist.) We'll begin by looking at some common constraint patterns that W3C XML Schema does not support very well and then develop a transformation-based approach to solving them... XPath and XSLT can form a second line of defense against invalid data. The value of this second stage in the validation architecture will be judged by what it can do that W3C XML Schema cannot. Here's a short list of constraint patterns XPath can express well. (1) Structural analysis of the tree as a whole; (2) Weakly-typed designs; (3) Finer control over use of subtypes -- say base types A and B are associated but subtype A2 should only see

instances of B2, not B1 or B3, etc.; (4) Single values based on numeric or string calculation -- a number that must be a multiple of three, a string that must list values in a certain order; (5) Relationships between legal single values -- a checksum over a long list of values, or a rule limiting the total number of occurrences of a common token; (6) Constraints that span multiple documents -- for instance a dynamic enumeration where the legal values are listed in a second document, and so cannot be hardcoded into a schema... We've discovered a multi-stage validation architecture based entirely on W3C-standardized technology. Out in the world, another popular transformation-based approach is Schematron, an open source tool which specifies constraint definitions in its own language. Its vocabulary simplifies the XSLT structure shown in the previous section and relies on XPath for its constraint expressions. It also allows for both 'positive' and 'negative' assertions. The big difference is that a Schematron schema must be pre-compiled, or 'pre-transformed' if you will, into a validating stylesheet, which once created is the true counterpart to the pure-XSLT transformations used here..."

- [March 29, 2002] "W3C XML Schema Needs You." By Leigh Dodds. From XML.com. March 27, 2002. ['One of the consequences of complexity in an open specification is a decreased likelihood of interoperability in implementation. XML developers have been bumping into this problem with the W3C XML Schema language recently. Leigh Dodds covers these problems, and a call for developers to aid the progress of greater interoperability.] "The W3C XML Schema (XSD) specifications have drawn fire again recently, with a number of concerns being aired about an apparent lack of interoperability between implementations. Jonathan Robie, a member of the Schema Working Group, has issued a rallying cry for developers to unite and help push for interoperability... There was a resurgence of the 'XML Schema is too complex' debate on XML-DEV last week. While this is an oft debated topic, the issues have had a slightly different slant this time around with claims that XSD is so complex that it's proving extremely difficult to implement... A few constructive suggestions were circulated during the discussion, some more radical than others. Rob Griffin suggested producing a list of standard error messages for validators, which ought to help achieve some level of consistency across implementations, as well as clarifying the circumstances in which each error should arise. Andrew Watt recommended the addition of a use case document that would provide an additional means of tackling the specifications. Watt pointed

to the XML Query documents as a good exemplar. Rick Jelliffe's suggestion to modularize XML Schema was the most radical. Jelliffe suggested that instead of a rewrite the schema specifications should be split into eight small sections which '...would allow greater modularity, let readers and implementers concentrate and advertise conformance on different parts, and fit in with ISO DSDL, for users who, say, want to use RELAX NG with XML Schemas primitive datatypes'. Jelliffe also commented that rather than criticizing XML Schema, the important first question should be to consider which schema language or combination of languages is most suited to a particular application domain. Jelliffe offered a prediction that document oriented systems will likely settle on DSDL, while database oriented applications will find XML Schemas most suitable..."

- [March 18, 2002] "Mastering XML Schemas." By Elliotte Rusty Harold. Presented at "XML & Web Services 2002 Conference" (Queen Elizabeth II Conference Centre, London, March 2002). [Tuesday 12 March, 2002.]
- [February 19, 2002] "XML Schema and RELAX NG Element Comparison." By Michael Fitzgerald. Reference posted to the RELAX NG TC list. "This document briefly compares XML Schema's 42 elements with RELAX NG's 28 elements. In the table that follows, the first column lists all the XML Schema elements while the second column lists any RELAX NG elements that have a one-to-one relationship, a comparable purpose, or only a roughly similar purpose to XML Schema elements. Elements unique to each language are also listed in separate tables below..." ['I have made an attempt to briefly compare the purpose of XML Schema's elements with RELAX NG's elements. The comparison appears in three tables totaling about 2 and 1/2 pages printed. I would appreciate any comments you have about this document...'] Note also the relax ng links on the Wy'east Communications web site.
- [February 14, 2002] Clark Updates Jing - A RELAX NG Validator in Java. James Clark has announced a new version of Jing with significant changes and revised documentation. Jing version '2002-02-13' implements the final RELAX NG 1.0 Specification and also implements parts of RELAX NG DTD Compatibility, specifically checking of ID/IDREF/IDREFS. James has "almost completely rewritten the validator using an improved algorithm. In the old algorithm, the state of the validation was represented by a stack of sets of patterns; in the new algorithm, the state is represented by a single pattern... The new release includes a documented API for Jing; in fact there are two APIs, a native API and JARV. James has

rewritten the description of derivative-based validation to correspond to what's been implemented and to incorporate feedback received on the previous version from Murata-san and Kawaguchi-san... The Jing implementation is available for download as a JAR file and as a Win32 executable for use with the Microsoft Java VM. [*Full context*]

- [February 14, 2002] IBM Releases XML Schema Quality Checker Version 2.0. IBM alphaWorks labs has released an enhanced version of its XML Schema Quality Checker (SQC). SQC is "a program which takes as input documents containing XML Schemas written in the W3C XML schema language and diagnoses improper uses of the schema language. Where the appropriate action to correct the schema is not obvious, the diagnostic message may include a suggestion about how to make the fix. The updated version now provides direct validation of embedded schemas like those which may appear in WSDL documents or XForms. It includes bug fixes and now uses Xerces version 2.0.0. For Eclipse or WSAD users, the IBM XML Schema Quality Checker can now be installed as an Eclipse or WSAD plugin; it can also still be run as a standalone command line program." [*Full context*]
- [February 12, 2002] dtd2xs version 1.54. Announcement posted by Joerg Rieger. "We are pleased to announce a new release of 'DTD to XML Schema translator' [dtd2xs]. One may use it to translate a Document Type Definition (XML 1.0 DTD) into an XML schema (REC-xmlschema-1-20010502). The translator can map meaningful DTD entities onto named and therefore reusable XML Schema constructs such as `<simpleType>`, `<attributeGroup>` and `<group>`. The translator can map DTD comments onto XML Schema `<documentation>` elements. Freely available as a Java class, as a Web tool, and as Java application..."
- [February 11, 2002] "Combining the Power of W3C XML Schema and Schematron." By Eddie Robertsson. "This article shows how to combine W3C XML Schema and Schematron by inserting Schematron rules in the `<xs:appinfo>` element of the W3C XML Schema... After the W3C ratified W3C XML Schema as a full recommendation on May 2nd 2001 it has become clear that this is the most popular XML Schema language for developers. Many believed that W3C XML Schema would solve all the problems that existed with validation of XML documents but this was never the goal of W3C XML Schema... When W3C XML Schema is not powerful enough there are other options for developers. One option is to find a different XML Schema language that can express all the needed constraints. Another option

is to add extra code to your application to check the things not expressible in the W3C XML Schema language. A third option, made available through one of W3C XML Schema's extension mechanisms, is to combine W3C XML Schema with another XML Schema language. This article will provide an explanation and several examples of how Schematron rules can easily be embedded within W3C XML Schemas. Schematron has its strengths where W3C XML Schema has its weaknesses (co-occurrence constraints) and its weaknesses where W3C XML Schema has its strengths (structure and data types). In the examples provided W3C XML Schema is used as far as possible and then the embedded Schematron rules are used to express what cannot be done with W3C XML Schema alone. The following four areas, which W3C XML Schema does not fully address, will be covered: dependant attributes, interleaving of elements, co-occurrence constraints and relationships between different XML documents. A short introduction to Schematron is provided but the reader will need a basic understanding of W3C XML Schema to benefit from the article..." [Posted note on 'xmlschema-dev@w3.org': "I'm currently working on a paper that will explain the details of embedding Schematron rules in the <xs:appinfo> element in a W3C XML Schema. I've put up a draft which contains some background, introduction to Schematron, examples of embedded Schematron rules and how the validation process works. The draft also contains a link to a zip file with all the examples used so you can try it out yourself. All comments are welcome..."

- [January 24, 2002] "Relax NG, Compared." By Eric van der Vlist. From XML.com. January 23, 2002. ['The RELAX NG schema language explained and compared to W3C XML Schemas.] "This article is a companion to two different works already published on XML.com: my introduction to W3C XML Schema is a tutorial introducing the language's main features, with a progression which I hope is intuitive; and my comparison between the main schema languages, an attempt to provide an objective and practical feature-by-feature comparison between XML schema languages. In this new article, I have taken the same approach as the one used in the W3C XML Schema tutorial but this time I've implemented the schemas using RELAX NG... it provides a good starting point for those of us who know W3C XML Schema and want to quickly point out the differences with RELAX NG. Links are provided throughout to the corresponding sections of the W3C XML Schema tutorial, and you are encouraged to follow both simultaneously... Throughout this comparison, we have seen that

one of the main differences between the two languages is a matter of style: while RELAX NG focuses on generic 'patterns', W3C XML Schema has differentiated these patterns into a set of distinct components (elements, attributes, groups, complex and simple types). The result is on one side a language which is lightweight and flexible (RELAX NG) and on the other side a language which gives more 'meaning' or 'semantic' to the components that it manipulates (W3C XML Schema). The question of whether the added features are worth the price in terms of complexity and rigidity is open, and the answer probably depends on the applications. Independently of this first difference between the two, the different positions regarding 'non-determinism' between RELAX NG, which accepts most of the constructs a designer can imagine, and W3C XML Schema, which is very strict, mean that a number of vocabularies which can be described by RELAX NG cannot be described by W3C XML Schema. A way to summarize this is to notice that an implementation such as MSV (the 'Multi Schema Validator' developed by Kohsuke Kawaguchi for Sun Microsystems) uses a RELAX NG internal representation as a basis to represent the grammar described in W3C XML Schema and DTD schemas. This seems to indicate that RELAX NG can be used as the base on which object oriented features such as those of W3C XML Schema can be implemented. The value of an XML-specific object-oriented layer is still to be determined, though, since generic object-oriented tools should be able to generate RELAX NG schemas directly..." See W3C XML Schema and "RELAX NG."

- [January 09, 2002] "XML and WebSphere Studio Application Developer. Part 1: Developing XML Schema." By Christina Lau (Senior Technical Staff Member, IBM Toronto Lab). In *IBM WebSphere Developer Technical Journal* (December 30, 2001). "IBM's WebSphere Studio Application Developer is a new application development product that supports the building of a large spectrum of applications using different technologies such as JSP, servlets, HTML, XML, Web services, databases, and EJBs. This is the first of a series of articles that will focus on the XML tools provided with Application Developer. This article covers the XML Schema Editor. It provides a birds-eye view of the XML Schema Editor that is included in WebSphere Studio Application Developer. In future articles, we will cover more advanced topics such as: (1) Creating schemas from multiple documents; (2) Identity constraints; (3) Generating Java beans from XML Schema; (4) Generating XML documents from XML Schema; (5) How the wildcard works. The

XML Schema Editor is a visual tool that supports the building of XML Schema that conforms to the XML Schema Recommendation Specification (May 2001)..."

- [December 20, 2001] "XML Schema tome 0: Introduction." Recommandation du W3C du 2 Mai 2001. From W3C XML Schema Recommendation Part 0. Translated by Jean-Jacques Thomasson. Referenced by Eric van der Vlist: "It's my pleasure to announce the publication of an excellent translation of XML Schema part 0 by Jean-Jacques Thomasson on XMLfr...". Note: "Ce document est une traduction de la recommandation XML Schema du W3C, datée du 2 mai 2001. Cette version traduite peut contenir des erreurs absentes de l'original, introduites par la traduction elle-même." See the W3C web site for XML Schema.
- [December 13, 2001] "Comparing XML Schema Languages." By Eric van der Vlist. From XML.com. December 12, 2001. ['DTDs, W3C XML Schema, RELAX NG: what's the difference? And which is the best tool for the job? There is a healthy ecology in XML schema technologies: ranging from DTDs, through the W3C's XML Schema Definition Language to newer entrants such as RELAX NG and Schematron. In his article, Eric gives us a timeline of XML schema languages, and compares the strengths of each of these technologies. Eric's talk in Orlando "XML Schema Languages" was standing-room only.'] "This article explains what an XML schema language is and which features the different schema languages possess. It also documents the development of the major schema language families -- DTDs, W3C XML Schema, and RELAX NG -- and compares the features of DTDs, W3C XML Schema, RELAX NG, Schematron, and Examplotron... The English language definition of schema does not really apply to XML schema languages. Most of the schema languages are too complex to 'present to the mind' or to a program the instance documents that they describe, and, more importantly and less subjectively, they often focus on defining validation rules more than on modeling a class of documents. All XML schema languages define *transformations* to apply to a class of instance documents. XML schemas should be thought of as transformations. These transformations take instance documents as input and produce a validation report, which includes at least a return code reporting whether the document is valid and an optional *Post Schema Validation Infoset (PSVI)*, updating the original document's infoset (the information obtained from the XML document by the parser) with additional information (default values, datatypes, etc.) One important consequence of realizing that XML

schemas define transformations is that one should consider general purpose transformation languages and APIs as alternatives when choosing a schema language... One of the key strengths of XML, sometimes called 'late binding,' is the decoupling of the writer and the reader of an XML document: this gives the reader the ability to have its own interpretation and understanding of the document. By being more prescriptive about the way to interpret a document, XML schema languages reduce the possibility of erroneous interpretation but also create the possibility of unexpectedly adding 'value' to the document by creating interpretations not apparent from an examination of the document itself. Furthermore, modeling an XML tree is very complex, and the schema languages often make a judgment on 'good' and 'bad' practices in order to limit their complexity and consequent validation processing times. Such limitations also reduce the set of possibilities offered to XML designers. Reducing the set of possibilities offered by a still relatively young technology, that is, premature optimization, is a risk, since these 'good' or 'bad' practices are still ill-defined and rapidly evolving. The many advantages of using and widely distributing XML schemas must be balanced against the risk of narrowing the flexibility and extensibility of XML. There are currently no perfect XML Schema languages. Fortunately, there are a number of good choices, each with strengths and weaknesses, and these choices can be combined. Your job may be as simple as picking the right combination for your application."

- "Schema Languages Comparison." Thursday, December 13, 2001. XML 2001 Conference. 7:30 PM - 9:00 PM. Town Hall Meeting. Moderator: Lauren Wood. Analysts: John Cowan, Norman Walsh. DTD Team: Tommie Usdin (lead), Debbie Lapeyre, Steve de Rose. RELAX NG: James Clark, Murata Makoto. Schematron: Rick Jelliffe, Eddie Robertsson, Francis Norton. W3C Schemas: Henry Thompson (lead), Martin Gudgin, Priscilla Walmsley. An opportunity to find out more about the strengths and weaknesses of currently available schema languages This Town Hall will give you more information about when to use W3C Schemas, Schematron, RELAX NG or DTDs. This session bring together experts in four different schema languages, so that you can learn the strengths and weaknesses, and when each schema language is likely to be most useful. Schema language experts in four languages were asked to participate and they formed four teams, one each for W3C Schemas, Schematron, RELAX NG and DTDs. Each team created schemas for a set of document types (also created by the teams). At the Town Hall meeting, two analysts will report on the schemas, and then the open

discussion will begin. Attendees will be able to ask questions of the team members and the analysts. Note that not all team members from all teams are able to attend the session, although all participated in the design of the schemas..."

- [December 11, 2001] SoftQuad's Enhanced XMetaL 3 Supports W3C XML Schema and Collaborative Authoring. An announcement from SoftQuad Software describes the January 2002 release of XMetaL 3, Softquad's flagship XML content creation software. The new version of XMetaL "features innovative ease-of-use and collaboration capabilities for content authors and provides developers with a rich XML development environment. XMetaL 3 is the first customizable XML editor with a rich development environment to support XML Schema, an essential standard developed by the W3C to define common languages for specific business applications. XML Schema enables companies to exchange information seamlessly with partners, customers, and suppliers. XMetaL 3 is a validating XML editor within an open and scriptable development environment that allows developers to use common Web development and programming skills to create integrated XML content applications. With an extensive COM and JAVA API comprising over 300 interfaces, support for DOM, CSS, XSL and standard scripting languages, XMetaL gives developers the power to control exactly how XML is displayed and entered by end users. New XMetaL features for end users include support for revision marking, preview in HTML and PDF, enhanced integration with Microsoft Office, workgroup document sharing, and enterprise document sharing." W3C XML Schema Support: "(1) XMetaL will load a W3C Schema and validate documents using features that are common to both W3C Schema and DTDs as well as important features unique to W3C Schema; (2) Supports the structural validation features in demand by industry XML standards organizations such as RIXML; (3) Supports locally-scoped element declarations, occurrence indicators, and substitution group elements, and more." [*Full context*]
- [December 11, 2001] "What's a Schema Anyway?" By Dave Peterson. In *XML Files: The XML Magazine* Issue 32 (December 2001). Edited by Dianne Kennedy. "... [DTD syntax is] a concise language, special purpose for DTDs. It was designed that way, because back in the '80s there were no authoring tools already existing; DTDs had to be read and written entirely by hand. The SGML designers considered writing DTDs using SGML 'tag' markup, but found it too hard to read. On the other hand, now there are tools for authoring XML (and SGML). And there is in the

XML culture a desire to do everything with tags when possible. So the Schema designers chose to describe a Schema using an XML document -- that is, using 'tag' syntax. This makes written Schemas (properly called Schema Documents) somewhat 'wordier' -- more difficult to read -- than DTDs, in the raw, but also makes it easier to find tools to help deal with them. The standard that describes schemas makes an explicit distinction between the abstract set of information (the Schema) and the written description thereof (one or more Schema Documents). Not only does this make it easier to understand, it also makes it easier to consider alternate description languages..."

- [December 07, 2001] Xerces-C++ XML Parser Version 1.6.0 Provides Full Support for W3C XML Schema Recommendation. A communiqué from Tinny Ng (XML Parsers Development, IBM Toronto Laboratory) announces the release of the Apache Xerces-C++ XML parser version 1.6.0, including full support for the W3C XML Schema Recommendation. Xerces-C++ is "a validating XML parser written in a portable subset of C++; Xerces-C++ makes it easy to give your application the ability to read and write XML data. A shared library is provided for parsing, generating, manipulating, and validating XML documents. The parser features: (1) Conformance to the XML Specification 1.0; (2) Tracking of latest DOM [Level 1.0], DOM [Level 2.0], SAX/SAX2, Namespace, and XML Schema specifications; (3) Source code, samples, and documentation; (4) Programmatic generation and validation of XML; (5) Pluggable catalogs, validators and encodings; (6) High performance; (7) Customizable error handling." The latest version of Xerces-C++ includes a port to FreeBSD and support for specifying a schema location through a method call. Source code and binaries are available for several platforms. [*Full context*]
- [November 29, 2001] Schema Toolkit. "XML-Schema Toolkit is an innovative XML parsing, data-binding and schema validation tool. XML-Schema Toolkit can help turn XML standards into implementations. Using the Schema Coder GUI application, an XML schema is transformed into C ++ code, enabling the rapid creation of XML schema-valid vocabularies. It also provides a framework to use these vocabularies, once they have been implemented. Version 0.12 contains SchemaCoder, which can convert XML schemas to C++ code. The web site now has better documentation, explaining the aims of the toolkit. The toolkit is currently free to use, but not to redistribute. It works for Windows 98, NT and upwards, and the code integrates with Microsoft's Visual C++ 6..." See also the description.

- [November 21, 2001] "GXS Releases XML Schema Plug-in. Saves Time Converting Between XML and Other Formats." In *eai Journal* (November 13, 2001). ['Mapping transactions from proprietary data formats, such as those for enterprise ERP or legacy systems, into XML is a time-consuming process. GXS XML Schema Plug-in saves time in this mapping process by allowing the use of XML schemas to help define the structure, content, and semantics of XML documents.'] "GE Global eXchange Services (GXS) has released XML Schema Plug-in, e-business software that will allow companies to more quickly integrate XML transactions into back office systems. The new software can help companies save time and reduce costs associated with exchanging XML documents with suppliers, customers, and other trading partners. The Plug-in works in conjunction with the GXS data transformation engine, allowing customers to import XML schemas into the mapping tool, and then validate XML messages against the schemas. Platforms supported by the Plug-in include Windows and Unix. Said Jim Rogers [GXS general manager of integration solutions]: 'Clients who are implementing their e-buy process using XML will find that this capability can save them a great deal of time and effort in integrating XML B2B transactions with their ERP or legacy business applications'..."
- [November 12, 2001] XSV (XML Schema Validator) Version 1.4. This version introduces a 'derivation-by-restriction experiment'. "The web interface, the self-installer, and the sources for XSV, our W3C XML Schema validator, have all been updated with a new version. Version 1.4 introduces a cheap-but-effective approximation to enforcement of the constraints on derivation by restriction for complex type definitions. This works by enforcing the subset invariant, and rejects any content model for a type definition derived by restriction which allows anything *not* allowed by the base type definition's content model. This is slightly weaker than the REC: *i.e.*, everything it rules out is ruled out by the REC, but a few things ruled out by the REC will not be caught. Note this is *not* what the REC envisages, but it will go a lot further towards enforcing interoperability between XSV and other processors which *do* enforce the REC as intended. Nothing is as good as full conformance, but I hope this step, which only took about three hours to implement, will be of use never-the-less..." From Henry S. Thompson 2001-11-12.
- [November 01, 2001] "Extending Schemas." Edited by Paul Kiel (HR-XML). Contributors: Members of the HR-XML Technical Steering

Committee. Working Draft 2001-09-11, Version 1.0. "HR-XML Consortium specifications are meant to model specific business practices. Recognizing that it cannot satisfy the needs of all implementers all the time, the need for a standard way to extend schemas becomes clear. This document is aimed to provide guidance regarding the extension of XML Schemas so that trading partners can exchange information in the real world as well as experiment with new data that could be incorporated into a future specification... Given that extension is a reality, how can we accommodate extensions without undermining the principle of open standards? This document is meant to provide guidance on the best practice for extending schemas. Its goal is to show: (1) Official endorsement of different methods for implementation; (2) Conventions for creating extensions to encourage consistency... This document focuses on XML Schema extension methods. Where possible, references to DTD equivalent issues are included. Addressing all possible extension methods for DTDs (*i.e.*, internal subsets) is not in scope... As discussed here, 'extending' is meant to 'add additional elements and attributes to an existing schema'. This is not to be confused with how Roger Costello uses it in the *XML Schemas: Best Practices* discussion; he refers to 'extending' meaning adding functionality to schema that does not currently exist... The Technical Steering Committee has approved two methods that enable extension of HR-XML schemas without undermining an open standards mission. The 'wrapper' and 'ANY' techniques are explained herein. Additionally, a 'Namespace' extension method was examined and rejected, details in Appendix C..." [Comment from Chuck Allen: "Like every other organization developing XML schemas, HR-XML is wrestling with 'standard' approaches to extending 'standards'. The editors welcome comments on the extension document; send email to Chuck Allen or Paul Kiel. See: (1) "HR-XML Consortium"; (2) Elists for lists.hr-xml.org. [source .DOC]

- [November 01, 2001] "[XML Schema] Enumeration Extension." By Paul Kiel (HR-XML). 2001-10-29. 5 pages. "HR-XML Consortium work groups are increasingly grappling with a common problem regarding the use of enumerations in schemas. The traditional use of enumerations consists of a fixed number of provided values, which are determined at the time of the schema design. The problem arises when a business process is modeled with a schema that includes enumerated lists that do not cover 100% of the foreseen cases. The main question arises: how can a work group standardize enumerated values

when less than 100% of the foreseeable values are known at design time? The objective of this text is to endorse a method for standardizing enumerated values without preventing extensions to cover unknown or trading partner specific values... The two most debated approaches to standardizing incomplete enumeration lists are a union of values with a string, essentially a convention, and a union of values with a string pattern, known as pattern extension... The Technical Steering Committee has determined that while the principle of separating data from metadata has significant merit, in this case, making the best use of the parser can be more important. Consequently, it endorses the Pattern Extension method of standardization of incomplete enumeration lists when most of the values are known. When only a few values are know, the Convention Method is acceptable as well as using an atomic data type such as a simple string..." Comment from Chuck Allen: "We regard this document on handling enumerations as a schema design technique with some use in certain circumstances. We're also interested in the notion of 'interfaces to taxonomies' -- being able to reference or (carry along) taxonomies that would provide data values. One of our use cases is posting job openings to job boards; each job board has a different classification system (these usually translate to drop-down lists in a web user interface). We definitely *will not* take on the task of try to develop standard skill and job taxonomies, but we want to be able to plug-in those in use by major job boards and by government agencies (US Dept of Labor and many other national labor boards have developed skill/job taxonomies). While our use case is specific to HR, the basic problem of referencing and using external taxonomies is not unique to our problem domain..." See: (1) "HR-XML Consortium"; (2) Elists for lists.hr-xml.org. [source .DOC]

- [October 30, 2001] New Implementation of the W3C/LTG Validator for XML Schema (XSV). A posting from Henry S. Thompson (HCRC Language Technology Group, University of Edinburgh) announces the availability of XSV Version 1.3. XSV (Validator for XML Schema) is an open source (GPLed) work-in-progress attempt at a conformant schema-aware processor, as defined by the W3C Recommendation for XML Schema. A significant change has been introduced in version 1.3, which "switches from using DTD to pre-validate schema documents to using schema-for-schemas. Hitherto, XSV has pre-validated all schema documents involved in a schema-validation episode with the DTD for schemas. This was both not quite right, in that it meant certain constraints were not enforced, because not expressed in the DTD, and messy, in

that if a schema document included an internal subset, it was tricky to preserve it. With the advent of version 1.3, all schema documents are now pre-validated with a pre-compiled version of the schema for schemas itself. Even if a DOCTYPE is present, XML 1.0 validation will not be performed, although processing will reflect attribute defaults and general entity bindings, as required by XML 1.0." The authors have done their best with regression testing to ensure that the new XSV release is working properly, but there is potential for introduction of new vulnerabilities and backwards incompatibilities; feedback from the public is solicited. Both the online web version and the standalone/download implementation of XSV have been upgraded to XSV version 1.3. The XSV Schema validator has been developed by Henry S. Thompson and Richard Tobin, with contributions (Web interface) by Dan Connolly. [*Full context*]

- [October 26, 2001] Apache XML Project Releases Xerces-C++ Parser Version 1.5.2 with Enhanced Schema Support. A posting from Tinny Ng (XML Parsers Development, IBM Toronto Laboratory) announces the release of the Xerces C++ 1.5.2 XML parser from the Apache XML Project. The Xerces C++ Parser Version 1.5.2 provides additional support for W3C XML Schema. Highlights of the new release include: (1) More Schema Subset support (see http://xml.apache.org/xerces-c/schema.html for details and restriction) (2) `XMLPlatformUtils::Initialize/Terminate()` pair of routines can now be called more than once within a process; (3) Progressive parse support in SAX2XMLReader; (4) Project files for BCB 5; (5) `runConfigure` script to accept multiple compiler and linker options; (6) more bug fixes, and performance improvement." The distribution includes source code as well as binaries for AIX, HP11, Linux, Solaris, and Windows. [*Full context*]
- [October 19, 2001] "Modeling XML Vocabularies with UML: Part III." By Dave Carlson. From XML.com. October 10, 2001. ['The third and final part of Dave Carlson's series on modeling XML vocabularies covers a specific profile of UML for use with XML Schema, and describes how UML can contribute to the analysis and design of XML applications.' See previously Part I and Part II.] "This article is the third installment in a series on using UML to model XML vocabularies. The examples are based on a simple purchase order schema included in the W3C XML Schema Primer, and we've followed an incremental development approach to define and refine this vocabulary model with UML class diagrams. The first objective of this third article is to complete the process of refining the PO model so that the

resulting schema is functionally equivalent to the one contained in the XSD Primer. The second objective is to broaden our perspective for understanding how UML can contribute to the analysis and design of XML applications... The following list summarizes several goals that guide our work. (1) Create a valid XML schema from any UML class structure model, as described in the first two parts of this series. (2) Refine the conceptual model to a design model specialized for XML schema by adding stereotypes and properties that are based on a customization profile for UML. (3) Support a bi-directional mapping between UML and XSD, including reverse engineering existing XML schemas into UML models. (4) Design and deploy XML vocabularies by assembling reusable modules. Integrate XML and non-XML information models in UML; to represent, for example, both XML schemas and relational database schemas in a larger system... Even this relatively narrow scope covers a broad terrain. The following introduction to a UML profile for XML adds a critical step toward all of these goals. These extensions to UML allow schema designers to satisfy specific architectural and deployment requirements, analogous to physical database design in a RDBMS. And these same extensions are necessary when reverse engineering existing schemas into UML because we must map arbitrary schema structures into an object-oriented model... One of the benefits gained by using UML as part of our XML development process is that it enables a thoughtful approach to modular, maintainable, reusable application components. In the first two parts of this series, the PurchaseOrder and Address elements were specified in two separate diagrams, implying reusable submodels. UML includes package and namespace structures for making these modules explicit and also specifying dependencies between them... A package, shown as a file folder in a diagram, defines a separate namespace for all model elements within it, including additional subpackages. These UML packages are a very natural counterpart to XML namespaces. A dashed line arrow between two packages indicates that one is dependent on the other. When used in a schema definition, each package produces a separate schema file. The implementation of dependencies varies among alternative schema languages. For DTDs they might become external entity references. For the W3C XML Schema, these package dependencies create either `<include>` or `<import>` elements, based on whether or not the target namespaces of related packages are equal. A dependency is shown from the PO package to the XSD_Datatypes package, but an import element

is not created because this datatype library is inherently available as part of the XML Schema language. This object-oriented approach to XML schema design facilitates modular reuse, just as one would do when using languages such as Java or C++..."

- [October 17, 2001] Sun Microsystems Releases Generalized Schema-Related Tools for Validation and Conversion. A posting from Kohsuke KAWAGUCHI (Sun Microsystems) announces the availability of an updated version of Sun's Multi-Schema XML Validator (MSV), along with three new schema-related tools. The new Sun XML Instance Generator "is a Java technology tool to generate various XML instances from several kinds of schemas; it supports DTD, RELAX Namespace, RELAX Core, TREX, and a subset of XML Schema Part 1. The RELAX NG Converter is a tool to convert schemas written in various schema languages to their equivalent in RELAX NG. The new Multi-Schema XML Validator Schematron add-on is a Java tool to validate XML documents against RELAX NG schemas annotated with Schematron schemas. By using this tool, you can embed Schematron constraints into RELAX NG schemas, making it easy to write many constraints that are difficult to achieve by RELAX NG alone." [*Full context*]
- [October 06, 2001] "Possible Extensions to RELAX NG. DSDL Use Cases." By Martin Bryan (The SGML Centre). BSI IST/41. Posted to the RELAX-NG mailing list. "In talking to James Clark earlier today about the relationship between RELAX NG and the proposed new ISO Document Structure Definition Language (DSDL) James asked if I could provide some use cases that would justify the initial set of requirements that the DSDL proposal contained. The attached document starts by listing the requirements identified as being essential for DSDL, and then provides a set of use case statements that seeks to justify each requirement. It also contains brief use cases for supporting three optional features of SGML that are not supported by XML, and not listed as being requirements for DSDL, but for which cases can be made within data streams being used by businesses..." See "Document Schema Definition Language (DSDL) Proposed as ISO New Work Item." (June 12, 2001).
- [October 02, 2001] Zvon XML Schema Reference. [Another cool tool] Prepared by Jiri Jirat and Miloslav Nic. The Schema reference is based on the W3C XML Schema Recommendations [updated from CR specifications]. The reference tool consists of two parts: (1) A Schema browser, based on the analysis of normative XML Schema; (2) A DTD browser, based on the analysis of non-

normative DTD. The reference features hyperlinked clickable indexes and schemas: clicking on 'Annotation Source' or 'Go to standard' leads you to the relevant part of the specification.

- [September 24, 2001] "XML Schema Quick Reference Cards." Prepared by Danny Vint. See: (1) XML Schema - Structures Quick Reference Card, and (2) XML Schema - Data Types Quick Reference Card. XML-DEV posting: "I've just uploaded 2 quick reference cards that I built for the XML Schema Data types and Structures specifications. These cards are available in PDF format. If you download and print them realize that they are setup for 8.5 x 14 paper. If when you print these files, just set the 'Fit to page' and Landscape mode to get a properly scaled copy of these documents. I'm also in the process of moving my 'XML Family EBNF Productions Help' to this new site as well as updating the content. This isn't completed; I'm currently showing the older version that I have previously published..."
- [September 21, 2001] "Modeling XML Vocabularies with UML: Part II." By Dave Carlson. From XML.com. September 19, 2001. "Mapping UML Models to XML Schema: This is where the rubber meets the road when using UML in the development of XML schemas. A primary goal guiding the specification of this mapping is to allow sufficient flexibility to encompass most schema design requirements, while retaining a smooth transition from the conceptual vocabulary model to its detailed design and generation. A related goal is to allow a valid XML schema to be automatically generated from any UML class diagram, even if the modeller has no familiarity with the XML schema syntax. Having this ability enables a rapid development process and supports reuse of the model vocabularies in several different deployment languages or environments because the core model is not overly specialized to XML... The default mapping rules described in this article can be used to generate a complete XML schema from any UML class diagram. This might be a pre-existing application model that now must be deployed within an XML web services architecture, or it might be a new XML vocabulary model intended as a B2B data interchange standard. In either case, the default schema provides a usable first iteration that can be immediately used in an initial application deployment, although it may require refinement to meet other architectural and design requirements. The first article in this series presented a process flow for schema design that emphasized the distinction between designing for data-oriented applications versus text-oriented applications. The default mapping rules are often sufficient for data-oriented applications. In fact,

these defaults are aligned with the OMG's XML Metadata Interchange (XMI) version 2.0 specification for using XML as a model interchange format. This approach is also well aligned with the OMG's new initiative for Model Driven Architecture (MDA). Text-oriented schemas, and any other schema that might be authored by humans and used as content for HTML portals, often must be refined to simplify the XML document structure. For example, many schema designers eliminate the wrapper elements corresponding to an association role name (but this also prevents use of the XSD <all> model group). This refinement and many others can be specified in a vocabulary model by setting a new default parameter for one UML package, which then applies to all of its contained classes..." See: (1) Part I of Carlson's article; (2) "Conceptual Modeling and Markup Languages."

- [September 13, 2001] Altova Releases Comprehensive Tool Suite for Advanced XML Application Development. A posting from Alexander Falk announces the final production release of the XML Spy 4.0 Suite, "a comprehensive product-line of easy-to-use software tools, facilitating all aspects of XML application development. The XML Spy 4.0 Suite consists of the XML Spy 4.0 Integrated Development Environment (IDE), the XML Spy 4.0 XSLT Designer, and the XML Spy 4.0 Document Editor, a comprehensive tool-set for all XML application development. The XML Spy 4.0 Integrated Development Environment is a solution for developing XML-based applications, making it easy to create and manage XML documents, stylesheets, and schemas. The XSLT Designer is an innovative new approach to automate writing of complex XSLT Stylesheets using an intuitive, drag-and-drop user interface. The XML Spy 4.0 Document Editor is available as a browser plug-in or a stand-alone application; it offers a word-processor style free-flow WYSISYG editor for XML documents, empowering non-technical users to create and edit XML documents." A 30-day evaluation version is available for download. [*Full context*]
- [September 07, 2001] W3C Presents a First Public Release of the XML Schema Test Collection. A posting from Henry S. Thompson announces a "first public release of the W3C XML Schema Test Collection, made possible by a substantial contribution of tests from Microsoft. Both positive and negative expected outcomes are tested with respect to a range of core XML Schema features. [The tests are presented] in a standard form which tabulates (without ratifying) the test materials, together with a brief description, and the outcomes for each one

expected by the contributor. The document also includes the first of what the W3C team hopes will be many outcome tabulations for a publically available XML Schema processor... the column labelled 'Expected' means the outcome expected by the contributor [not necessarily what's expected by the W3C WG]. For the test file(s) present which has/have extension `.xsd`, its/their conformance to the XML Schema REC's definition of valid XML representations of XML Schemas is what is at issue. When a test file with extension `.xml` is present as well, its schema-validity is at issue as well." Thompson reports that the W3C team already has in hand an additional contribution of tests from NIST; these will be added soon to augment the 100+ tests from Microsoft. The test materials are available for download from the W3C web site as a single package, distributed under the W3C Document License. [*Full context*] [cache]

- [September 04, 2001] DTDinst Tool Converts XML DTDs into XML Instance Format. A posting from James Clark announces the availability of a DTD converter 'DTDinst' which converts XML DTDs into XML instance format. "The XML instance can be in either a format specific to DTDinst or can be in RELAX NG format." DTDinst-specific output format is documented in RELAX NG non-XML syntax and in RELAX NG format. The key feature of DTDinst "is its handling of parameter entities: it is able to reliably turn parameter entity declarations and references into a variety of higher-level semantic constructs. It can do this even in the presence of arbitrarily deep nesting of parameter entity references within parameter entity declarations. At the same time, it accurately follows XML 1.0 rules on parameter entity expansion, so that any valid XML 1.0 DTD can be handled. If a parameter entity is used in a way that does not correspond to any of the higher-level semantics constructs supported by DTDinst, then references to that parameter entity will be expanded in the DTDinst output. DTDinst is available as a precompiled JAR file; the source is also available." Clark provides an XSLT stylesheet that "converts DTDinst format to RELAX NG; it has many more limitations than the converter builtin to DTDinst, but it may be useful as a basis for XSLT-based processing of DTDinst format." James writes: "Feedback is welcome, especially on any DTDs it doesn't handle well and on additional features that you would like to see..." [*Full context*]
- [August 24, 2001] "Semantic Data Modeling Using XML Schemas." By Murali Mani, Dongwon Lee, and Richard R. Muntz (Department of Computer Science, University of California, Los Angeles, CA). [To be published] in *Proceedings of the 20th*

International Conference on Conceptual Modeling (ER 2001), Yokohama, Japan, November, 2001. "Most research on XML has so far largely neglected the data modeling aspects of XML schemas. In this paper, we attempt to make a systematic approach to data modeling capabilities of XML schemas. We first formalize a core set of features among a dozen competing XML schema language proposals and introduce a new notion of XGrammar. The benefits of such formal description is that it is both concise and precise. We then compare the features of XGrammar with those of the Entity-Relationship (ER) model. We especially focus on three data mod- eling capabilities of XGrammar: (a) the ability to represent ordered binary relationships, (b) the ability to represent a set of semantically equivalent but structurally different types as 'one' type using the closure properties, and (c) the ability to represent recursive relationships... Ordered relationships exist commonly in practice such as the list of authors of a book. XML schemas, on the other hand, can specify such ordered relationships. Semantic data modeling using XML schemas has been studied in the recent past. ERX extends ER model so that one can represent astyle sheet and a collection of documents conforming to one DTD in ERX model. But order is represented in ERX model by an additional order attribute. Other related work include a mapping from XML schema to an extended UML, and a mapping from Object-Role Modeling (ORM) to XML schema . Our approach is different from these approaches: we focus on the new features provided by an XML Schema -- element-subelement relationships, new datatypes such as ID or IDREF(S), recursive type definitions, and the property that XGrammar is closed under union, and how they are useful to data modeling... The paper is organized as follows. In Section 2, we describe XGrammar that we propose as a formalization of XML schemas. In Section 3, we describe in detail the main features of XGrammar for data modeling. In Section 4, we show how toconvert an XGrammar to EER model, and vice versa. In Section 5, an application scenario using the proposed XGrammar and EER model is given. Finally, some concluding remarks are followed in Section 6. ... [Conclusions:] In this paper, we examined several new features provided by XML schemas for data description. In particular, we examined how ordered binary relationships `1:n` (through parent-child relationships and IDREFS attribute) as well as `n:m` (through IDREFS attribute) can be represented using an XML schema. We also examined the other features provided by XML grammars -- representing recursive relationships using recursive type definitions and union types.

The EER model, conceptualized in the logical design phase, can be mapped on to XGrammar (or its equivalent) and, in turn, mapped into other final data models, such as relational data model, or in some cases, the XML data model itself (*i.e.*, data might be stored as XML documents themselves). We believe that work presented in this paper forms a useful contribution to such scenarios." Also available in Postscript format. [cache PDF, Postscript]

- [August 23, 2001] "Understanding W3C Schema Complex Types." By Donald Smith. From XML.com. August 22, 2001. "Are W3C XML Schema complex types so difficult to understand that you shouldn't even bother trying? Kohsuke Kawaguchi thinks so; or so he claimed in his recent XML.com article, in which he offered assurances that you can write complex types without understanding them. My response to that assertion is to ask why would you want to write complex types without understanding them, especially when they are easily understandable? There are four things you need to know in order to understand complex types in W3C Schemas... One of the most important, but least emphasized, aspects of W3C schemas is the type hierarchy. The importance of the type hierarchy can hardly be overstated. Why? Because the syntax for expressing types in schemas follows precisely from the type hierarchy. Schema types form a hierarchy because they all derive, directly or indirectly, from the root type. The root type is `anyType`. (You can actually use `anyType` in an element declaration; it allows any content whatsoever.) The type hierarchy first branches into two groups: simple types and complex types. Here we encounter the first two of the four things you need to know in order to understand complex types: first, derivation is the basis of connection between types in the type hierarchy; and, second, the initial branching of the hierarchy is into simple and complex types. It's no wonder that people get confused about complex types. They generally don't realize that all complex types are divisible into two kinds: those with simple content and those with complex content. The reason why people don't generally realize this is because they normally learn the abbreviated syntax first. But, as we've seen, if you learn the full syntax and the logic behind it first, then the abbreviated syntax, and complex types in general, cease to be a befuddingly conundrum. If all of this is now as clear to you as it is to me, you don't have to trust anyone's assurances that you should use complex types without understanding them. You can now use and understand them..."
- [August 18, 2001] XSD documentation generator. Preliminary version. By Christopher R. Maden. ['...

I need a stylesheet that when applied will report the metadata for a `.xsd` stylesheet (similar to the one that Microsoft has posted to their website for .xdr reporting...'] Chris replies: "I have one that handles simple features of XSD (basically, as much as I needed for a specific project) and reports the documentation and structure of the schema. It's rather crude right now; if you want to tweak the layout or add features, please feel free. (I would appreciate any mods being shared.) The documentation generator is at http://crism.maden.org/consulting/pub/xsl/xsd2html.xsl. There's a correspondingly-featured DTD generator at http://crism.maden.org/consulting/pub/xsl/xsd2dtd.xsl... Neither has any documentation right now; more as it is..." [cache]

- [August 16, 2001] Wrox Press Publishes Major Reference Tool for XML Schemas. Wrox Press has published a full-length volume on XML Schemas in its 'Programmer to Programmer' Series. *Professional XML Schemas* has been authored by Kurt Cagle, Jon Duckett, Oliver Griffin, Stephen Mohr, Francis Norton, Nikola Ozu, Ian Stokes-Rees, Jeni Tennison, and Kevin Williams. *Professional XML Schemas* "exhaustively details the W3C XML Schema language, and teaches the new syntax in an intuitive and logical way. [It documents] how to declare elements and attributes, how to create complex content models, how to work with multiple namespaces, and how to use XML Schemas in real-world situations. A number of practical case studies illustrate the design and creation of schemas in the diverse worlds of relational databases, document management, and e-commerce applications." The book covers all major aspects of schema application, including: "(1) A complete guide to XML Schema Syntax; (2) Using XML Schema built-in types, and deriving new types; (3) Working with XML Schemas and XML Namespaces; (4) Creating identity and uniqueness constraints; (5) Good schema design, illustrated in a number of different areas; (6) Working with schemas and XSLT; (7) Writing XML Schemas for working with SOAP; (8) Integrating Schematron and XML Schemas." Reference tools in appendices include Schema Element and Attribute Reference, Schema Datatypes Reference, UML Reference, Tools and Parsers, and Bibliography and Further Reading. [*Full context*]
- [August 11, 2001] "Information Supply Chain: XML Schemas Get the Nod." By Solomon H. Simon. In *Intelligent Enterprise* (July 23, 2001). ['Now that XML Schemas have reached final recommendation status, they are more attractive than DTDs.'] "In spite of the kick-start that B2B e-commerce provided for XML, many companies

held back because of their perception that XML lacked standards. But now that XML Schemas have been given final recommendation status by the World Wide Web Consortium (W3C), that resistance can start to subside. This status is tantamount to making XML Schemas a metadata standard and a valid alternative to Document Type Definitions (DTDs). With the general acceptance and use of schemas, companies will be ready to kick their XML communications and data interchange efforts into high gear. XML Schemas greatly simplify the use of XML in business applications because they follow XML format, enable data reuse, are compatible with extensible stylesheet language transformations, and are simpler compared to DTDs... A schema is the XML construct used to represent the data elements, attributes, and their relationships as defined in the data model. By definition, a DTD and a schema are very similar. However, DTDs usually define simple, abstract text relationships, while schemas define more complex and concrete data and application relationships. A DTD doesn't use a hierarchical formation, while a schema uses a hierarchical structure to indicate relationships. The XML Schema standard uses the XML syntax exclusively, rather than borrowing from SGML, and it will augment, then later supplant, DTDs... Schemas give the developer richer control over the data type declarations than is possible in DTDs. Second, schemas allow greater reuse of metadata by permitting the developer to include more external schemas than allowable with DTDs. The main reason to use schemas is to improve compatibility and consistency within an XML document or application. In isolation, it doesn't matter significantly if an XML document uses a DTD or a schema. However, the moment that a developer or user wants to modify the document, share the document, or combine multiple documents, the differences become more apparent. Because schemas follow the XML format, it is easier to design tools, such as extensible stylesheet language transformation scripts, that will modify them. A real concern about XML documents is that developers will use different vocabularies, which will minimize interoperability. To leverage the capabilities of XML, developers must be able to bend the syntax rules of a specific document without breaking the vocabulary. Although there are still obstacles to overcome, such as vocabulary, the W3C's recommendation of XML Schemas is a major step toward better data interchange between companies and, eventually, more sophisticated, widely used B2B e-commerce."

- [August 01, 2001] Glosses on 'PSVI' (Post Schema Validation Infoset). XML-DEV posts. (1) "The Post-Schema Validation Infoset is an XML

infoset as modified by a schema processor. Validating a document against a DTD requires that the DTD be read first, and then the document evaluated with respect to the DTD, because that's how SGML works. But since a schema is just another XML document, there's no order requirement. You can parse an XML document as well-formed and build an infoset from that. Then you take your schema and evaluate the document's infoset with respect to it. You augment the infoset with type information, default attributes or values, and validity status (yes, no, not checked) - that's your PSVI." [Christopher R. Maden] (2) "Post Schema Validation Infoset. A goal of Schema was to leave the infoset unchanged after doing schema processing, unlike DTDs. So, there needed to be a place to stick all the aditional information a Schema provides an instance, hence the creation of the PSVI. A processor or application that doesn't care about the PSVI can still validate a document against a schema and not have to worry about it." [David E. Cleary]

- [August 01, 2001] Sun Microsystems Releases Java 'Multi-Schema XML Validator'. A posting from Kohsuke KAWAGUCHI (Sun Microsystems) announces the availability of a 'Sun Multi-Schema XML Validator.' The Sun Multi-Schema XML Validator (MSV) is "a Java technology tool to validate XML documents against XML schemata. MSV supports RELAX NG, RELAX Namespace, RELAX Core, TREX, XML DTDs, and a subset of W3C XML Schema Part 1. The validator can be used as a command-line tool (to validate XML documents against a schema or DTD) or as a library (to validate documents or to manipulate schemas from inside a Java application). The distribution includes binaries, sample source code, and detailed documentation." [*Full context*]
- [July 23, 2001] "Xerces, XML4J, and XML4C add XML Schema support. Summer 2001 updates to Apache and IBM parser." By Natalie Walker Whitlock (Writer/Owner, Casaflora Communications) . From IBM developerWorks. July 2001. ['New versions of the Apache XML Project's Xerces parsers released in June support the W3C XML Schema Recommendation. The new Xerces for Java supports essentially all of the XML Schema spec; Xerces for C++ implements a more limited subset of XML Schema, an incremental step toward complete support of the newly anointed specification that will in many cases take the place of DTDs in XML development. IBM also released updates to the alphaWorks parsers -- XML4C and XML4J -- that correspond to the Xerces parsers. A table outlines the XML Schema features supported in this release of the parsers.'] "The two popular Xerces

parsers from the Apache Software Foundation, Xerces Java (aka Xerces-J) and Xerces C++ (aka Xerces-C) made a great leap forward in June to support XML Schema. Xerces-J 1.4.1 boasts essentially complete support for the entire W3C XML Schema Recommendation. Xerces-C 1.5 supports a more limited subset of XML Schema. The alphaWorks parsers based on them, XML4J and XML4C, have also been updated with corresponding XML Schema support. The Xerces-C update is characterized as an important incremental step toward full W3C XML Schema support. In its announcement to its mailing list, the Apache XML Project promises to continue to update its open-source C++ parsers steadily, with the goal of implementing all of the features of the current XML Schema Recommendation before the end of the year... Other parsers, such as those from Oracle, XSV, XmlSpy, MSParser, and Extensibility, all claim some support for XML Schema. According to the companies' technical specifications, however, currently these parsers merely edit and validate XML schemas; they cannot read or interpret XML Schema instances. Additionally, at this writing the MSXML parser supports only Microsoft's version of the schema language, XML-data. According to my review of the online literature (Web sites, newsgroups, and mailing lists), the Xerces parsers (and their alphaWorks relatives) are the first to truly support advanced W3C XML Schema functionality..."

- [July 21, 2001] **Microsoft Releases MSXML Parser 4.0 Beta 2.** Microsoft has announced the release of a technology preview 'Beta 2' version of MSXML Parser 4.0, offering "a faster SAX and XSLT, complete XSD," and other enhancements. "The July 2001 release of the Microsoft XML Parser (MSXML) 4.0 Technology Preview is a preliminary release of MSXML 4.0. This technology preview has a number of improvements compared to the April release: (1) XSD validation with SAX; (2) XSD validation with DOM, using the `schemaLocation` attribute; (3) Schema Object Model (SOM) to access schema information in DOM and SAX; (4) Substantially faster XSLT engine -- tests show about x4, and for some scenarios x8, acceleration, except the known serious performance bug for `xsl:keys`; (5) New and substantially faster SAX parser, which is also available in DOM with the `NewParser` property... [See discussion.]
- [July 18, 2001] "XML for Data: Styling With Schemas. Using XML Schema Archetypes and XSLT Style Sheets to Simplify Your Code." By Kevin Williams (Chief XML architect, Equient - a division of Veridian). From IBM developerWorks. July 2001. ['This column by developer and author Kevin Williams demonstrates how to use XML

Schema archetyping (and style sheets) to control styling of data for various presentation modes. Ten code samples in XML, XML Schema, and XSLT show how the techniques work to reduce code bulk and simplify maintenance.'] "In my previous column, I described how simple and complex archetypes may be used to simplify and streamline your XML schema designs. This column takes a look at one practical application of XML Schema archetypes: using style sheets to provide a consistent rendering of archetypes in the presentation layer... What are 'archetypes'? Archetypes are common definitions that can be shared across different elements in your XML schemas. In early versions of the XML Schema specification, archetypes had their own declarations; in the released version, however, archetypes are implemented using the simpleType and complexType elements... This column outlines the way you can use archetypes to streamline your coding experience. This discussion really only scratches the surface. In a large system that must support many presentation targets (HTML, wireless, other machine consumers) and many different source-document types (for bandwidth reduction or security reasons), using archetypes properly makes it very easy to keep your style sheet output consistent and correct." See previously: "XML for Data: Using XML Schema Archetypes. Adding Archetypal Forms to Your XML Schemas."

- [July 17, 2001] "RELAX NG: Unification of RELAX Core and TREX." By MURATA Makoto (International University of Japan, currently visiting IBM Tokyo Research Lab.) Paper [to be] presented at Extreme Markup Languages 2001, August 12-17, 2001, Montréal, Canada. "RELAX Core and TREX are schema languages for XML. RELAX Core was designed in Japan and has recently been approved as an ISO Technical Report (ISO TR 22250-1); TREX was designed by James Clark. RELAX Core and TREX are similar: they are based on tree automata and do not change information sets. On the other hand, there are some significant differences: attributes, unordered content models, namespaces, wild cards, the syntax, and the underlying implementation techniques. At OASIS, it was decided to unify these two languages and the new language is called RELAX NG. This talk shows how differences between RELAX Core and TREX are resolved in RELAX NG." See: "RELAX NG."
- [July 16, 2001] XML Schema for ISBN. By Roger L. Costello and Roger Sperberg. Description: "Roger Sperberg and I have collaborated to create an ISBN simpleType definition. It defines the legal ISBN values for every country in the world... The ISBN schema was not able to check all the

constraints on an ISBN number. One of the constraints on ISBNs is that the last digit must match a certain sum of the previous digits modulo 11. (This is all documented in the ISBN schema.) Clearly, this constraint is not expressable with XML Schemas. Consequently we needed to supplement the ISBN schema simpleType definition with something else. We choose to express the additional constraints using XSLT... this allows anyone in publishing who is working with XML Schema to incorporate ISBN validation in their applications without having to create it from scratch. The agencies for the 126 group codes (which mostly represent countries, but also geographical regions and language groupings) do not all follow the recommendations of the international ISBN agency, so pending further research, the validation for some groups is not full. (It is complete, however, for the English-speaking ISBNs)..." -- Postings from Roger L. Costello and Roger Sperberg. [cache, with examples, XSLT script]

- [July 16, 2001] "Taxonomy of XML Schema Languages Using Formal Language Theory." By MURATA Makoto (International University of Japan, currently visiting IBM Tokyo Research Lab.) Dongwon Lee, and Murali Mani (University of California at Los Angeles/Computer Science Department). Paper to be presented at Extreme Markup Languages 2001, August 12-17, 2001, Montréal, Canada. PDF (print) version: 25 pages. "Most people are familiar with regular expressions, which define sets of strings of characters (regular languages). An extension of the idea of regular languages (as sets of strings) yields the idea of regular tree languages, which are sets of trees. From the ideas of regular tree languages, a mathematical framework for the description and comparison of XML Schema languages can be constructed. In this framework, four subclasses of regular tree languages and distinguished: local tree languages, single-type tree languages, restrained-competition tree languages, and regular tree languages. With these subclasses one can classify a few XML schema proposals and type systems: DTDs, the W3C XML Schema language, DSD, XDuce, RELAX, and TREX. Different grammar subclasses have different properties under the operations of XML document validation and type assignment..." Also available in HTML format. See the XPress project publications listing for related papers. [cache]
- [July 16, 2001] "A Standards-Based Framework for Comparing XML Schema Implementations." By Henry S. Thompson (HCRC Language Technology Group and World Wide Web Consortium) and Richard Tobin (HCRC Language

Technology Group). Paper to be presented at Extreme Markup Languages 2001, August 12-17, 2001, Montréal, Canada. "XML Schema processing may be described as a mapping from an input information set to an output post-schema-validation information set (PSVI). Information sets are not defined as concrete data structures, APIs, or data streams; they are abstractions. But if they realize the PSVI in different ways, how can one compare two implementations of XML Schema to check their consistency with each other? One simple standards-based approach is to reflect the PSVI as an XML document. One can then use standard tools to compare the output of the two processors. XSLT stylesheets can be used to display the reflected PSVI and to highlight differences between results produced by different processors or by the same processor from different inputs..."

- [July 16, 2001] ZedX XML Studio, under development 2001-07-13 by Zheng Min. See the announcement. The editing tool supports "structure based and rule based XML Schema (also supporting other formats) editor that allows a user to define schemas without knowing the syntax of the schemas... [Supports] content-sensitive element/attribute listing and auto-completion. When inserting an element, a pull-down list shows only the valid elements at the point of insertion (not like some editors that show all the elements available in the document). Once an element is selected, the editor automatically creates the element and any sub-elements required by its DTD/Schema. This feature is available for both wellformed and validated document editing..."
- [July 12, 2001] IBM's XML Parser for Java (XML4J) Supports W3C XML Schema Recommendation. IBM alphaWorks has released an updated version of the XML Parser for Java (XML4J) which supports the W3C XML Schema specification and includes other enhancements. XML4J version 3.2.0 is distributed as source code and as a binary; it is covered by the standard Apache 1.1 license. XML4J now incorporates the following: "(1) W3C XML Schema Recommendation 1.0 support; (2) SAX 1.0 and SAX 2.0 support; (3) Support for DOM Level 1, DOM Level 2, and for some features of DOM Level 3 Core Working Draft; (4) JAXP 1.1 support." The IBM XML applications development team has also released an improved version of the 'XML Schema Quality Checker' tool. Version 1.85 of the XML Schema Quality Checker fixes 15 bugs present in the previous version, and improves usability under Solaris 2.7 and Windows 98. IBM's XML Schema Quality Checker "is a program which takes as input an XML Schema

written in the W3C XML schema language and diagnoses improper uses of the schema language; where the appropriate action to correct the schema is not obvious, the diagnostic message may include a suggestion about how to make the fix." [*Full context*]

- [July 09, 2001] "From DTDs to XML Schemas. [EXPLORING XML.]" By Michael Classen. From Webreference.com. July 2001. ['Describing XML documents using XML Schemas offers a number of advantages over DTDs. In today's Tools Treasure Hunt, XML explorer Michael Classen introduces you to a utility that will help you convert your existing DTDs to XML Schemas.'] "The XML Schema standard was conceived to improve on DTD limitations and create a method to specify XML documents in XML, including standard pre-defined and user-specific data types. Defining an element specifies its name and content model, meaning attributes and nested elements. In XML Schemas, the content model of elements is defined by their type. An XML document adhering to a schema can then only have elements that match the defined types. One distinguishes simple and complex types. A number of simple types are predefined in the specification, such as string, integer and decimal. A simple type cannot contain elements or attributes in its value, whereas complex types can specify nesting of elements and associations of attributes with an element. User-defined elements can be formed from the predefined ones using the object-oriented concepts of aggregation and inheritance. Aggregation groups a set of existing elements into a new one. Inheritance extends an already defined element so that it could stand in for the original. The DTD to XML Schema Conversion Tool takes a DTD and translates it into its equivalent XML schema definition..."
- [July 06, 2001] New Release of XML Schema Validator (XSV). A posting from Henry S. Thompson (HCRC Language Technology Group, University of Edinburgh) announces an update of the W3C/LTG XML Schema Validator tool. The Validator for XML Schema REC (20010502) version is "an open source work-in-progress attempt at a conformant schema-aware processor, as defined by *XML Schema Part 1: Structures*, May 2, 2001 (REC) version. XSV has been developed by Henry S. Thompson and Richard Tobin of at the Language Technology Group of the Human Communication Research Centre in the Division of Informatics at the University of Edinburgh." The new release [XSV 1.197/1.99 of 2001/07/06 10:02:16] is available interactively online from the W3C web site. The '2001/07/06' release provides bug fixes and better handling of attribute defaults. Source code and

Win32 binaries have also been updated. The online version of the tool provides two HTML forms: (1) one for checking a schema which is accessible via the Web, and/or schema-validating an instance with a schema of your own, (2) another for file upload if you are behind a firewall or have a schema to check which is not accessible via the Web. Four styles of output may be selected (verbose/concise; styled for different generations of HTML browsers). [*Full context*]

- [July 05, 2001] Updated IBM XML for C++ parser (XML4C). XML4C version 3.5.0 [released 06/27/2001] is based on the Apache Xerces XML C++ Parser v1.5.0. It includes experimental support for a subset of the W3C Schema language, bug fixes and performance improvements. The download now includes the iSeries (AS400) binaries... [it is] a validating XML parser written in a portable subset of C++. XML4C integrates the Xerces-C parser with IBM's International Components for Unicode (ICU) and extends the number of encodings supported to over 150. It consists of three shared libraries (2 code and 1 data) which provide classes for parsing, generating, manipulating, and validating XML documents. Source code, samples and API documentation are provided with the parser. Version 3.5.0 update contains: (1) Support of SAX 1.0 and SAX 2.0 specifications (2) Support of DOM 1.0 and DOM 2.0 specifiactions (3) Experimental support of a subset of the W3C Schema language (4) Support for ICU 1.8.1 (5) Bug fixes and performance improvemenents (6) Documentation in PDF format (7) Experimental IDOM - a new design of the C++ DOM API. (8) Lexical Handler of SAX2-ext (9) DOM implementation optimization."
- [July 05, 2001] "Schema to Java Compiler." [Pre-Alpha, 2001-07-05.] Overview: "Java code, outlined in the packages allows the processing of XML documents described by specific schema. Usually, the incoming XML document gets unmarshalled into the instances of the classes generated from the schema, processed, and, finally, gets marshalled back into XML form..." See the announcement: "Creative Science Systems Announces Release of Schema to Java Compiler." - Schema2Java Compiler Tool is the first publicly available tool to generate run time Java code from arbitrary XML Schema that supports the implementation guidelines of the World Wide Web Consortium (W3C) XML Schema Technical Recommendation. W3C is the world-renowned standards body for developing interoperable Web Technologies..."
- [June 29, 2001] "XML for Data: Using XML Schema Archetypes. Adding Archetypal Forms to Your XML Schemas." By Kevin Williams (Chief

XML architect, Equient - a division of Veridian). From IBM developerWorks. June 2001. ['In the first installment of his new column, Kevin Williams describes the benefits of using archetypes in XML Schema designs for data and provides some concrete examples. He discusses both simple and complex types, and some advantages of using each. Code samples in XML Schema are provided.'] "In my turn on the Soapbox, I mentioned in passing how archetypes can be used in XML Schema designs for data to significantly minimize the coding and maintenance effort required for a project, and to reduce the likelihood of cut-and-paste errors. In this column, I'm going to give you some examples of the use of archetypes in XML schemas for data, and show just where the benefits lie. What are archetypes? Archetypes are common definitions that can be shared across different elements in your XML schemas. In earlier versions of the XML Schema specification, archetypes had their own declarations; in the released version, however, 'archetypes' are implemented using the simpleType and complexType elements. Let's take a look at some examples of each. Simple archetypes are created by extending the built-in datatypes provided by XML Schema. The allowable values for the type may be constrained by so-called facets, a fancy term for the different parameters that may be set for each built-in datatype. It's also possible to create a simple type by defining a union of two other datatypes or by creating a list of values that correspond to some other datatype. For our purposes, however, the restrictive declaration of simple types is the most interesting. Let's take a look at some examples... This installment has taken a look at the use of archetypes in the design of XML schemas. You've seen that judicious use of archetypes, together with smart naming conventions, can make schemas shorter and easier to maintain. There's an additional benefit to using archetypes -- a little trick to ensure consistent styling of your information..." Note the reference to the author's book *Professional XML Schemas* [ISBN: 1861005474], from Wrox Press; released now/ soon.

- [June 29, 2001] *Professional XML Schemas*. By [Wrox Team] Kurt Cagle, Jon Duckett, Oliver Griffin, Stephen Mohr, Francis Norton, Nik Ozu, Ian Stokes-Rees, Kevin Williams. Wrox Press. ISBN: 1861005474. 'July 2001'. [Provisional] Book Description from the publisher: "XML Schemas will take over from DTDs as the primary method of defining XML data. Some of the most powerful reasons for using XML Schemas are their ability to: Validate much more powerfully with extended constraint mechanisms The ability to create your own datatypes Dynamically bind instance

documents to schemas at run time Be used with existing xml tools as they are written in XML syntax Support namespaces Merge schemas into one Professional XML Schemas demystifies the complex W3C specification, showing how to create XML Schemas using the new syntax, and how to create schemas for documents, data transfer/storage, and object state. Data Modelling is strongly linked to learning about schemas. This book will discuss strategies for creating your own markup languages, and look at different models that authors should consider. The book will also introduce tools, practical examples of schemas developed for real world uses, and how they are used in the real world. This book is for all professional XML programmers who need to use XML Schemas to define data and need a practical guide to this new standard..."

- [June 22, 2001] IBM alphaWorks 'Regex for Java. "Updated 06/22/2001 to conform to the W3C Recommendation of XML Schema Datatypes. Also, new option to report non-matching position; some bug fixes. Regex for Java is a powerful, high-performance regular expression library for Java. You can search for a string matching to a regular expression pattern in your application with Regex for Java. Regex for Java supports almost all features of Perl5's regular expression. It also supports the syntax of XML Schema's regular expression." See *XML Schema Part 2: Datatypes*, W3C Recommendation 02-May-2001, Appendix F (Regular Expressions).
- [June 22, 2001] Regular Expression Generator for ranges. From Roger L. Costello, XML-DEV post. "I have created a simple tool which, given a range, will generate the corresponding regular expression. Example: for the range: 451 - 789, here is the regular expression which is generated: `45[1-9]|4[6-9][0-9]|[5-6][0-9][0-9]|7[0-7][0-9]|78[0-9]`." download.
- [June 22, 2001] "DTD to XML Schema Translator." dtd2xs version 1.0. [Use the 'dtd2xs' tool to] "translate a Document Type Definition (DTD) into a XML Schema (REC-xmlschema-1-20010502). The translator can map meaningful DTD entities onto XML Schema constructs (simpleType, attributeGroup, group), *i.e.* the XML document model is not anonymized. In addition, the translator can map DTD comments onto XML Schema documentation nodes in various ways. By default, DTD comments are ignored; with a flag, DTD comments may be preserved with default parameters. Freely available for download as Java class and as standalone Java application." [Posting from Joerg Rieger 2001-06-19 to 'xmlschema-dev@w3.org'; cache]
- [June 22, 2001] "Soapbox: Why XML Schema

beats DTDs hands-down for data. A look at some data features of XML Schema" By Kevin Williams (Chief XML Architect, Equient - a division of Veridian). From IBM developerWorks. June 2001. ['In his turn on the Soapbox, info-management developer and author Kevin Williams tells why he's sold on XML Schema for the structural definition of XML documents for data. He looks at four features of XML Schema that are particularly suited to data representation, and he shows some examples of each. Code samples include XSD schemas and schema fragments.'] "As you're no doubt aware, the W3C recently promoted the XML Schema specification to Recommendation status, making that spec the XML structural definition language of choice. While most people find the specifications a little hard to read, the jargon conceals a very strong set of features, especially for those of us who are designing XML structures for data. I'd like to take a look at a few of those features. Strong typing is probably the biggest advantage XML Schema has over DTDs, and it is the aspect of XML Schema you've heard the most about. In a DTD, you don't have a whole lot of choices for constraining the allowable content of your elements and attributes... [Conclusion:] I've taken a brief look at some aspects of XML Schema that make schemas much better than DTDs for the definition of XML structures for data. While DTDs are likely to be around for a while yet (there are plenty of legacy documents that still rely on them for their structural definition), support for XML Schema is quickly being implemented for all the major XML software offerings. In the following months, I'll take a look at some of the ideas I've laid out here in greater depth in my forthcoming column." Article also in PDF format.

- [June 18, 2001] Xerces-C++ Parser Provides Support for W3C XML Schema Recommendation. A posting from Tinny Ng (IBM Toronto Laboratory) announces the release of Xerces-C 1.5.0 with partial support for the W3C XML Schema Recommendation. The developers intend to update this package until it implements all the functionality of the current XML Schema Recommendation. Apache Xerces-C is a "validating XML parser written in a portable subset of C++. Xerces-C makes it easy to give your application the ability to read and write XML data. A shared library is provided for parsing, generating, manipulating, and validating XML documents. Xerces-C is faithful to the XML 1.0 recommendation and associated standards (DOM 1.0, DOM 2.0. SAX 1.0, SAX 2.0, Namespaces). The parser provides high performance, modularity, and scalability. Source code, samples and API documentation are provided with the parser. For portability, care has been taken to

make minimal use of templates, no RTTI, no C++ namespaces and minimal use of `#ifdefs`. In addition to the implementation of XML Schema subset, Xerces-C 1.5.0 offers: (1) Mac OS X command line configuration and build support; (2) Enabled libWWW NetAccessor support under UNIX; (3) Enabled COMPAQ Tru64 UNIX machines to build xerces-c with gcc; (4) Updated support for SCO UnixWare 7 [gcc]; (5) Experimental IDOM; (6) Support for ICU 1.8; (6) Documentation in PDF format; (7) Bug fixes and performance improvement." Xerces-C 1.5.0 source code and binaries are available for AIX, HP11, Linux, Solaris, Windows. [*Full context*]

- [June 14, 2001] Altova's XML Spy 4.0 Beta Supports W3C XML Schema Recommendation. Altova has announced a limited beta testing phase for the XML Spy 4.0 product line, including the XML Spy 4.0 Integrated Development Environment (IDE) and the XML Spy 4.0 Document Framework, released to customers and invited industry experts. The XML Spy 4.0 Integrated Development Environment (IDE) "builds on the success of the award-winning XML Spy 3.5 product in the developer market and adds expanded ODBC database access functionality, enhanced user interface customization, as well as support for the final XML Schema Recommendation for both graphical XML Schema editing and validation of XML instance documents based upon XML Schema. The XML Spy 4.0 Document Framework is based on a combination of XML Schema and XSLT Stylesheets. This provides the customer with a highly user-friendly interface -- very much like a typical word processor -- that allows for true XML content editing and creation. The framework consists of two applications: (1) The XML Spy 4.0 Document Editor supports free-flow WYSIWYG text editing, form-based data input, graphical elements, presentation and editing of arbitrary repeating XML elements as tables, real-time validation, and consistency checking using XML Schema and is deployed on the end-users desk. (2) The XML Spy 4.0 Document Administrator application includes a graphical XSLT Generator that enables the customization of the document editor by defining an XSLT Stylesheet and additional editing-specific options based upon the underlying DTD or XML Schema for use during the content creation or editing process." [*Full context*]
- [June 09, 2001] "[W3C] XML Schema Tutorial." By Roger L. Costello (of xFront.com XML Technologies). The main tutorial is a PPT slide set with some 276 slides. The slides reference 36 worked examples and 14 lab exercises. From the June 9, 2001 update note: "The tutorial is now updated to the Recommendation specification (*i.*

e., the latest W3C specification). It includes a complete set of labs with answers. All examples and lab answers are complete and have been validated using Henry Thompson's schema validator, **xsv** [self-installing Win32 .exe], which is bundled in with the tutorial (thanks Henry!). It also includes a Javascript program, written by Martin Gudgin, that enables you to use MSXML4.0 (thanks Martin!)... I have provided a number of DOS batch files (*i.e.*, validate.bat, run-examples.bat, run-lab-answers.bat) to make it easy for you to schema validate your XML files. I am continually adding new material to this tutorial. Please check back periodically for updates..."

- [June 09, 2001] "New Breeze XML Studio Release 2.5 Available. Beta Release Adds W3C XML Schema Support to Leading Data Binding Solution." - "The Breeze Factor, a company focused on providing solutions that simplify e-business using XML, today announced the latest release of its XML to Java data binding product: Breeze XML Studio Release 2.5. The release adds direct support for XML Schema, which was recently issued as a standard recommendation by the World Wide Web Consortium. Breeze XML Studio 2.5 is a substantial upgrade over previous releases and now can import structure in four formats: XML Schema (XSD), XML DTD, XML document (schema by example), and relational DBMS structure via JDBC/ODBC. With Breeze XML Studio 2.5, developers can take an XML schema and convert it directly to a set of Java classes that model the schema and encapsulate the parsing and validating of XML files conformant with that schema... While Breeze XML Studio can be used to design simple structures in its internal IDE, many customers will utilize a tool such as Tibco Extensibility's XML Authority for design and editing of complex structures. 'XSD is ideally suited to expressing business semantics. With its early support of XSD, The Breeze Factor is enabling users to bring that power to the application layer,' Lee Buck, VP, chief scientist XML Technologies, Tibco Extensibility. In addition to XML Schema support, the newest release includes improvements to the generated code when the schema has complex content models. 'It is one thing to parse an XML structure and understand the relationships between structure elements,' said Gregory Messner, CTO at The Breeze Factor. 'It is quite another to generate clean, readable and programmer-friendly component interfaces which make instances of that structure easy to use. The 2.5 release includes enhanced interfaces which make working with complex content very straightforward.' Breeze is licensed on a developer seat and OEM deployment basis. Developer seat licenses are

sold for $495 with discounts for volume purchases. The Breeze Factor simplifies e-business using XML. The company accelerates e-business efforts by generating frameworks for XML and XML-based protocols making it easier for developers to work with and extend the newest language of the Internet. The company's products include Breeze XML Studio, a visual development environment that binds XML data to Java classes and provides an alternative to the DOM for working with XML for application programming. The Breeze Factor has offices in Encinitas, Calif. and Park City, Utah."

- [June 09, 2001] TIBCO Software Releases 'XML Validate' with Support for W3C XML Schema Recommendation. An announcement from TIBCO Software Inc. describes the release of a new streaming XML validator with full support for W3C XML Schema. Details: Tibco has "announced the commercial release of XML Validate, a member of the TIBCO Extensibility product family. XML Validate is an enterprise-grade solution for validating streaming XML documents or messages against an XML Schema or DTD. The Simple API for XML (SAX)-based implementation for run-time validation provides organizations with the core component in developing high bandwidth, XML-based processing. This release of XML Validate is also the first commercially available validator to fully support the World Wide Web Consortium (W3C) XML Schema Recommendation. The XML Schema Recommendation was released [2001-05-02] by the W3C... this XML Schema validation support will facilitate the creation of XML driven ecosystems based on open-standards. Additionally, XML Validate supports the validation of DTDs to allow connectivity with organizations not currently using XML Schema. XML Validate is a core building block for creating an e-commerce processing engine for XML documents and messages. As organizations conduct e-commerce with a growing and global audience, the processing capabilities of XML Validate can scale to the demand. XML Validate has the potential of handling millions of transactions per day per server. XML Validate can easily be inserted into an existing XML parsing scenario, enabling validation to occur the instant it is received by the parser. Because SAX is an event-based API, XML Validate is the ideal solution in a streaming run-time environment, creating an enterprise-grade XML processing engine." [*Full context*]
- [June 05, 2001] "Translating XML Schema." By Timothy Dyck. In *eWEEK* (May 28, 2001). "Earlier this month at the Tenth International World Wide Web Conference in Hong Kong, XML took its biggest step forward since the document format was first standardized in February 1998. At the

conference, the World Wide Web Consortium released XML Schema as a W3C Recommendation, finalizing efforts that started in 1998 to define a standard way of describing Extensible Markup Language document structures and adding data types to XML data fields. Now that it is finally out, the long-delayed XML Schema standard will catalyze the next big step in XML -- allowing cross-organizational XML document exchange and verification. Just as discovery of the Rosetta stone in 1799 provided a way to fix the meaning of Egyptian hieroglyphs so they could be understood across the gulf of two millennia, XML Schema provides a way for organizations to fix the meaning of XML documents so they can be understood across the gulf of organizational boundaries and otherwise incompatible IT architectures. As a result, XML Schema will be a cornerstone in the new e-commerce architecture that we are collectively building and will be a vital component for making business exchanges and other loose associations of trading partners possible. The arrival of XML Schema, more than three years after XML itself, has left many chafing at the bit (and others, such as Microsoft Corp., running off in their own direction implementing and shipping products based on prestandard efforts), and the market is now more than ready for this standard to take hold. However, XML Schema's long development cycle gave vendors time to understand the specification and start writing compliant software, and we are now seeing the rapid release of XML Schema-compliant (or soon-to-be-compliant) authoring tools and servers... That long, committee-driven development cycle also resulted in a specification that has a bit of everything in it, and fully compliant XML Schema parsers will have to be complex pieces of software to support all the options the specification allows. Fortunately, XML Schema documents have to reference only the functionality they need, and the more complex options in XML Schema, such as null elements and explicit types, may just fade away through disuse. The W3C recently published a recommendation on how to group Extensible HTML, the consortium's replacement for HTML, into well- defined subgroups so XHTML browsers (such as those in cellular phones) can clearly define which parts of the language they support and which they don't. Something similar is a possibility for XML Schema if the full specification proves too difficult to implement for some vendors (although large players such as IBM, Microsoft and Oracle Corp. are moving ahead full speed with plans to support the full specification as published). Over the next few years, eWeek Labs predicts XML Schema will become integral to the way that many companies exchange information..."

- [June 05, 2001] "[W3C XML Schema] Speedy Adoption Expected." By Jim Rapoza. In *eWEEK* (May 28, 2001). "When XML was introduced, although there were early adopters, it still took about a year before Extensible Markup Language began to be regularly used in enterprise- level applications and deployments. Now that XML Schema is a standard, the waiting period for its adoption should be much shorter. Part of this can be attributed to how long businesses have been waiting for this schema. Many have been working on tools and compatibility issues while the standard was under development. However, it is also due in part to the complexity of the schema. Whereas the initial XML standard could be easily built and managed by anyone with an editor, many vendors plan to provide new tools to help shield users from the size and complexity of XSD (XML Schema Definition). Given the importance of XML Schema for handling data-driven communications among businesses, eWeek Labs recommends that developers begin evaluating tools that will help them move to XSD. In addition, companies should find out what their enterprise software vendors' plans are for supporting and integrating with XML Schema. As is true of most standards, many of the initial sets of XML Schema tools are essentially validators that help developers stay within the standard. Several are from individual World Wide Web Consortium members and universities, but some are also available from vendors such as IBM, and Java-based validators are available from Sun Microsystems Inc... Another important set of tools for businesses moving to XML Schema are conversion tools, which will help develop-ers convert content to the new standard. Probably the most important will be tools for converting standard XML DTDs (Document Type Definitions) to XSD, although some of those currently available have not been updated to the final standard. There are also tools for converting files from other schema languages, including a tool from Microsoft Corp. for converting files from XML Data Reduced to XSD...Microsoft recently released betas of MSXML and SQLXML that support the schema and has said that most of its products will support XSD in their next versions. Sun has released a new XML data types library that supports the final XML Schema standard, and Tibco Software Inc. includes tools for validating documents using XSD...
- [June 05, 2001] "Using Schema and Serialization to Leverage Business Logic." By Eric Schmidt. From Microsoft MSDN Online. 'Extreme XML' Column. May 17, 2001. ['New columnist Eric Schmidt addresses how you can use schemas and serialization technology to leverage XML in

your applications and services.'] "In this issue of Extreme XML, we are going to examine the importance of schema usage and the use of serialization technology to leverage XML in your applications and services. The majority of development tasks today revolve around developers taking existing infrastructure (business components, databases, queues, and so on) and morphing them into the next version of their product... The surge of XML usage over the past several years has not led to a complimentary increase in defined data models for XML documents. For this section, I am referring to a data model for XML to be the structure, content, and semantics for XML documents. The one main reason for this slow growth in XML data models is the lack of, until now, a robust XML schema standard. Document Type Definitions (DTDs) have out grown their usefulness in the enterprise space because of their focus on XML from a document perspective and not viewing XML document instances from a data and type perspective. Typed data items like addresses, line items, employees, orders, and so on have complex models and are the basis for most applications. Applications look at data from strongly typed perspective. For example, a Line Item is an inherited member of an order and contains typed information like product price, which is of type currency. The majority of this type of modeling cannot be accomplished with DTDs. Due to the simple structuring and typing mechanisms in DTDs, numerous XML validation, structuring, and typing systems have been created, including Document Content Description (DCD), SOX, Schematron, RELAX and XML-Data Reduced (XDR). The later, XDR, has gained much momentum in the Windows and B2B based communities due to its usage in products like SQL Server, BizTalk Server, and MSXML. In addition, most independent software vendors (ISVs) and B2B integrators support XDR because of its data typing support, namespace support, and its XML-based language. However, XDR's usefulness stills falls short of providing a truly extensible modeling and typing system for complex data structures. This was a known issue at the time of XDR's creation. Building on the lessons learned from previous schema implementations, the W3C XML Schema working group set out to create a specification (XML Schema) for defining the structure, content, and semantics of XML documents. Ultimately, this specification should provide an extensible environment so that it could be applied to any type of business or processing logic. During the development of this article, I was pleased to see that the W3C released XML Schema as a recommendation. This is a tremendous step in solidifying and stabilizing XML-

based implementations that need to employ schema services. Next, we're going to look at the importance and power behind XML Schema... I have distilled five core items you need to know about XML Schema so you can get up and running: (1) XML Schema is represented in XML 1.0 syntax; this makes parsing XML Schema available to any XML 1.0-compliant parser, and thus can be used within a higher-level API like the DOM. (2) Data typing of simple content: XML Schema provides a specification for primitive data types (string, float, double, and so on) found in most common programming languages. (3) Typing of complex content: XML Schema provides the ability to define content models as types. (4) Distinction between the type definition and instance of that type: unlike XDR, XML Schema type definitions are independent of instance declarations; this makes it possible to reuse type definitions in different contexts to describe distinct nodes within the instance document. (5) W3C support and industry implementation... creating specific and lucid schema should be your first task when creating XML- and Web Service-enabled applications. If your partners need other schema definitions than XML Schema, for example DTD, start with an XML Schema approach and then port the implementation. You'll come out ahead in the long run." See also the sample code for the article.

- [June 05, 2001] "XSD for Visual Basic Developers." By Yasser Shohoud. From the DevXpert Web Services Depot [for VB Developers]. May 2001. "The W3C's XML Schema is sometimes referred to as XML Schema Definition language or XSD for short. XSD is an XML-based grammar for describing the structure of XML documents. A schema-aware validating parser, like MSXML 4.0, can validate an XML document against an XSD schema and report any discrepancies. To solve the [invalid invoice document] problem outlined above, you'd create an XSD schema that describes the invoice document. You'd then make this schema available to the UI tier developers. The schema is now part of the 'interface contract' between the middle tier and the UI. While the application is in development, the UI tier can validate the invoice documents that they send against that schema to ensure they are valid. Similarly, the SaveInvoice function can validate the input invoice document against the schema before attempting to process it. Now if you change the invoice document to support a new feature, you must change the schema accordingly. Now the UI team tries to validate the invoice documents they're sending and this validation fails so they immediately realize that the schema has changed and that they must change the invoice documents they are

sending. This can also help catch version mismatch problems where you have an older client trying to talk to a newer middle tier or vice versa.... In this brief introduction to XSD, you've seen how you can make a Visual Basic class to an XSD schema and how to use that schema with MSXML 4.0 to validate documents. You also learned the relation between XSD and XML namespaces and how namespaces can be used to combine elements from different schemas in one XML document. This tutorial barely scratches the surface of what you can do with XSD schemas. There are many more features and details you might be interested in (or might not care about). Once you are comfortable with the concepts explained in this tutorial, check out the XML Schema Primer (part of the XSD specification) which goes into a lot more details about XSD with many examples..."

- [May 22, 2001] IBM XML Schema Quality Checker Supports the W3C XML Schema Recommendation. A communiqué from Bob Schloss (IBM Research) reports on the availability of an updated IBM XML Schema Quality Checker tool from IBM alphaWorks. The new version of this downloadable tool (Version 1.0.17, 05/21/2001) assists users who are creating XML Schemas conforming to the May 2, 2001 W3C Recommendation. The updated release "contains fixes bugs, adds a checker to verify that identity constraint definitions (Key, KeyRef and Unique) are consistent with the type of the element declaration where they appear, and updates the default stylesheet used to view the error reports. The XML Schema Quality Checker is a program which takes as input an XML Schema written in the W3C XML schema language and diagnoses improper uses of the schema language. Where the appropriate action to correct the schema is not obvious, the diagnostic message may include a suggestion about how to make the fix. For XML Schemas which are composed of numerous schema documents connected via `<include>`, `<import>`, or `<redefine>` element information items, a full schema-wide checking is performed. The tool can also be run in batch mode to quality-check multiple XML schemas in a single run." Schloss reports that the team is continuing to work on more complete checking of the consistency of identity-constraint definitions and on additional improvements; they welcome feedback and suggestions. The tool has been produced by the IBM Application Development team, including Achille Fokoué, Bob Schloss, Tom Gallivan, and Roberto Galnares. [*Full context*]
- [May 14, 2001] "XML Schema becomes W3C Recommendation: What This Means. With the approval of the W3C and its 500+ members, XML

is ready for the next big step to worldwide deployment." By Natalie Walker Whitlock (Casaflora Communications). From IBM developerWorks. May 2001. ['After more than two years of review and revision, the World Wide Web Consortium (W3C) announced on May 1 that it has embraced the XML Schema with a formal Recommendation. W3C Recommendation status is the final step in the consortium's standards approval process, indicating that the schema is a fully mature, stable standard backed by the 510 W3C member organizations.'] "Speaking at the 10th International World Wide Web Conference in Hong Kong, Web pioneer and W3C Director Tim Berners-Lee said that XML Schema (parts 0, 1 and 2) should now be considered as one of the foundations of XML, together with XML 1.0 and Namespaces in XML. He also stated that the specification provides 'an XML language for defining XML all languages.' The finalized Schema brings rich data descriptions to XML. Schema will solve the primary problem of B2B communication and interoperability that has held XML back from its full potential. The standardized Schema is expected to integrate data exchange across business, and ultimately realize the full promise of XML to facilitate and accelerate electronic business... Schema increases XML's power and utility to the developer by providing better integration with XML Namespaces. By introducing datatypes to XML, Schema makes it easier than ever to define the elements and attributes in a namespace, and to validate documents that use multiple namespaces defined by different schemas. XML Schema also introduces new levels of flexibility intended to speed its adoption for business use. According to [IBM's Noah] Mendelsohn, who also helped write the spec, XML Schema addresses a number of new issues and therefore has features for demanding apps. Yet, he says, developers can learn how to use XML Schema to do what they've been doing in XML with DTDs in 'about an hour or two.'... Berners-Lee added that XML Schema would need to be clarified and simplified after the many implementations and unexpected interpretations of the specification. Indeed, the cry of simplification has been one of the loudest heard from critics. The current complexity has been blamed for driving others to create alternative, lighter weight schemas, such as TREX and RELAX. Some have even said XML Schema is so complex that even some W3C insiders are calling for future versions to be incompatible with this first release so they do not repeat what critics say are the flaws of the first version... Despite the controversies, most groups have publicly stated that they will support and incorporate the W3C's XML Schema. These groups include IBM,

Microsoft, Sun Microsystems, Commerce One, and Oracle. In a public statement, Oracle said its Oracle9i will be the first production database to implement the new Schema. In addition, both Microsoft's .Net initiative and Sun's SunOne Web services effort will take advantage of XML Schema..."

- XSDSchema mailing list. Yahoo mailing list 'XSDSchema' "is for the purpose of open discussion and mutual help in relation to the W3C XML Schema specification which reached full Recommendation status at W3C on 2nd May 2001..."
- [May 03, 2001] Sun XML Datatypes Library Supports W3C XML Schema. A communiqué from Eduardo Gutentag reports on the availability of the 'Sun XML Datatypes Library'. Developed by Kohsuke Kawaguchi, the datatypes library is Sun's implementation of W3C's XML Schema Part 2 intended for use with applications that incorporate XML Schema Part 2. The preview version 1 of 'April 2001' implements the proposed recommendation version of the W3C XML Schema Part 2 Datatypes. The distribution of the XML Datatypes Library includes a sample class file `src/com/sun/tranquilo/datatype/CommandLineTester.java` provided "as a guide for implementing your own Java classes with the Datatypes Library." Documented examples include validating a string with an integer datatype, deriving a new type from an existing DataType object, and diagnosing errors. The library distribution includes software developed by the Apache Software Foundation; its use requires JDK 1.3. [*Full context*]
- [April 24, 2001] "XML Schema Catches Heat." By Roberta Holland. In *eWEEK* (April 23, 2001). "After more than two years of development, the World Wide Web Consortium could be only weeks away from releasing its long-awaited XML Schema specification. But despite its release, the specification, which is designed to automate data exchange between companies, is coming under fire. Now in the final review phase by W3C Director Tim Berners-Lee, the specification, according to critics, is far too complex -- so complex that it has driven several XML experts to create alternative and lighter-weight schemas. Furthermore, some W3C insiders are even calling for future versions to be incompatible with this first release so as not to repeat what they say are the flaws of the first version. 'There has been controversy,' said Tim Bray, co-author of the W3C's Extensible Markup Language specification, in Vancouver, British Columbia. 'XML Schema is a very large project. The working group is a very large body that has been very visible. All the major vendors are on it. As a result, the

[specification] tends to have a lot of compromises in it.' XML Schema has been one of the most watched standards efforts of late. The schema expresses shared vocabularies and defines the structure and content of XML documents. XML Schema is expected to make data exchange among businesses cheaper and easier than what is possible using Document Type Definitions. The comment period for XML Schema ended last week. The spec now lies with Berners-Lee, who will determine if any technical issues raised should prevent its release. W3C officials expect a decision within weeks.... Trex, which Clark submitted to the Organization for the Advancement of Structured Information Standards, is simpler and more modular, as it focuses just on the validation of XML documents. A similar effort, Relax, was started late last year by schema working group member Makoto Murata. Murata, who works with the International University of Japan Research Institute and IBM in Tokyo, also dissatisfied with the W3C's direction, said the schema group was focused more on benefits to vendors than on the technology, unlike the original XML working group. Clark and Murata recently merged their efforts under OASIS, in Billerica, Mass. Clark hopes they can produce a first draft in two to three months. Another alternative, called Schematron, was started in October 1999 by working group member Rick Jelliffe, who represented Academia Sinica Computing Centre, in Taipei, Taiwan, until this month. The most positive change 'is a widespread realization that XML Schema will not be the universal and terminal schema language,' said Jelliffe, now CTO of Topologi Pty. Ltd., in Sydney, Australia. 'I think if we can hose down people's expectations and mindshare-grabbing marketing, XML Schema will be successful.' [...] Despite the controversies, many are supporting XML Schema, including Microsoft, IBM and Oracle Corp. Micro soft last week announced a technical preview of its XML parser supporting schema. The Redmond, Wash., company also will include XML Schema in the second beta version of Visual Studio.Net, which will be given to attendees at Microsoft's TechEd conference in June."

- [April 20, 2001] "Taxonomy of XML Schema Languages Using Formal Language Theory." By Murata Makoto (IBM Tokyo Research Labs), Dongwon Lee (UCLA / CSD), and Murali Mani (UCLA / CSD). 24 pages. April, 2001. "On the basis of regular tree languages we present a mathematical framework for XML schema languages. This framework helps to formally describe, compare, and implement such XML schema languages. Our main results are as follows: (1) Four subclasses of regular tree

languages: local tree languages, single-type tree languages, restrained competition tree languages, and regular tree languages. (2) A classification and comparison of a few XML schema proposals and type systems:DTD, XML-Schema, DSD, XDuce, RELAX, and TREX (3) Properties of the grammar classes under two common operations: XML document validation and type assignment... As the popularity of XML increases substantially, the importance of XML schema language to describe the structure and semantics of XML documents also increases. Although there have been about a dozen XML schema language proposals made recently, no comprehensive mathematical analysis of such schema proposals has been available. We believe that providing a framework in abstract mathematical terms is important to understand various aspects of XML schema languages and to facilitate their efficient implementations. Towards this goal, in this paper, we propose to use formal language theory, especially tree grammar theory, as such a framework. Given an XML document and its schema, suppose one wants to check whether the document is valid against the schema and further find out the types (or non-terminals) associated with each element in the document. We are interested in algorithms for such document validation and type assignment operations, and the time complexity of such algorithms. Furthermore, we would like to know if such algorithms can be implemented on top of SAX or rather require DOM. Such issues are closely related with the efficient implementation of XML schema language proposals, and are directly addressed by our mathematical framework... [Conclusion:] A mathematical framework using formal language theory to compare various XML schema languages is presented. This framework enables us to define various subclasses of regular tree languages, and study the closure properties and expressive power of these languages. Also, algorithms for document validation and type assignment for the different grammar classes are described. Finally, various schema language proposals are compared using our framework, and the implementations available are discussed. Our framework brings forward a very important question: Do we need to migrate from deterministic content models of XML 1.0 in favor of schema languages such as RELAX and TREX that allow non-deterministic content models? Our work in this paper as well as other work, with regard to document processing and XML query, makes us believe that we should allow non-deterministic content models in XML schema languages. We have multiple directions for future research which we are pursuing presently. We are examining ambiguity in regular tree grammars and

languages, and studying how to determine whether a given regular tree grammar is ambiguous or not. We are also examining integrity constraints necessary for a schema language as studied widely in the area of database systems, and examining efficient implementations of these constraints for XML applications." [Note the comment posted: "I hope that this paper (submitted to Extreme) helps to understand validation algorithms for schema languages such as RELAX and TREX. You might have seen earlier versions of this paper, but this is away more readable... In my understanding, Algorithm 4.1 shown in this paper is similar to the algorithms of PyTREX, VBRELAX, and XDuce. The algorithm of RELAX Verifier for Java is based on Algorithm 5, but is more advanced. The algorithm of JTREX is more advanced than Algorithm 5 in that it constructs tree automata lazily. In the final version, I will try to add more information about JTREX."] References for related papers are given on Murali Mani's web site. [cache]

- [April 02, 2001] W3C XML Schema Specifications Developed in the 'OSS Through Java Initiative'. A communiqué from Ben Eng (Nortel) reports on "a significant XML Schema effort that has been underway in the 'OSS Through Java Initiative' for the past year. The first three API specifications being developed are for Service Activation, Trouble Ticketing, and Quality of Service; extending across all OSS through Java specifications is a common J2EE Design Guidelines document. All three API specifications are currently in Community Review ending April 16, 2001, at which time they will be promoted to Public Review status. We specify APIs in three styles: EJB session interfaces with Java Value Types, EJB session interfaces with XML Value Types, and XML messaging (transportable by JMS, ebXML/SOAP, or whatever). There is functional equivalence between the styles. Each OSS API will specify all three styles of APIs; specifying one automatically generates the other two. The latter two styles of interfaces are specified in XML Schema." [*Full context*]
- [March 29, 2001] "XML-Deviant: Schemas by Example." By Leigh Dodds. From XML.com. March 28, 2001. ['There has been a lot of activity in the area of XML schema languages recently: with several key W3C publications and another community proposed schema language. Another alternative schema language has emerged from the XML community, relying entirely on example instance documents.'] (1) "W3C XML Schema: The finish line is now in sight for the members of the W3C XML Schemas Working Group. The XML Schema specifications are an important step closer to completion with their promotion to

Proposed Recommendation status. All that remains now is for Tim Berners-Lee, as Director of the W3C, to approve the specifications before they become full Recommendations. The road has been long and hard, and it's had a number of difficult sections along the way." (2) Examplotron: "Eric van der Vlist has been helping to realize Rick Jelliffe's vision of a plurality of schema languages by publishing Examplotron, a schema language without any elements. Examplotron's innovation lies in its "'schema by example' approach to schema generation. Rather than define a dedicated schema language with which a document can be described, Examplotron uses sample instance documents, annotated with several attributes that carry schema specific information such as occurrence of elements, and assertions about element and attribute content. Like Schematron before it, Examplotron is implemented using XSLT. An Examplotron instance document can be converted into a validating stylesheet by applying a simple transformation..." For schema description and references, see "XML Schemas."

- [March 23, 2001] Examplotron 0.1." By Eric van der Vlist (Dyomedea). "The purpose of examplotron is to use instance documents as a lightweight schema language -- eventually adding the information needed to guide a validator in the sample documents. 'Classical' XML validation languages such as DTDs, W3C XML Schema, Relax, Trex or Schematron rely on a modeling of either the structure (and eventually the datatypes) that a document must follow to be considered as valid or on the rules that needs to be checked. This modeling relies on specific XML serialization syntaxes that need to be understood before one can validate a document and is very different from the instance documents and the creation of a new XML vocabulary involves both creating a new syntax and mastering a syntax for the schema. Many tools (including popular XML editors) are able to generate various flavors of XML schemas from instance documents, but these schemas do not find enough information in the documents to be directly useable leaving the need for human tweaking and the need to fully understand the schema language. Examplotron may then be used either as a validation language by itself, or to improve the generation of schemas expressed using other XML schema languages by providing more information to the schema translators..." From the XML-DEV posting: "Beating Hook, Rick Jelliffe's single element schema language has been quite a challenge, but I am happy to announce examplotron a schema language without any element. Although examplotron does include an attribute, this attribute is optional and

you can build quite a number of schemas without using it and I think it fair to say that examplotron is the most natural and easy to learn XML schema language defined up to know ;=) ... The idea beyond examplotron -and the reason why it's so simple to use- is to define schemas giving sample documents. Although examplotron can be used as a standalone tool, it can also be used to generate schemas for more classical -and powerful- languages and I don't think it will compete with them but rather complement them. Thanks for your comments..." See also: (1) the XML-DEV posting, and (2) "XML Schema Element and Attribute Validator."

- [March 16, 2001] "XML Schemas: Best Practice. [Homepage.]" By Roger L. Costello (Mitre). March 13, 2001. Table of Contents: Motivation and Introduction to Best Practices; Default Namespace - targetNamespace or XMLSchema?; Hide (Localize) Versus Expose Namespaces; Global versus Local; Element versus Type; Zero, One, or Many Namespaces; Variable Content Containers; Creating Extensible Content Models; Extending XML Schemas. Roger says: "I have created a homepage containing all of our work. Also, based upon our recent discussions (especially on Default Namespace) I have updated all the online material and examples. In so doing I fixed a lot of typos, clarified things, etc. [You can] download Online Material Plus Schemas: I have zipped up all the online discussions, along with the schemas and instance documents that are referenced in the online material. Now you can download all this material and run all the examples. Also download Best Practice Book: I have put the Best Practice material into book form. You can download this book and print it out... In a few days I would like to start up again our discussions on Creating Extensible Schemas..."
- [March 12, 2001] Tibco Releases Commercial Version of XML Canon/Developer. An announcement from TIBCO Software Inc. describes the availability of XML Canon/ Developer (XCD) which "enables organizations to build an XML infrastructure that accesses, stores, and integrates the vocabulary from schemas or DTDs in any XML-based application. XCD supports a 'logical schema analysis' approach for creating XML vocabularies and grammars which can then be re-purposed with new semantic meaning." XCD features include support for design-time repository for XML assets (document-level and component-level object control from a centralized repository) and distributed Web-based access to an organization's XML assets repository. The Web-based interface also leverages the Internet for collaboration with

suppliers, customers, trading partners and industry groups. XCD "enables the analysis of schemas and DTDs at the component-level by creating a data dictionary or vocabulary of an Enterprise's XML assets; this Enterprise vocabulary can then be browsed, searched, and re-constructed to create an infinite set of new semantically different schemas." [*Full context*]

- [March 09, 2001] White Paper Demonstrates 'Modeling XHTML with UML' and XML Schema Generation. A communiqué from Dave Carlson (Ontogenics Corp., Boulder, Colorado) reports on creation of an XML Schema that covers all of XHTML Basic (this may be the first complete XML Schema for XHTML Basic). Details are given in the white paper *Modeling XHTML with UML*. Carlson writes: "There are a 3-4 situations where it is a bit lenient in accepting markup that it shouldn't, but overall it seems to work quite well. This model makes very heavy use of inheritance to capture the XHTML concept of content groups, such as Flow, Block, Inline, etc. I have generated two different schemas: one uses extension of complexType definitions, the other employs a copy-down strategy to avoid extension. Both schemas work with the XSV validator... What's interesting about this is that the schema was automatically generated from a UML model. The white paper includes all the UML class diagrams for the XHTML Basic modules. I've written a schema generator that produces schemas from any UML tool that can export an XMI 1.0 document representing the model. This model of XHTML was created using Rational Rose... the generated schema also provides a good stress test case for validation tools." [*Full context*]
- [March 07, 2001] Schema For Representing Infosets Explicitly in XML. From Richard Tobin (HCRC, University of Edinburgh). 2001-03-07 or later. "This is a schema for representing infosets explicitly in XML. There are two main versions: one for the basic infoset, and one for the post-schema-validation (PSV) infoset. One reason for producing this schema was to allow comparison of the infosets generated by different processors (including parsers and schema validators). It has also proved useful for finding flaws in the infoset and schema specifications themselves..." In the '20010216' version, "the basic schema matches the Infoset Last Call draft; the PSV schema has not yet been updated to match." See also earlier versions online. Contents: (1) XMLInfoset.xsd; (2) XMLSchema-infoset.xsd; (3) infoset-basic-items.xsd; (4) infoset-basic-properties.xsd; (5) infoset-basic-types.xsd; (6) infoset-schema-components.xsd; (7) infoset-schema-facets.xsd. [cache 2001-03 for '20010216' version']

- [March 07, 2001] "Mapping W3C Schemas to Object Schemas to Relational Schemas." By Ronald Bourret (The Open Healthcare Group). March 2001. "This paper summarizes two different mappings. The first, part of the process generally known as XML data binding, maps the W3C's XML Schemas to object schemas. The second, known as object-relational mapping, maps object schemas to relational database schemas. The two mappings can be joined (and the intermediate object schema eliminated) to create a mapping from XML Schemas to database schemas. This is not shown, but left as an exercise to the reader. Note that because individual XML Schema structures can often be mapped to multiple object structures, and because individual object structures can often be mapped to multiple database structures, there are usually multiple possible mappings from XML Schemas to database schemas. The mapping is described in terms of the data model presented in XML Schemas Part 1: Structures, rather than the XML syntax used to describe schemas. Although I might eventually add a section describing the mapping based on the XML syntax, this is currently left as a (non-trivial) exercise for the reader...The purpose of this paper is to help people write code that can automatically generate object and database schemas from XML Schemas, as well as transferring data between XML documents, objects, and databases according to mappings between them. Because the set of possible mappings from XML Schemas to object schemas is fairly large, I do not expect any software to support all possible mappings any time soon, if ever. A more reasonable strategy is for the software to pick a subset of mappings that make sense for its uses and implement those." [Introduction on XML-DEV: 'I've posted a paper mapping a (very slight) variant of the data model in W3C schemas to object schemas, and then mapping object schemas to relational schemas. The first part of the paper -- mapping XML schemas to object schemas -- is likely to be of most interest to people. It is undoubtedly similar to Sun's XML data binding (JSR-31) and Veo Systems work with SOX. In fact, I wrote it because neither of those specifications seems to be publicly available. The work also appears to be a superset of the mappings in Bill La Forge's Quick and Enhydra's Zeus project. Please note that the paper is rather terse and assumes you understand the general ideas behind the mapping from XML schemas / DTDs to object schemas. If not, see the presentation "Mapping DTDs to Databases", available from: http://www.rpbourret.com/xml.'] [cache]
- [March 06, 2001] "Extending XML Schemas." By

Roger L. Costello (*et al.*). XML-DEV post March 06, 2001. Topic: 'What is Best Practice of checking instance documents for constraints that are not expressible by XML Schemas?' "XML Schemas - Strive to be All Powerful? As XML Schemas completes version 1 and begins work on version 2, the question comes to mind: 'should XML Schemas strive in the next version to be all powerful?' Programming languages seem to have that goal - to enable a programmer to express any problem using the language. Perhaps the goal of version 2 of XML Schemas should be to provide enough flexibility that any constraint may be expressed. Alternatively, perhaps XML Schemas should just provide a core set of constraint expressing mechanisms (as it does today), and let the marketplace create a technology (technologies?) to supplement XML Schemas. Then version 2 of XML Schemas would have few changes from version 1..."

- [March 05, 2001] "Comparing W3C XML Schemas and Document Type Definitions (DTDs). [XML Matters #7.]" By David Mertz, Ph.D. (Idempotentate, Gnosis Software, Inc.). From IBM developerWorks, XML Library. March 2001. ['Many developers expect that XML schemas will soon supplant DTDs for specifying XML document types. David Mertz is skeptical that schemas will replace DTDs, though he believes that XML schemas are an invaluable tool in a developer's arsenal. This installment of the "XML Matters" column steps up to the challenge of comparing schemas and DTDs and clarifying just what is going on in the XML schema world.'] "While there are a number of instances where W3C XML Schemas excel, there remain, nonetheless, a number of areas where DTDs are better. Developers are continually left with tough choices... Much of the point of using XML as a data representation format is the possibility of specifying structural requirements for documents: rules for exactly what types of content and subelements may occur within elements (and in what order, cardinality, etc.). In traditional SGML circles, the representation of document rules has been as DTDs -- and indeed the formal specification of the W3C XML 1.0 Recommendation explicitly provides for DTDs. However, there are some things that DTDs cannot accomplish that are fairly common constraints; the main limitation of DTDs is the poverty in their expression of data types (you can specify that an element must contain PCDATA, but not that it must contain, for example, a nonNegativeInteger). As a side matter, DTDs do not make the specification of subelement cardinality easy (you can compactly specify 'one or more' of a subelement, but specifying 'between seven and

twelve' is, while possible, excessively verbose, or even outright contorted). In answer to various limitations of DTDs, some XML users have called for alternative ways of specifying document rules. It has always been possible to programmatically examine conditions in XML documents, but the ability to impose the more rigid standard that, 'a document not meeting a set of formal rules is invalid,' essentially, is often preferable. W3C XML Schemas are one major answer to these calls, but not the only schema option out there... At least two fundamental and conceptual wrinkles remain for any 'schemas everywhere' goal. The first issue is that the W3C XML Schema Candidate Recommendation, which just ended its review period on December 15, 2000, does not include any provision for entities; by extension, this includes parametric entities. The second issue is that despite their enhanced expressiveness, there are still many document rules that you cannot express in XML schemas (some proposals offer to utilize XSLT to enhance validation expressiveness, but other means are also possible and in use). In other words, schemas cannot quite do everything DTDs have long been able to, while on the other hand, schemas also cannot express a whole set of further rules one might wish to impose on documents. At a more pragmatic level, tools for working with XML schemas are less mature than those for working with DTDs... W3C XML Schemas let XML programmers express a new set of declarative constraints on documents for which DTDs are insufficient. For many programmers, the use of XML instance syntax in schemas also brings a greater measure of consistency to different parts of XML work; others disagree, of course. Schemas are certainly destined to grow in significance and scope as they become more familiar, and as developers enhance more tools to work with them. One way to get a jump start on schema work is to automate the conversion of existing DTDs to XML schema format. Obviously, automated conversions cannot add the new expressive capabilities of XML schemas themselves; but automation can create good templates from which to specify the specific typing constraints one wishes to impose." Also in PDF format, cache.

- [February 17, 2001] Abbreviated Tag Names / ASN.1 / binary encodings for XML Schema data types. From Charles Reitzel, XML-DEV. "Surely, ASN.1 is preferable to inventing a new set of binary encodings for XML Schema data types. Besides being available now, ASN.1 has the advantage of interoperability with LDAP. I'm sure there are many uses for representing LDAP data as XML and vice-versa: lookup web service deployment descriptors, directory data export/

import, UDDI registry implementation, ... LDAP already defines "Syntaxes" for many primitive data types (date, string, integer, ...). LDAP has the ability to use different names for the same OID, which might be helpful in supporting multiple XML views of shared LDAP data. Conceivably, these attribute types could be used independently of directories, per se. However, done with care and restraint, it shouldn't be to hard find a usable overlap between LDAP schemas and XML Schema. Both support basic "struct" data types as well as attribute and element/entry type inheritance. Complications: 1) Because LDAP attributes are multi-valued, however, they don't always map to XML attributes. So some additional meta-data is needed here. 2) It is difficult to separate LDAP object classes from directories. An XML data type is probably needed to represent an LDAP distinguished name as a URI ("urn:ldap: cn=joebob,l=texas")? However, LDAP relative distinguished names (RDNs) can probably be represented with a properly scoped XML Schema key definition."

- [February 14, 2001] MSL - A model for W3C XML Schema. Conversations by Dan Connolly, Andrea Asperti, and Philip Wadler on the the public W3C 'spec-prod@w3.org' mailing list [forum for discussion of W3C Spec Production Issues] have (incidentally) referenced MSL (Model Schema Language), which represents "an attempt to formalize some of the core idea in XML Schema." MSL, as with Hypertextual Electronic Library of Mathematics (HEML) and the online The COQ proof assistant, may be of interest to XML developers having expertise in mathematics and formal logic. A presentation entitled "MSL: A model for W3C XML Schema" will be given in an XML Foundations session at the Tenth International World Wide Web Conference (WWW10). The same WWW10 session, chaired by Carl Lagoze of Cornell University, will feature also "A Unified Constraint Model for XML" (Wenfei Fan, Gabriel M. Kuper, Jerome Simeon) and "Keys for XML" (Peter Buneman, Susan Davidson, Wenfei Fan,Carmem Hara, Wang-Chiew Tan). An online draft paper indicates that "MSL has already proved helpful in work on the design of XML Query; we expect that similar techniques can be used to extend MSL to include most or all of XML Schema." According to one online authority, MSL (Model Schema Language) is "an initiative by members of the W3C's XML Schema Working Group, to provide a formal model for the XML Schema language." Cf. the Software AG Glossary, which notes also a 'New MSL Draft*Full context*]

- [February 09, 2001] "The Hook: A Minimal Validation Language of One Element Based on Partial Ordering." By Rick Jelliffe. 2001/02/07. "The Hook validation language is a thought experiment in minimalism in XML schema languages. The purpose of such a minimal language would be to provide useful but ultra-terse success/fail validation for basic incoming QA, especially of datagrams. It is like a checksum for a schema. The validation it performs can be characterized as "Does this element have a feasible name, ancestry, previous-siblings and contents?", there being some tradeoff between the how fully the later criteria are tested. Let us start with the following technical criteria: (1) Smaller than DTD: if it is downloaded from a server as a separate file, it should be downloadable in the first packet group, so less than 512 (the minimum MTU) -100 (for MIME header) =412 bytes; (2) Implementable by a streaming processor; (3) No forward references; (4) No pathological schemas as far as blowouts; (5) An efficient implementation should be possible; (6) Suitable for coarse validation of document for some significant issues; (7) The schema should be namespace-aware; (8) The minimal schema should only require 1 element or perhaps fit in a PI; (9) The datatype should be expressible using XML Schemas regular expressions or simple space-separated tokens; (10) The schema paradigm is the (partial) ordering of elements against the information kept during stream processing... A Hook schema is an element containing a list of element names, some of which may be grouped by square brackets. This list represents a certain ordering of the names and validation consists of checking conformity to this ordering. The DTD for the language is [7 lines]... Hook seems to suit languages that have large flat bottoms, languages specific requirements early on in each content model, languages with specific elements that do not re-occur in different contexts with different priorities, languages with attributes that are not vital or will be checked by other mechanisms. Hook would seem useful as a coarse-grained but ultra-terse validation language. If we say that validation is to catch errors that are most likely to happen, the most likely errors are spelling errors, children in the wrong order, and required parents: Hook gets or catches most. How much would this help an interactive editor? It would know which elements can start, but for new documents it would present to many choices: however if editing existing documents it would cull the available list pretty well, because it would know what the current level was. It would know empty elements... Joe English has posted interesting material

regarding formalisms for Hook, algorithm for implementing and other material..."

- [February 09, 2001] "The Politics of Schemas: Part 2." By Kendall Grant Clark. From XML.com. February 07, 2001. ['Having established in the first half of this essay that schemas are essentially political, this second installment examines the relevance of this to the XML community, and avenues for further consideration.'] "You may find yourself agreeing that schemas are political but wondering, nevertheless, what it has to do with XML practitioners or with XML itself. XML is, however, a universal data format. If we take the universal claims made about XML seriously, professional schema-makers must ask whether some interests and views of contested concepts might be excluded, perhaps systematically, from schema-making and from schemas; whether such exclusion is socially beneficial or harmful; and, if harmful, what should be done about it. From the early days of XML's development there's been talk about vendor neutrality, interoperability, and universality. Such talk was part of SGML's appeal since the mid-70s and rightly so. Today that talk fails regularly to take account of politics. XML advocacy often ignores the fact that schemas may be vendor neutral but cannot be interest neutral; that schemas may be universally accessible but formalize a strongly contested understanding of a vital part of the world; or that schemas may distort or impede some people's interactions with the world in ways they find inequitable or inappropriate. XML schemas are often placed in the public domain and available for anyone's royalty-free use (subject obviously to uncommon levels of knowledge and expertise) -- a state of affairs clearly preferable to proprietary alternatives. But is it enough? What good does it do that one can use, even modify a de facto standard schema, royalty free, when the schema reflects interests inimical to one's own, formalizes an understanding of the world one strongly contests, and is used in a widely deployed, vital Semantic Web application that has no serious competitor? What good does it do to modify the schema to reflect one's own interests and understandings if doing so renders it unusable? [...] Political schemas may limit what we notice, what we can say or think about what we notice, and to whom we can say it, especially inasmuch as we use machines to mediate parts of the world to us. The Semantic Web vision means, if anything at all, creating software systems that mediate the world to some of us in useful and, one hopes, fair, just, and good ways. What XML technologists say and think and do about the politics of schemas, the Semantic Web, and the social benefits of the technology they create will go a long way to determining the Web's future,

and maybe something of society's future too. I hope I at least have said enough to encourage the wide-ranging and free conversation it is the responsibility of XML technologists, along with others, to have."

- [February 09, 2001] "XML-Deviant: Schemarama." By Leigh Dodds. From XML.com. February 07, 2001. ['For the past two weeks XML-DEV has seen fascinating exchanges between three inventors of alternative XML schema proposals.'] "During the last week, XML-DEV has been the scene of a series of interesting and innovative discussions concerning schemas in general and also specific schema languages. The XML-Deviant provides a round-up. Grammars Versus Rules: Most schema languages rely on regular grammars for specifying schema constraints, a fundamental paradigm in the design of these languages. The one exception is Schematron, produced by Rick Jelliffe. Schematron throws out the regular grammar approach, replacing it with a rule-based system that uses XPath expressions to define assertions that are applied to documents... A unique feature of Schematron is its user-centric approach, allowing useful feedback messages to be associated with each assertion. This allows individual patterns in a schema to be documented, giving very direct feedback to users. Indeed a recent comparison of six schema languages highlights how far Schematron differs in its design. At times the discussion strayed into comparisons of several schema languages. Rick Jelliffe provided his interpretation of the different approaches behind TREX, RELAX, XML Schemas and Schematron: 'Underlying Murata-san's RELAX seems to be that we should start from desirable properties that web documents need: lightweightedness, functioning even if the schema goes offline (hence no PSVI) and modularity. I think underneath James Clark's TREX is that we can support plurality if we have a powerful-enough low-level schema language into which others can be well translated. I think underlying W3C XML Schemas is that a certain level of features and monolithicity is appropriate (though perhaps regrettable) because of the need to support a comprehensive set of tasks and to make sure that there are no subset processors (validity should always mean validity); however the processors are monolithic but the schemas are fragmented by namespace. Underlying Schematron is that we need to model the strong (cohesive) directed relationships in a dataset and ignore the weak ones, that constraints vary through a document's life cycle, and that lists of natural language propositions can be clearer than grammars.' [...] James Clark's summary of the advantages of TREX over W3C XML Schemas is

also worth reading in its entirety. TREX, like Schematron, is a very simple yet powerful schema language..."

- Data Sheet for the Hackerlab Rx XML Regular Expression Matcher. "Hackerlab Rx-XML is a regular expression pattern matcher for Schema-capable validating XML processors. It is also a general purpose Unicode regular expression matcher. Key Features: Rx-XML is fast and accurate; Supports the regular expression language specified in the W3C document 'XML Schema Part 2'; Supports alternative regular expression syntaxes; Clean and simple 'classic C' interface; Patterns may use UTF-8 or UTF-16; Strings compared to compiled patterns may use UTF-8 or UTF-16; Provides protection against encoding-based illegal data attacks: Ill-formed encoding sequences (*e.g.*, non-shortest form UTF-8) are detected and rejected during regular expression compilation and matching; Configurable space/time trade-offs; Ready for Unicode 3.1: Designed for a character set with 2^21 code points; Validation tests are included; Postscript and HTML documentation is included. Hackerlab Rx-XML is part of the Hackerlab C Library which is distributed under the terms of the GNU General Public License, Version 2, as published by the Free Software Foundation..." See also the An Introduction to XML Regular Expressions [sw].
- [February 05, 2001] "XML Schema Slowly Matures. XML Schema can't fix everything by itself, but it fills a gaping hole in the XML group of technologies and specifications." By Don Kiely (Third Sector Technologies). In *XML Magazine* Volume 2, Number 1 (February/March 2001). ['XML's Document Type Definition provides a means of defining XML structure. DTDs are well supported in the software industry, but they come with a substantial set of problems, too. Can this marriage be saved? Find out how XML Schema may save structured XML.'] "Now that the XML Schema specification is a W3C candidate recommendation, it is entering a period of its life when the standards committee thinks that all the basic parts are there and working. There are still a few kinks to work out, but people are encouraged to start building proof-of-concept tools and applications. With any luck, it will hit full maturity sometime later this year, and we'll have full benefit of all its features. But what's the big deal about those features? The XML 1.0 recommendation has a means of defining XML structure built into it, the Document Type Definition, or DTD. DTDs are well supported in the software industry because of their origins with SGML, and XML is a derivative language of SGML. There are lots of DTDs out there doing lots of good work, and lots of people

understand them well. There is a wealth of books, journal articles, and Web resources that provide plenty of information about them. The problems with DTDs are several. DTDs are a decidedly non-XML syntax that is hard to learn, they have no sense of data types and only the loosest limits on some structural constraints, they have a closed architecture, they lack support for namespaces, and generally they do not adhere even to the most trivial of the goals built into the design of XML. Probably the biggest issue driving the XML Schema specification is the lack of data types in DTDs. Even if you ship me an XML document that declares itself to be strictly compliant with a DTD, the XML data can have almost random data as element content and attribute values, even if the element name clearly suggests an integer or floating point number, for example. Very messy, and very unlike XML... The listings and code that accompany this article use sample XML data from the XML Schema Part 0: Primer candidate recommendation document for purchase order data... In general, you'll want to validate XML data that is shared between different applications, particularly if you don't have control over both applications. This way the application that is consuming the XML data doesn't need to have extensive data-checking code to make sure that the data is usable and in the structure it expects. Depending on the validation structure you use, you still may need to do some programmatic error checking. For example, when using a DTD for validation, you'll still need to check that content and attribute values can be converted from their string representation to the type of data you are expecting, such as currency or date values. On the other hand, if you have control over both ends of the data sharing, such as two applications you wrote or between two components in a single application, you may be able to forego validation and save the processing cycles. It really boils down to yet another of many design and architecture decisions necessary for software development. XML Schema, unlike XML itself, is unlikely to cure the world of all its ills. But it fills a gaping hole in the XML group of technologies and specifications and can achieve full status as a W3C recommendation none too soon."

- [February 02, 2001] "The Status of Schemas." By Steve Gillmor and Sean Gallagher. In *XML Magazine* (February/March 2001). ['The XML Schemas specification, now one step closer to finalization, will enhance XML document exchange on the Web'] "The W3C XML Schema specification has advanced to candidate recommendation status after several years of effort. Editorial director Sean Gallagher and editor in chief Steve Gillmor talked with IBM E-Business

Standards & Technology Lead David Fallside, IBM representative to the W3C Schema Working Group and the W3C Advisory Committee and chair of the XML Protocol Group, and Lotus Distinguished Engineer Noah Mendelsohn, IBM and Lotus representative to the W3C Schemas Working Group and the W3C Advisory Committee. Mendelsohn: [an overview of the significance of the XML Schema specification as it reaches CR] 'XML provides a standard means of interchanging data and documents, especially on the World Wide Web. XML Schemas is a standard way of interchanging descriptions of those documents, and that's important not only because it gives you a way to validate that the document you receive is in some sense correct -- that it meets at least the minimum standards for format and content -- but Schema descriptions will be extremely important in supporting a variety of tools. There will be Schema-aware editing systems that use the Schema to help create a better editing experience. There will be tools for mapping XML into various database systems that will use the information in the Schema to find out what needs to be mapped, or will produce Schemas to expose to the world the kind of information that they're making available. Schemas will be key for building XML queries -- before you know what you're querying, you have to know what it looks like. Schemas as a whole are a core foundation technology that moves XML forward for all of these applications.' Fallside: 'Candidate Recommendation [CR] is the stage where the W3C solicits implementation experience from other W3C members and the world at large on the XML Schema specification. Achieving candidate recommendation status is a way for the W3C to say that we believe that this is a stable draft of the specification; it's stable to the extent that people should be comfortable building implementations using this specification, and we would like your feedback based on those implementations.'..."

- [January 31, 2001] XML Schema FAQ. By Francis Norton. 2001-01-31. See the supporting documents on the www.SchemaValid.com web site. The source for the FAQ document is XML conforming to a FAQ Schema; it has been formatted into HTML with FAQ Stylesheet. "[XML] DTDs have several limitations, one of which is the fact that they are not written in standard XML data syntax. This means, for instance, that while it is quite possible to write an XSLT transform to document an XML Schema, there are far fewer tools to process DTDs. XML Schema also offers several features which are urgently required for data processing applications, such as a sophisticated set of basic data types including

dates, numbers and strings with facets that can be superimposed - inlcuding regular expressions and minimum and maximum ranges and lengths..." [cache from new URL]

- [January 19, 2001] "XML Schema Extension Mechanisms." By David E. Cleary. December 2000. "This presentation documents the two methods for extending XML Schema with application specific information. It includes examples of real world uses of these extensions today." Examples: "(1) `XMLSchema-hasFacetAndProperty.xml` - This is the schema for the facet and property appinfo extension. This interesting thing to note is that the human readable documentation in the schema specifies how an application should use this extension. For instance, it specifies how you determine what facets and properties user defined datatypes supports via walking the basetype chain. (2) `enum2.xml` - This is an annotated version of TextNumbers that includes localized enumerations. Applications that support this extension can use these localized enumerations for their UI instead of relying on the English version. (3) `appinfo.xml` - This is the XML Schema definition for the appinfo element taken directly out of the Schema for Schemas. It shows it allows an attribute called source that is of type uriReference. It also supports mixed content (*i.e.*, both character data as well as child elements) and uses the "any" wildcard to specify it can have any content. Make note of the processContents attribute that is set to lax, which sets validation rules. (4) `string.xml` - A fragment of the XML Schema datatypes schema. This schema uses an appinfo extension that specifies what property and facets a datatype supports. This extension is also used in generating the datatypes specification." [cache]
- [January 19, 2001] "4th Generation XML Application Development." By David E. Cleary. December 2000. "This presentation discusses an application development methodology that relies on molding XML instance data to your application as opposed to writing your application based on the XML vocabulary used. It details how Progress Software uses schema annotation to map XML data to business logic and includes a example of using this methodology to map XML instance data to existing Java classes... The SchemaMapper application requires the Xerces-J Parser from Apache to be in your classpath. If you are using Microsoft's JVM, you can do this by adding an entry for xerces.jar in the registry. To use the SchemaMapper, you just give it a qualified filename of an XML instance document. The instance document must conform to a schema located in the Schemas directory." See the

announcement. [cache]

- [January 16, 2001] "The W3C XML Schema Specification in Context." By Rick Jelliffe. From XML.com. January 10, 2001. ['This article compares the W3C XML Schema Definition Language with XML document instances and DTDs, SGML DTDs, Perl regular expressions, and alternative schema technologies such as RELAX and Schematron.'] "This article gives simple comparisons between the W3C XML Schemas and [related formalisms:] W3C XML instances, W3C XML DTDs, ISO SGML DTDs, ISO SGML meta-DTDs, Perl regular expressions. And some technologies that have arisen as a response to it: JIS RELAX, Schematron, DSD. It does not provide an exhaustive list of all W3C XML Schemas features. The information was prepared with the October Candidate Recommendation versions in mind. W3C XML Schemas does not operate on marked-up instances per se, but on the information set of a document after it has been parsed, after any entity expansion and attribute value defaulting has occurred. Think of it as if it were a process looking at the W3C DOM API. The result of schema-validating a document is a set of *outcomes* giving, in particular, any violations of constraints -- there is currently no standard API for this; however the W3C XML Schemas specification gives a complete list of the constraint violations; an enhanced information set, the *post-schema-validation information set*, which can include various details about type and facets -- there is currently no standard API for this either; however, the W3C XML Schemas specification gives a complete list of the additional information...W3C XML Markup Declarations (DTDs) are geared to provide simple datatyping on attributes sufficent to support graph-structures in the document only. W3C XML Schemas are intended to provide a systematic datatyping capability. W3C XML DTDs provide a basic macro facility, parameter entities, with which many good effects can be achieved. W3C XML Schemas reconstructs the most common of these in various high-level features..."
- [January 13, 2001] Updated DTD to XML Schema Conversion Tool. Mary Holstege posted an announcement for the release of an updated version of a DTD to XML Schema tool. I am attaching an updated version of the Perl script ['dtd2xsd.pl'] that is available on the W3C site. This new version makes the following changes: (1) Use the CR syntax instead of the April [XML Schema] draft syntax; (2) Add support for an external mapping file for type aliases, simple types, model, attribute, and substitution groups; (3) Map ANY correctly to wildcard rather than element 'ANY'; (4) Support for treating lead

PCDATA as string or other aliased simple type instead of as mixed content (may be more appropriate for data-oriented DTDs) *e.g.*, `<! ELEMENT title (#PCDATA)> => <element name="title" type="string"/>`; (5) Support subsitution groups (simplistically). For the record: this update has no official standing... It is worth pointing out that this tool does not produce terribly high quality schemas, but it is a decent starting point if you have existing DTDs." See Mary Holstege's web site for samples and documentation on this version of the application.

- [January 13, 2001] Zvon XML Schema Reference. By Miloslav Nic. January 09, 2001. "This reference is based on W3C Candidate Recommendations for XML Schema Part 1: Structures and XML Schema Part 2: Datatypes. This reference will be upgraded when the standard is finalized. This reference consists of two parts: (1) Schema browser - based on the analysis of normative XML Schema; (2) DTD browser - based on the analysis of non-normative DTD. Main features of the XML Schema reference include: (1) Clickable indexes and schemas. (2) Click on 'Annotation Source' leads to the relevant part of the specification." The Zvon web site provides tutorials for a wide range of XML-related technologies (DOM, XSLT, CSS, XML DTDs, XHTML, XLink, XPointer, SVG, etc.).
- [October 12, 2000] (Beta) XSDs for XMLSpec, XLink, and Namespaces documents. Note of 12 Oct 2000: "As my initial foray into non-toy XML Schemas, I took a stab at creating an XML Schema for the XML Specification doctype. My goal was to write an XML Schema that was configurable in the same general way that Maler/Andaloussi-methodology DTDs are, and that would validate the same set of documents (modulo additional constraints that DTDs can't express). I'm sure I've overlooked things, possibly large things, but the schema at http://dev.w3.org/cvsweb/spec-prod/schema/ now validates the XML 2e Recommendation with XSV 1.166/1.77 of 2000/09/28 15:54:50." - Norm Walsh. See "XML Specification DTD."
- [February 26, 2000] Comparison of XML/SGML DTDs and the W3C XML Schema Specification. Rick Jelliffe (Academia Sinica Computing Centre) announced the availability of a learning document which compares SGML/XML DTDs with the new W3C XML Schema specification. The document will be useful alongside the W3C's new *XML Schema Part 0: Primer* as an aid to understanding the specification. "This note gives simple comparisons between XML Schemas and the technologies that have influenced it. *The XML Schema Specification in Context* does not provide

an exhaustive list of all XML Schemas features. [...] As an aid to the bewildered, I have started making a little note comparing XML Schemas from the new [2000-02-25 W3C] draft and: (1) XML DTD; (2) SGML DTD, and (3) SGML meta-DTD (architectures). A draft is at http://www.ascc.net/~ricko/XMLSchemaInContext.html. This draft is not suitable/stable/correct enough for linking or reference, but I hope some people may find it useful. Any improvements are welcome. Sections on Murata-san's RELAX or Nils' DSD or other schema languages would be nice too; I may put in something about Schematron too; I hope we can avoid juvenile acrimony and have friendly discussions on the features and niches that the various Schema languages will lend themselves to." Note, in this connection, a recent request from C. Michael Sperberg-McQueen and Dave Hollander (Co-chairs, W3C XML Schema Working Group) that the new three-part XML Schema specification *be read* in preparation for the upcoming XML Schema discussion at XTech 2000: "Those interested in XML Schema may want to take a look at the new draft before the town-meeting on XML Schema Tuesday night at XTech." The URLs are: (1) Primer, (2) Structures, and (3) Datatypes.

- [March 31, 2001] XSV and XSU: see PR 2001-0-316 version of XSV and XSU.
- [June 22, 2000] [See previous entry.] New UNICODE Version of W3C/LTG XML Schema Validator Released. Henry S. Thompson (W3C XML Schema Structures Co-Editor; HCRC Language Technology Group, University of Edinburgh) recently announced an updated release of the online W3C/LTG 'XML Schema Validator, XML Output Version'. "The original 8-bit-only, text output version of XSV has been retired, as signalled last week. The full UNICODE version, with `text/xml` output, is now the main line public version, and it's at a new address, [was: http://www.w3.org/2000/06/webdata/xsv]. The usage of XSV is up -- running at roughly 100 validations a day, with a high of nearly 200 last Friday. Thanks to those who tick the 'Contribute' box -- I'm about to harvest recent contributions and expand the regression test suite to reflect the increase in breadth of usage. The latest version has a number of bug fixes and improved compliance in the area of enforcing content-model determinism. Of the not-yet-implemented-by-XSV aspects of XML Schema, I would welcome feedback on what users are most keen to see covered first, [from among]: (1) Simple type conformance, other than enumerations and max/min for numeric types; (2) Detailed enforcement of derivation by restriction; (3) Full XPath expressions for identity constraints; and (4) Post-schema-validation infoset

contributions." Readers are invited to note, in this connection, activity on the **xmlschema-dev@w3.org** discussion list, which is publicly archived on the W3C server. Henry Thompson announced this public list on April 07, 2000 with a message "'XML Schema Developers List Launched': To accompany the XML Schema Last Call drafts, the W3C is pleased to announce the opening of a public mailing list for XML Schema implementation developers, `xmlschema-dev@w3.org`. To subscribe, send mail to xmlschema-dev-request@w3.org with 'subscribe' as the subject."

- [April 27, 2000] [See previous entry.] XSV: W3C/HCRC Language Technology Group XML Schema Validator. The W3C Web site now hosts an 'XML Schema Validator', referenced from a document entitled 'W3C/HCRC Language Technology Group Schema Validator'. [was: http://cgi.w3.org/cgi-bin/xmlschema-check] The XML Schema validator has been provided by Henry S. Thompson and Richard Tobin (Language Technology Group of the Human Communication Research Centre in the Division of Informatics, University of Edinburgh); the Web interface is from Dan Connolly. XSV (XML Schema Validator) is an open source (GPLed) work-in-progress attempt at a conformant schema-aware processor, as defined by *XML Schema Part 1: Structures*. Documentation on the current status of XSV is provided on the LTG Web site. The easiest way to use the Schema Validator is to enter one or more URL in the forms interface. The form provides a check-box to check if you want to attempt schema-validation even if the schema(s) have errors. Additionally, one may download the (Python) sources from the W3C public CVS repository. **Note:** also now the announcement of 2000-05-05 posted to XML-DEV: "You are invited to experiment with the open source Edinburgh/W3C schema validator XSV, via a webpage interface. It will schema-validate instances using schemas, and as an obvious special case, check schemas against the schema for schemas. This is an alpha release, which is undoubtedly buggy, and known not to check everything it should, but please *do* use it, and if you are willing, tick the box which lets us copy your schemas to build up a regression test suite." See also the reference page for XML 'DTD' and well-formedness validation. [XSV noted by Don Box as "Henry Thompson's most excellent schema validator" 2000-04-27] Note 2000-05-28 - new test version with XML output options.
- [August 10, 2000] 'XML serialization of a XML document'. "We have been developing a schema

for representing the infoset of a document, with the intention of using it to compare the output of XML Schema implementations. See the directory www.cogsci.ed.ac.uk/~richard/infoset/0718/. infoset-basic-subset.xsd is a schema allowing any subset of the infoset. infoset-psv-subset.xsd is the same for the post-schema-validation infoset. From Richard Tobin, posting to XML-DEV, 2000-08-10. [cache]

- [December 01, 2000] "Tutorials: Using XML Schemas. Part 1." By Eric van der Vlist. From XML.com (November 29, 2000). ['In the first half of this introduction to XML Schemas, a W3C XML language for describing and constraining the content of XML documents, we cover the basics of creating structured, readable schemas.'] "This is the first of a two-part introduction to the W3C's XML Schema technology. XML Schemas are an XML language for describing and constraining the content of XML documents. XML Schemas are currently in the Candidate Recommendation phase of the W3C development process... The second part of this tutorial will cover mixed-content types, identity constraints, building reusable schemas, using namespaces, and referencing schemas from instance documents." See also Part 2.
- [December 22, 2000] "Using W3C XML Schema. Part 2." By Eric van der Vlist. From XML.com. December 15, 2000. ['The second half of a comprehensive introduction to the W3C's XML Schema Definition Language, including coverage of namespaces, object-oriented features and instance documents.'] "In the first part of this series we examined the default content type behavior, modeled after data-oriented documents, where complex type elements are element and attribute only, and simple type elements are character data without attributes. The W3C XML Schema Definition Language also supports the definition of empty content elements, and simple content elements (those that contain only character data) with attributes..."
- [December 22, 2000] "W3C XML Schema Tools Guide." ['A run-down of editors, validators and code libraries with support for XML Schema.'] By Eric van der Vlist. From XML.com. December 15, 2000. "The list of tools supporting XML Schema is still short, reflecting the fact that the specification is not yet a W3C Recommendation. When using a tool, check that it supports the version of XML Schema you are expecting: we've listed the support available at the time of writing. The most recent version of XML Schema is the Candidate Recommendation, dated 2000/10/24."
- [December 01, 2000] "Reference: W3C XML Schema Structures Reference." By Eric van der

Vlist. From XML.com (November 29, 2000). ['A complete quick reference to the elements of the W3C XML Schemas Structures specification, including content models and links to the original definitions.'] "The quick reference below has been created using material from the W3C XML Schema Candidate Recommendation, 24-October-2000. Links to the original document are provided for each element (labeled as 'ref' after each element name)..."

- [December 01, 2000] "Reference: W3C XML Schema Datatypes Reference." By Rick Jelliffe. From XML.com (November 29, 2000). ['A brief primer on the essential aspects of the W3C XML Schema Datatypes, including a diagrammatic reference to the XML Schemas Datatypes specification.'] "This quick reference helps you easily locate the definition of datatypes in the XML Schema specification. A 'What You Need To Know' section gives a brief introduction to the way datatypes work... W3C XML Schema specification defines many different built-in datatypes. These datatypes can be used to constrain the values of attributes or elements which contain only simple content. These datatypes are not available for constraining data in mixed content. All simple datatypes are derived from their base type by restricting the values allowed in their lexical spaces or their value spaces. Every datatype has a set of facets that characterize the properties of the datatype. For example, the length of a string or the encoding of a binary type (*i.e.*, whether hex encoding or base64). By restricting some of the many facets, a new datatype can be derived. There are three varieties of datatypes that you can use when deriving your own datatypes: as well as atomic datatypes, where the data contains a single value, you can derive a list, where the data is treated as a whitespace-separated list of tokens, and a union type, where the lexical value of the data determines which of the base types is used... There is no [current] provision for (1) overriding facets in the instance document, (2) creating quantity/unit pairs, (3) declaring n>1 dimensional arrays of tokens, (4) specifying inheritance effects, (5) declaring complex constraints where the value of some other information item in the instance (*e.g.*, an attribute) has an effect on the current datatype."
- [December 08, 2000] "Talks: XML 2000 Focuses on Schemas." By Eric van der Vlist. From XML.com. December 06, 2000. ['Reports from the first afternoon of the "XML Leading Edge" track from XML 2000, which was dedicated to the W3C XML Schema Definition Language.'] "XML 2000 dedicated the first afternoon of its 'XML Leading Edge' track to W3C XML Schema. The sessions highlighted XML Schema's application for

validating documents, showed its extensibility, and presented applications that separate logic and presentation from the structure of the document. The first presentation was a rapid overview of the specification, currently a Candidate Recommendation, by Michael C. Sperberg-McQueen, co-chair of the W3C XML Schema Working Group. Sperberg-McQueen began with an introduction in which he explained that error detection, even at a purely syntactic level, may be very beneficial by showing flaws in the expression of what a programmer writes... Matthew Fuchs, from Commerce One, in his presentation entitled 'The Role of an Extensible, Polymorphic Schema Language for Electronic Commerce Communities', talked about the possibilities created by the object-oriented features of W3C XML Schema for defining the extensible vocabularies needed in global marketplaces... Lee Buck, from TIBCO, presented the Schema Adjunct Framework, an initiative to define a common vocabulary to extend W3C XML Schema for different purposes, such as database mappings or business rules validation... Matthew Gertner, from Schemantix, went further down the extensibility path by showing how schema-based development might be 'A New Paradigm for Web Applications'. He began by saying that rich data types and inheritance are the features that categorize modern computing, going so far as to present W3C XML Schema and its extensions as a 'Universal Data Model' that can be used to define database mappings and to generate the classes of an application."

- [October 16, 2000] XML Schema: A W3C Recommendation?" By Michael Classen. From Webreference.com. October 16, 2000. Now that the XML Schema specification is one step away from becoming a W3C Recommendation, it is a good time to take a closer look at the new improved way to declare document type definitions. As mentioned in [XML] column10, DTDs have a number of limitations: (1) The syntax of a DTD is different from XML, requiring the document writer to learn yet another notation, and the software to have yet another parser; (2) There is no way to specify datatypes and data formats that could be used to automatically map from and to programming languages; (3) There is not a set of well-known basic elements to choose from. DTDs were inherited by XML from its predecessor SGML, and were a good way to get XML started off quickly and give SGML people something familiar to work with. Nevertheless it became soon apparent that a more expressive solution that itself uses XML was needed... XML Schema offers a rich and flexible mechanism for defining XML vocabularies. It promises the next level of

interoperability by describing meta-information about XML in XML. Various tools for validating and editing schemas are available from the Apache Project and IBM alphaworks."

- [December 05, 2000] "TIBCO Software Launches First Infrastructure Software Product for Managing XML Assets. XML Canon/Developer Delivers First Collaborative, Internet-Enabled, XML Schema And DTD Repository." - "TIBCO Software Inc., a leading provider of real-time infrastructure software for e-business, today announced XML Canon/Developer (XCD). XCD, the first member of the XML Canon family, is a comprehensive development platform that enables the life cycle management of XML-based business rules in a Web-accessible repository. XCD allows businesses to dynamically create and adapt XML-based standards that facilitate collaboration for e-business internally and with trading partners, customers and industry groups. The XML Canon product line supports the diverse requirements and various stages of XML development organizations are implementing. First to be launched is XCD for organizations developing and deploying XML infrastructure assets (*e.g*,. XML schemas, DTDs, instance documents, stylesheets, or adjuncts.) Features: (1) Managing Business Rules and Taxonomies: Access and manage the inter-relationships of an organization's e-business rules or XML schemas at a granular level. (2) Web-enabled Collaboration: internal and external exchange and collaboration of e-business rules based on customizable access control. (3) Life Cycle Management: configurable staging for all XML infrastructure assets, including DTDs, XML schemas, adjuncts, instance documents or stylesheets. (4) Extensible Architecture: extend and integrate with existing applications without having to re-architect them through the processing technology of XCD. (5) Full integration: across TIBCO Extensibility product family: Turbo XML, XML Authority, XML Instance, and XML Console. XML Canon/Developer will run on Windows 2000, NT 4, SP6 with support for SQL 7.0 SP1 or Oracle 8i. Microsoft IE 5.0 or 5.5 is also required. TIBCO's Extensibility client interface Turbo XML is also required."
- [December 04, 2000] IBM Regex for Java. "Regex for Java is a powerful, high-performance regular expression library. Regex for Java supports almost all features of Perl5's regular expression. It also supports the syntax of XML Schema's regular expression. Runs on all Java platforms, JDK/JRE 1.1, or later."
- [June 15, 2000] Extensibility has announced the release of XML Instance, a 'Breakthrough Schema Driven Data Editor. XML Instance is a

schema-driven XML business document editor which provides real-time validation and editing facilities against an XML schema or DTD. XML Instance is the ideal platform for the creation of XML business documents, messages, and configuration files for use in XML-based applications. Organizations can embed their XML-based business rules in an XML Instance document so that internal, trading partner, and industry standards are achieved. XML business documents can be generated and edited conforming to DTDs or schemas in major and emerging XML schema dialects including, XDR, SOX v.2 and a sub-set of XSDL (April 7) processors, bridging diverse e-business environments. . . When opening an existing document with a schema reference, XML Instance automatically locates and loads the schema, producing a template which facilitates fast and accurate document editing. When creating a new document, a schema can be set to create a fresh template. XML Instance supports all major and emerging schema dialects. The support of these dialects creates flexibility when exchanging or receiving XML business documents. XML Instance provides thorough document-building guidance based upon the rules of the schema. Real-time validation facilities ensure accurate data representation and promote seamless and accurate data interchange with your trading partners and industry groups. XML Instance is now available for download.

- [December 08, 2000] "Schemantix (formerly Praxis) to Launch Schemantix Development Platform (SxDP) at XML 2000. Innovative New-Generation Web Platform Makes Its Debut At Premier XML Conference." - "Schemantix today [2000-11-29] announced that it will be launching Schemantix Development Platform (SxDP), the world's first XML schema-driven web development platform, at XML 2000 in Washington, DC. SxDP takes advantage of the innate power of XML to provide a next-generation platform for developing and deploying powerful applications on the Web. Traditional template-based Web development languages (such as PHP, Allaire ColdFusion and Microsoft ASP) have reached their limits, and developers are actively seeking more flexible and robust solutions. Companies using existing systems are particularly frustrated by the overwhelming task of maintaining large-scale applications using hundreds or even thousands of templates. Moreover, since these systems mix application logic and presentation together in each template, it is extremely difficult to reuse business logic in new applications that do not share the same visual design. In contrast, SxDP uses XML schemas to model application data structures, and

XSLT stylesheets to produce the desired application look-and-feel. In this way, modifications to the application can be made centrally by adapting the XML schemas appropriately. Pages using the schemas are updated automatically, significantly reducing the effort needed to maintain large applications. In addition, application logic is cleanly separated from appearance and can be associated with a different presentation simply by changing the stylesheet used to produce the final formatting. Preliminary versions of SxDP have already met with favorable response from industry-leading technology companies. 'Commerce One has performed a detailed technical evaluation of Schemantix Development Platform (SxDP). We are impressed by both the vision behind the platform and its excellent technical implementation.' says Mudita Jain, SxDP is being made available as Open Source software with BSD-style license similar to that used by the Apache Software Foundation, a leading Open Source group. Schemantix plans to develop commercial products on top of the platform, with the first product slated for release in Q3 2001."

- [August 08, 2000] Envision XML. "Widely known as the manufacturer and vendor of System Architect 2001 the enterprise modeling tool, Popkin Software is poised to launch its attack on the e-business tools market with envision XML, a graphical based XML schema generator and reverser. envision XML will allow users to more easily implement and manage the XML schema, critical for successful implementation of B2B applications and other Web based information technologies including the second generation Web. At the core of envision XML is a data dictionary that stores the definitions used in the schema and allows reuse of these items across the organization. Above the dictionary is the graphical editor providing drag-and-drop facilities to immensely improve productivity, accuracy and standardization of schema development. The reverse engineering facility will let users easily incorporate existing schema into the dictionary to create their own schema management environment." See the announcement of 2000-08-04 [cache].
- [September 12, 2000] Matthew Gertner (CTO, Praxis) posted an announcement for Schemantix version 0.3. "Schemantix is our contribution to the frequently recurring discussion about where XML, and XML schemas in particular, are actually useful. In essence, it is a Open Source system for developing web application using XML schemas as the core representation of application data structures. This provides a single point of maintenance for these applications and thus

solves many of the problems associated with large-scale web applications written in template-based languages like ASP, JSP, PHP and ColdFusion. For much more on Schemantix, see www.schemantix.com. The current version is an alpha release that includes the functionality for generating HTML forms from XML schemas. The only schema language we currently support is SOX, but we will have preliminary XSDL and DTD support integrated over the next couple of weeks, as well as support for generating reports as well as forms. I'll make a followup announcement as new features become available. The entire system is available in full source code compliant with the J2EE platform. We'd be most interested in any feedback you might have, both with regard to the overall philosophy of the system and the specific implementation. This is an Open Source project, so if anyone would like to find out more about contributing, please contact me directly." Background: "As browser-hosted applications become increasingly complex and sophisticated, popular approaches to web development such as Microsoft Active Server Pages (ASP) and its open-source competitor PHP are reaching their limits. When underlying data structures are changed, each individual template must be checked and modified accordingly -- a maintenance nightmare for larger applications. Schemantix addresses these issues by moving application logic from the individual templates and back-end data sources into a single central location: XML schemas. XML schemas add powerful new facilities supporting object-oriented features such as inheritance, polymorphism and rich datatyping. As such, they represent an ideal repository for storing business and presentation logic that can be reused across an entire web application, from the browser-hosted user interface to the backend data storage engine."

- [September 23, 2000] "XML Schemas: Best Practices." By Roger L. Costello *et al.* (1) Hiding (Localizing) Namespace Complexities within the Schema (2) Namespaces: Expose them or Not? Purpose: collectively come up with a set of 'best practices' in designing XML Schemas. The specifics of designing a schema are dependent upon the task at hand. The goal of this effort is to come up with a set of schema design guidelines that hold true irrespective of the specific task. Below are some of the things that must be considered in designing a schema. It is by no means an exhaustive list. For example, it doesn't address when to block a type from derivation, when to create a schema without a namespace, when to make an element or a type abstract, etc. Nonetheless, it is a start to some hopefully useful discussions. First, a quick list of the issues: (1)

Element versus Type Reuse; (2) Local versus Global; (3) elementFormDefault - to qualify or not to qualify; (4) Evolvability/versioning; (5) One namespace versus many namespaces (import verus include); (6) Capturing semantics of elements and types ...

- [June 29, 2000] XML Schema and XML DTDs - will XML Schemas replace XML DTDs? [Opinion by] Rick Jelliffe as of 2000-06-29. XML-DEV post. "For back-end data interchange, especially from a DBMS, you can expect XML Schemas to replace DTDs. They provide better datatyping and they are designed to fit in conveniently with RDBMS and OO languages such as Java. For data that is closer to human users or file-systems, XML Schemas cannot entirely replace DTDs yet. . ."
- [September 29, 2000] "The Beginning of the Endgame. A Look at the Changes in the Pre-CR W3C XML Schemas Draft." By Rick Jelliffe. From XML.com. September 27, 2000. ['The W3C's XML Schemas technology, vital to the use of XML in e-business, is finally nearing completion. This article catalogs the most significant changes from the recent draft specs,and highlights areas where priority feedback is required from implementors and users.'] "This article looks at those changes in the recent Pre-CR draft of W3C XML Schemas that will most effect developers and users. Requirements for data interchange with database systems have been important during W3C XML Schema's development. The recent changes also support markup languages and schema construction better. The Candidate Recommendation (CR) drafts are slated to appear hot on the heels of the current drafts. The XML Schema Working Group was aware that authors, implementers, schema writers, and technical evaluators needed to know the most recent changes, especially since they include some syntax changes that will affect schemas using type derivation."
- [August 30, 2000] "The basics of using XML Schema to define elements. Get started using XML Schema instead of DTDs for defining the structure of XML documents." By Ashvin Radiya and Vibha Dixit (AvantSoft, Inc.). From IBM DeveloperWorks. August 2000. ['The new XML Schema system, now nearing acceptance as a W3C recommendation, aims to provide a rich grammatical structure for XML documents that overcomes the limitations of the DTD. This article demonstrates the flexibility of schemas and shows how to define the most fundamental building block of XML documents -- the element -- in the XML Schema system.'] " We have covered the most fundamental concepts needed to define elements in XML Schema, giving you a flavor of its power through simple examples. Many more powerful

mechanisms are available: (1) XML Schema includes extensive support for type inheritance, enabling the reuse of previously defined structures. Using what are called facets, you can derive new types that represent a smaller subset of values of some other types, for example, to define a subset by enumeration, range, or pattern matching. In the example for this article, ProductCode type was defined using pattern facet. A subtype can also add more element and attribute declarations to the base type. (2) Several mechanisms can control whether a subtype can be defined at all or whether a subtype can be substituted in a specific document. For example, it is possible to express that InvoiceType (type of Invoice number) cannot be subtyped, that is, no one can define a new version of InvoiceType. You can also express that, in a particular context, no subtype of ProductCode type can be substituted. (3) Besides subtyping, it is possible to define equivalence types such that the value of one type can be replaced by another type. (4) By declaring an element or type to be abstract, XML Schema provides a mechanism to force substitution for it. (5) For convenience, groups of attributes and elements can be defined and named. That makes reuse possible by subsequently referring to the groups. (6) XML Schema provides three elements -- appInfo, documentation, and annotation -- for annotating schemas for both human readers (documentation) and applications (appInfo). (7) You can express uniqueness constraints based on certain attributes of child elements. . ." Also available in PDF format; [cache]

- [July 21, 2000] XML Schema Conformance at XMLConf. "XML Schema, a W3C work-in-progress, is a very complex specification. The development of test suites and harnesses in this project is intended to help identify underspecified behavior in the spec and to advance the development of conformant processors. Since the specification is subject to change, the test cases in the suite may need to be modified for each release. The initial development thrusts are the development of test cases for specific constraints on schemas and the enhancement of the Java XML conformance harness for use in testing of Java parsers. The suite may be downloaded either as part of the nightly CVS snapshot or by checking out the schema module from the CVS. The development of a designed conformance suite should complement the sample schemas collected by the XML Schema Validation Service (XSV). The SourceForge XMLConf project hosts XML related testing efforts, focusing initially on conformance testing. Notice that all of this software is under the GPL. The first testing effort hosted here addresses XML conformance. It

includes test harnesses for Java (with SAX/SAX2) and for JavaScript (with DOM/COM). The second such effort is currently in its early stages, and addresses XML Schema conformance. Other projects discussed include DOM testing, performance measurement, XSLT conformance ... the whole gamut. Basically, if it's an XML related technology and there's enough of a standard API that an automated harness could usefully compare different implementations, it could fit in here. The intention here is provide a home for open, public, collaborative development of harnesses and test cases for testing XML (and related) processors. It complements the corresponding efforts of W3C, NIST, OASIS, and many others..."

- [October 12, 2000] Noted: An OASIS mailing list 'xmlschema-conf' and corresponding XML Schema Conformance Committee. For related references, see "XML Conformance".
- [September 14, 2000] "Beyond Schemas." By Scott Vorthmann (Extensibility, Inc.) and Jonathan Robie (Software AG). Paper presented at the Extreme Markup Languages 2000 Conference (August 13 - 18, 2000, Montréal, Canada). Published as pages 249-255 (with 3 references) in *Conference Proceedings: Extreme Markup Languages 2000. 'The Expanding XML/SGML Universe'*, edited by Steven R. Newcomb, B. Tommie Usdin, Deborah A. Lapeyre, and C. M. Sperberg-McQueen. "The Schema Adjunct Framework is an XML-based language used to associate task-specific metadata with schemas and their instances, effectively extending the power of existing XML schema languages such as DTDs or XML Schema. This is useful because in many environments additional information which is typically not available in the schema itself is needed to process XML documents. Such information includes mappings to relational databases, indexing parameters for native XML databases, business rules for additional validation, internationalization and localization parameters, or parameters used for presentation and input forms. Some of this information is used for domain-specific validation, some to provide information for domain-specific processing. No schema language provides support for all the information that might be provided at this level, nor should it -- instead, we suggest a way to associate such information with a schema without affecting the underlying schema language." See the Schema Adjunct Framework overview in the "Schema Adjunct Framework Developer's Guide" and specification document from Extensibility. See also the article in *MLTP 2/3*.
- [August 28, 2000] "Comparative Analysis of Six XML Schema Languages." By Dongwon Lee and

Wesley W. Chu (Department of Computer Science University of California, Los Angeles Los Angeles, CA 90095, USA Email: {dongwon,wwc} @cs.ucla.edu). UCLA CS-TR 200008 (Technical Report). Also published in *ACM SIGMOD Record* Volume 29, Number 3 (September, 2000). " Abstract: As XML is emerging as the data format of the internet era, there is an substantial increase of the amount of data in XML format. To better describe such XML data structures and constraints, several XML schema languages have been proposed. In this paper, we present a comparative analysis of the six noteworthy XML schema languages. As of June 2000, there are about a dozen of XML schema languages that have been proposed. Among those, in this paper, we choose six schema languages (XML DTD, XML Schema, XDR, SOX, Schematron, DSD) as representatives. Our rationale in choosing the representatives is as follows: (1) they are backed by substantial organizations so that their chances of survival is high (*e.g.*, XML DTD and XML Schema by W3C, XDR by Microsoft, DSD by AT&T), (2) there are publically known usages or applications (*e.g.*, XML DTD in XML, XDR in BizTalk, SOX in xCBL), (3) the language has a unique approach distinct from XML DTD (*e.g.*, SOX, Schematron, DSD)..." The document is also available in Postscript and HTML formats. [cache]

- [August 11, 2000] "XML Schema Languages: Beyond DTD." By Demetrios Ioannides (Michigan State University, East Lansing, MI). In *Library Hi Tech* Volume 18, Number 1 (2000), pages 9-14 (with 6 tables, 14 references). [ISSN: 0737-8831.] Abstract: "The flexibility and extensibility of XML have largely contributed to its wide acceptance beyond the traditional realm of SGML. Yet, there is still one more obstacle to be overcome before XML is able to become the evangelized universal data/document format. The obstacle is posed by the limitations of the legacy standard for constraining the contents of an XML document. The traditionally used DTD (document type definition) format does not lend itself to be used in the wide variety of applications XML is capable of handling. The World Wide Web Consortium (W3C) has charged the XML schema working group with the task of developing a schema language to replace DTD. This XML schema language is evolving based on early drafts of XML schema languages. Each one of these early efforts adopted a slightly different approach, but all of them were moving in the same direction. . . The XML new schema is not only an attempt to simplify existing schemas. It is an effort to create a language capable of defining the set of constraints of any possible data resource. Table VI XML schema goals simply illustrates the XML schema goals. The importance of having a fully

fledged and universally accepted schema language is paramount. Without it, no serious migration from legacy data structures to XML will be possible. Databases with unwieldy data structures such as MARC will greatly benefit from such a migration. We will no longer depend on data structures that were predefined years ago to meet different needs. The extensible nature of these schema languages will allow the easy creation of any data structure, thus providing the flexibility mandated by the mutability of today's information needs. Bringing a wealth of metadata into an extensible format and allowing it to take full advantage of the dynamic nature of networking is an extremely exciting prospect for information professionals. The first step, premature as it may be, yet very symbolic, is the CORC Project's creation of a DTD for MARC. In the future, intelligent schemas will allow for blending of existing metadata with full-text, multimedia, and much more. The possibilities are endless..."

- "Using Regular Expressions - XML Schema Style."
- [June 29, 2000] "In the Grand Schema of XML." By Yasser Shohoud (devxpert Corporation). In *XML Magazine* Volume 1, Number 3 (Summer 2000), pages 38-43. ['If you are designing an Internet, intranet, or client-server application, you should be thinking about XML schemas. XML Schema Language is a powerful feature that can be used to validate data in myriad ways and save you time in the process. Yasser Shohoud shows you how.'] "XML Schema Language is likely to become a standard in the near future. Although at the time of this writing, no validating parsers support the current version of the schema language, at least one parser, XML Authority from Extensibility, supports an earlier version of the language. XML Authority version 1.2 is planned for release soon and should support a more current version of the XML Schema Language specification. Also, the MSXML parser supports Microsoft's version of the schema language known as XML-data. Microsoft has plans to support the XML Schema Language when the W3C finalizes it. XML documents can be thought of as containers for data; they are similar to tables in a relational database and objects in an object-oriented language. In relational databases, the Data Definition Language is used to define new data types and create tables using those types, while specifying rules and constraints on columns in those tables. You can then insert data into the tables, and the database will ensure that your rules and constraints are enforced. Object-oriented programming languages let you define classes that have properties whose type may be one of the intrinsic data types or another class. You can then instantiate objects from those

classes and set the values of their properties. The run-time type checking system will ensure that property values are of the correct type according to the class definition. An XML document contains one or more elements containing attributes, other elements, and text. As with database tables and object-oriented classes, you need to define rules about the structure, permissible data types, and constraints that apply to each element within the document. These rules would be equivalent to a class definition in Java. An XML document can then be considered an instance of this class definition and therefore be validated against the definition at runtime. Using the XML Schema Language, document authors can define the structure and permissible data types within their documents. Validating parsers can then be used to check conformance of documents claiming to be instances of a given schema. Going back to our database analogy, XML Schema Language is like a DDL for databases, a specific XML schema is like a specific table definition in a database, and an XML instance document is like a record in a database table. The object-oriented language analogy may be obvious by now: XML Schema Language is like the language you use to define a class (for example, Java or C++); a specific XML schema is like a class with properties, and the XML instance document is like an object instance of that class. The object-oriented analogy applies only to class properties, since XML elements are data containers with no methods of their own. . ."

- [June 22, 2000] "Composite datatypes - XML Schema datatype for date+time." By Rick Jelliffe. "There are [currently, 2000-06-29] no composite datatypes in XML Schemas..."
- [July 14, 2000] "The results of my W3C XML Schema Questionnaire are now available at http://metalab.unc.edu/xql/tally.html." From Jonathan Robie, Fri, 14 Jul 2000 16:29:35 -0500
- [June 22, 2000] Extensible Types (eTypes) - From IBM alphaWorks. 'A Java component library which enables users to specify properties and determine whether objects satisfy these properties.' "Extensible Types (eTypes) is a Java component library that enables users to specify constraints and determine whether objects satisfy these particular constraints. With our first preview release we provide a library built on eTypes that can validate many of the datatypes defined by W3C XML Schema April 7th, 2000 working draft as well as several well know ISO datatypes. An important application of eTypes is the so-called automatic inference of datatypes from XML document instances. Indeed, the eType distribution includes a command line tool for deducing XML Schema text-only types from a set of XML documents or sets of example strings.

With eTypes, XML developers will get a head start in writing XML Schemas that constrain the data allowed 'in between' XML tags and attributes..."

- [January 13, 2001] XML Spy 3.5. An announcement for the evaluation version of XML Spy 3.5 describes the most recent added features of the XML Spy editing tool. "XML Spy is centered around a professional validating XML editor that provides four advanced views on your documents: an Enhanced Grid View for structured editing, a Database/Table view that shows repeated elements in a tabular fashion, a Text View with syntax-coloring for low-level work, and an integrated Browser View that supports both CSS and XSL style-sheets." From the announcement: "Version 3.5 Beta 4 is the last public beta release for our upcoming XML Spy 3.5 product, and includes several new features. Specifically this beta release contains/supports: (1) validation of XML Schemas with integrated error highlighting directly within the Schema design view; (2) improved validation of XML instance documents based on XML Schemas -- it consumes less memory and is much faster; (3) access and manipulate files in any respository that is accessible through an `ftp:`, `http:`, or `https:` URL; (4) browse and manipulate folders directly from the Open/Save URL dialog on any FTP or WebDAV server... [etc.] We are now also offering a quick introduction to the new XML Schema Design View online. If you are interested in working with XML Schema, please visit this URL, which explains the new XML Schema related features in detail: http://www.xmlspy.com/features_schema35.html... XML Spy 3.5 includes a new schema design menu that is available whenever an XML Schema document is opened and displayed in the XML Schema Design View. When you open an XML Schema document, XML Spy displays all globally defined particles (*i.e.*, elements, complexTypes, attributeGroups, etc.) in the XML Schema as a list in the XML Schema Design View. If an element, complexType, or attributeGroup is selected, the corresponding attributes are automatically shown in the list underneath the globals. For each particle that has the little tree symbol next to it, you can click on that symbol to open the content model for that particle in the advanced tree view of XML Spy. To edit the content model, simply use drag&drop to rearrange elements or use the right mouse button for other manipulations. To return to the global view, please use the menu command 'Display All Globals'. To navigate to related elements or types, you can also double-click on the name of any complexType (shown as a rectangle with yellow background) or Ctrl-double-click on any element in the tree view. In addition, the following floating XML Schema Navigator window is always visible

and lets you switch to different particles by simply double-clicking them in the list. At the same time, elements can be easily added to the content model, by dragging them from the XML Schema Navigator window onto the desired position in the content model. While most parameters of an element node (such as its name, type, and major facets) can be edited directly in the tree view, the full details of the selected node are always visible (and can be edited) in the detail views in separate floating windows. In addition to offering these flexible editing capabilities, the advanced Schema Design View of XML Spy is also highly configurable and lets the user choose what parameters should be displayed and how the display should be formatted..."

- [May 02, 2000] XML Spy 3.0 Beta 3 Supports XML Schema. Alexander Falk (Icon Information-Systems) recently announced the availability of XML Spy 3.0 for Windows, including new support for XML Schema/DTD editing and validation. "It is my pleasure to announce the last scheduled beta release of XML Spy 3.0 for Windows, which is no longer only an XML editor, but has matured into a true Integrated Development Environment (IDE) for XML that includes: (1) XML editing and validation, (2) Schema/DTD editing and validation, (3) XSL editing and transformation. XML Spy 3.0b3 contains our new incremental validating parser which fully supports Document Type Definitions (DTD), Document Content Description (DCD), XML-Data Reduced (XDR), BizTalk, and already contains support for most of the new April 7 W3C XML Schemas, which makes XML Spy the first editor that supports the new XML Schema draft from editing through schema validation to intelligent editing of XML instance documents based on the schema. The XML Schema support includes: (1) simpleType & complexType; (2) element & attribute; (3) group, sequence, choice, any; (4) all datatype facets, including user-defined patterns; (5) notation, annotation, documentation, include. [...] The editing tool provides four advanced views on your documents: an Enhanced Grid View for strucutred editing, a Database/Table view that shows repeated elements in a tabular fashion, a Text View with syntax-coloring for low-level work, and an integrated Browser View that supports both CSS and XSL style-sheets." See the web site for a detailed description of new features in XML Spy 3.0. The editor is available for download and 30-day evaluation.
- [June 10, 2000] "Use XML Even As It Changes. Here's how you can tackle application-to-application integration needs while building a migration path to XML Schema." By James Bean. In *Enterprise Development*. February, 2000. "The

current XML specification uses Document Type Definitions (DTDs) to describe the content and structure of an XML document. However, there's an innovation waiting in the wings: XML Schema. Schema will most likely present the best solution for describing metadata with XML. But current implementations are often based on DTDs. Schema should be adopted rather rapidly, but a number of industry-based XML vocabularies and numerous custom-developed XML DTDs will require a reasonable period of migration. This article will show you how to address your short-term A2A integration needs with XML DTDs and how to build in a handy migration path to Schema, then preview how Schema will most likely work..."

- [May 03, 2000] "The Schema Adjunct Framework." By Scott Vorthmann and Lee Buck (Extensibility, Inc.). Draft Specification 24-February-2000. ['Introducing Extensibility's Schema Adjunct Framework, an open standard for describing and utilizingschema-level information within processing environments.'] "Software applications that process XML often need to associate additional information with documents beyond the structures and properties that can be expressed in a schema language. For example, they may need to specify how XML structures are mapped into object-based or relational systems, provide business logic associated with structures, state how structures should be formatted, or state additional constraints not expressible in the schema language. If most applications needed the same set of relatively simple extensions, these extensions should be integrated into the schema language itself. In practice, the extensions needed by various systems differ widely, and they may need to be specified in many different ways, including XML data, procedural program code, or query statements. Any schema language that attempted to support the whole range of possible extensions would quickly become unwieldy. A more tractable approach is to provide a general framework that allows users to specify additional information about the structures or properties that the schema defines. For instance, an application that generates HTML forms from XML schemas must associate labels and controls with various elements specified in the schema. Naturally, no schema language supports such HTML-specific statements. To fill such needs, the Schema Adjunct Framework introduces the concept of a schema adjunct, an XML document that contains additional, application-specific data relative to a particular schema. The additional data may be stated in any language that can be placed in an XML document, including query languages, Java, JavaScript, XML-based languages, or prose. A schema adjunct provides the information that

enables the use of a schema (and its instances) within a particular application. This means that a given schema can be enabled in a family of interoperating applications by an equal number of adjuncts. Conversely, a given application can be applied to a variety of schemas by supplying an adjunct for each schema." See also the Executive Summary, the Developer's Guide, and the SDK in Java.

- [April 19, 2000] Curt Arnold has announced the publication of an "HTMLHelp file for the 7-April-2000 XML Schema Working Draft, available at the AEA Technology Web site. This help file has been generated from the 'schema for schemas' appearing in the 7-April-2000 XML Schema W3C Working Draft. There is minimal narration in this help file; however it should be useful for a quick reference and a roadmap to the concrete schema language. The help file does not attempt to document the context-specific variants of elements (for example, the element may not have a name attribute when it is not an immediate child of a element). If you like excessive narration, still available is the help file to an alternative December 17th working [XML Schema] draft where I tried to discuss some of the issues that I had with that draft. Some of the issues have been reflected in later drafts, others have been deferred to later, however some still seem like pressing issues to me, specifically complexType derivation by restriction. Please send any comments on this help file to xml@software.aeat.com."
- [April 21, 2000] XML Schema Compiler. Curt Arnold recently announced a 'schema compiler' effort, supported by a developers' mailing list. The XML Schema compiler project has been created to build reference implementations of schema evaluation and simplification in XSLT. The public is "invited to join a project to an open-source XSLT-based compiler for XML Schema. The XML Schema "compiler" project intends to provide reference implementations (and potentially other) of schema processing to: (1) produce analysis of errors in the source schema; (2) produce a compiled form of the schema that contains the expansion of inclusions, imports, complexType derivations, etc, in a form that closely related to the information set necessary for a parser to validate a conforming document. The compiled form should help reduce the potential for hacking by attacking resources used in the schema definition. (3) support transliteration (as much as possible) of XML schema constraints to other validation mechanisms such as Schematron or RELAX; (4) support generation of documentation for schemas; (5) provide a forum to discuss schema related issues; (6) provide feedback to the W3C XML Schema working group; and (7)

assist the development of schema-aware (and/or compiled schema-aware) parsers." Project XSDComp is accessible on SourceForge. Note: "I've packaged up my current work as the 0.0.1 version of XML Schema compiler and made it available on the project home page at http://sourceforge.net/project/?group_id=4826; it is definitely a work in progress and has known flaws in addition to unimplemented features. However, it is useful as a validator for XML Schema and a general indication of where the project is going..." [2000-05-01] And Curt Arnold wrote 25-August-2000: ["The XML Schema 'compiler' project intends to provide a reference implementations of schema evaluation and simplification in XSLT."] You might want to check out my XML Schema Compilation project at SourceForge (http://sourceforge.net/projects/XSDComp) which uses XSLT to convert XML Schema to a consise, fully evaluated form (all includes/imports brought into one file, all references converted to ID/IDREF pairs, etc). I've definitely advanced my working copy beyond what I've put in the CVS. If you are interested after taking a look around, I can update it." See the download, [cache]

- [March 17, 2000] "XSDL - A Next Generation Schema Language to replace DTDs. [XSDL Presentation.]" By Dr. Matthew Fuchs, CommerceOne. Presented at XML SIG (The Center for Information, Connection, and Education). February 24, 2000. "XSDL, A Whirlwind Tour" is available as a set of PowerPoint slides. "For over a year, the W3C's XML Schema Working Group has been developing XSDL, a next-generation schema language to replace DTDs. The goal has been a language to support the requirements of a whole range of applications beyond documents; active participants have included Oracle, Microsoft, HP, Sun, and CommerceOne, among others. The draft Schema language has several features that propel it far beyond DTDs as a means of describing information. It includes both object-oriented extensions for element types and strong datatyping for attribute values and string content. It has strong integration with namespace and a powerful composition mechanism to allow definitions from multiple schemas to work together. This talk will describe the important features of this language, due to become a W3C Candidate Recommendation in March (we will also describe what a "Candidate Recommendation" is), and describe potential applications, such as improved XML/Java integration. While XSDL implementations are not yet available, almost all the significant features are already available in CommerceOne's SOX

(Schema for Object-oriented XML), which was one of the inputs to the XSDL process. Participants who are interested in a preview of the future of XML schema languages can download and use our publicly-available SOX parser http://www.marketsite.net/xml/xdk." [From Earl Bingham: "I just heard a presentation last night from Mathew Fuchs from CommerceOne who is on the W3 board for XML Schemas. He gave a great presentation on XSDL and how it is evolving. If anyone is interested I can send out the powerpoint slides and Ialso made a video of the presentation that I can make copies for anyone who wantsit. It also has a great demo of XML Authority software and how this can be used with some of the latest standards."

- "UML for XML Schema Mapping Specification." By Grady Booch (Rational Software Corp.), Magnus Christerson (Rational Software Corp.), Matthew Fuchs (CommerceOne Inc.), and Jari Koistinen (CommerceOne Inc.). 12/08/99. "This paper describes a graphical notation in UML for designing XML Schemas. UML (Unified Modeling Language) is a standard object-oriented design language that has gained virtually global acceptance among both tool vendors as well as software developers. UML has been standardized by the Object Management Group (OMG). XML Schema is an emerging standard from W3C. XML Schema is a language for defining the structure of XML document instances that belong to a specific document type. XML Schema can be seen as replacing the XML DTD syntax. XML Schema provides strong data typing, modularization and reuse mechanisms not available in XML DTDs. There is currently no W3C recommendation for XML Schema, although several have been proposed and W3C is actively working on producing a recommendation. This paper describes the relationship between UML and the SOX schema used by CommerceOne. Our intention is, however, to adapt the mapping to the W3C recommendation when that becomes available. W3C discussions up to this point indicate the notation described here will be upward compatible with the eventual recommendation."
- [February 26, 2000] XML Schema for RDF. "For anyone interested, appended is a stab at an XML Schema for RDF. It shows how abstract elements and equivClass is useful for constructing frameworks..." From Rick Jelliffe.
- [August 04, 2000] XSD for RDF by Dan Connolly. Revision of an XSD sketched by Rick Jelliffe. v 1.17 2000/08/04. [cache]
- [April 21, 2000] "The Meanings of XML: DTDs, DCDs and Schemas." By Michael Classen. In *Internet.com WebReference* (April 16, 2000).

XPLORING XML - Column 10. ['Don't be scared by the schema. The syntax and semantics of XML are the sources of its strength. Join our Xpert as he reveals the secrets of XML data and structure definitions.'] "So far in this column we ignored the more formal aspects of XML, such as defining the correct syntax and semantics of specific documents. While we examined the set of rules common to all documents (remember well-formed vs. valid documents?), I have so far neglected the mechanisms for specifying your own families of documents..."

- [September 07, 2000] "Schema Round-up." By Leigh Dodds. From XML.com (September 06, 2000). "Noting an increasing interest in XML Schemas on several mailing lists, this week the XML Deviant takes a look at some of the resources available to the aspiring schema developer."
- [March 18, 2000] Mark Scardina (Oracle 'Group Product Manager and XML Evangelist') recently announced the availability of Oracle's 'XML Schema Parser', which supports the use of simple and complex datatypes in XML. "Version 0.9.0.0 of the XML Schema Processor for Java is now available on the Oracle Technology Network at http://technet.oracle.com/tech/xml/. This first release of the XML Schema Processor is a companion component to the XML Parser for Java that allows support to simple and complex datatypes into XML applications with Oracle8i. Since these components are implemented in Java, they can run 'out of the box' in the Oracle8i JServer Java VM or in any standalone Java 1.1 or greater VM. The tool supports XML documents in the following encodings: UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, US-ASCII, EBCDIC-CP-*, ISO-8859-1to -9, Shift_JIS, BIG, GB2312, EUC-JP, EUC-KR, KOI8-R, ISO-2022-JP, and ISO-2022-KR. The tool incorporates new APIs in XMLParser to invoke XML Schema validation, and new APIs to build a XMLSchema object. The following features are not implemented in this release: (1) unique, key and keyref constrains, (2) derivation by restriction from complexType, (3) pattern facet in string, datetime datatypes, (4) builtin types derived from integer [unsigned] long, short, int, byte), (5) comparison of datetime datatypes. The distribution includes sample XML applications to show how to use the Oracle XML parser with the XMLSchema processor. The Schema Processor supports much of the [2000-02-25] XML Schema Working Draft, with the goal being that it be 100% fully conformant when XML Schema becomes a W3C Recommendation. The XML Schema Processor makes writing custom applications that process XML documents straightforward in the Oracle8i

environment, and means that a standards-compliant XML Schema Processor is part of the Oracle8i platform on every operating system where Oracle8i is ported."

- [December 03, 1999] XML Schema Tutorials at the XML '99 / Markup '99 Conferences.

    - Henry S. Thompson wrote on XML-DEV, 29 Nov 1999 [Subject: XML Schema One-day intensive tutorial]: "It's not too late to sign up for the tutorial I'm giving at Markup '99/XML '99 on Sunday: a full day introduction to the up-to-the minute state of XML Schema. We'll be working from a pre-publication version of the next public working draft, due out 16 December, currently anticipated to be the last draft before the WG submits a version to Last Call. Don't miss this chance to pester one of the editors with your personal opinions about what is and isn't right about this important impending W3C recommendation." The blurb: Full-Day Tutorials. 9:00 am - 5:30 pm. **XML Schema Languages: A Technical Introduction** Instructor: Henry Thompson, University of Edinburgh. "Prerequisite skills: Technical knowledge of XML, XML DTDs, and computer grammars. XML Schema definition language proposes facilities for describing the structure and constraining the contents of XML 1.0 documents. The schema language, which is itself represented in XML 1.0, provides a superset of the capabilities found in XML 1.0 document type definitions (DTDs.) This tutorial provides a technically detailed examination of the most recent XML Schema by one of its editors. In addition, the tutorial presents an introduction to schema constraints, types, composition and symbol spaces along with terminology used throughout the specification. Part two of the tutorial discusses specifying a language for defining datatypes to be used in XML Schema."
    - Sunday, December 5, 1999. Morning Half-Day Tutorial. 9:00 am - 12:30 pm. **A Non-Technical Introduction to XML Schemas** Instructor: Murray Maloney, Muzmo Communications Inc. "An XML schema is a formal expression of the structure of an XML document and of constraints on text contained therein. XML's existing Document Type Definition can be used for this purpose. But there is a growing recognition that a DTD is inadequate or inappropriate for expressing what many of

the current and anticipated applications of XML require. This tutorial focuses on the requirement for an XML schema language and highlights the concepts of an XML schema definition. It also discusses the ways a schema language will facilitate the use of XML on the Web."
    - Also: (1) Monday December 06, 4:45 PM : "Modeling an XML Schema," by Lee Buck, Extensibility, Inc. (2) Wednesday, December 08, 9:00 AM: "XML Schema and Datatypes," by Michael Sperberg-McQueen. 'An XML schema is a mechanism somewhat analogous to DTDs for constraining document structure (order, occurrence of elements, attributes). In addition, specific goals beyond DTD functionality, such as the specification of datatypes have been identified within the scope of XML Schema. This informative session focuses on the emerging XML schema language and the proposed mechanism for specifying datatypes.'

- [January 06, 2000] XML Schema Tutorial. Updated September 05, 2000 [or later]. Roger L. Costello (Mitre) recently posted an announcement for a tutorial on the current W3C working draft specification for XML Schema. The document is presented as a set of 91 PowerPoint slides. Roger says: "With a lot of help from this list community and many hours studying the spec, I have created a tutorial on the latest draft (12-17-99) of the XML Schema specification. It is freely available at http://www.xfront.com/xml-schema.html. I personally learn best with examples, so the tutorial contains quite a few examples to demonstrate various features." [cache, 2000-09-06]
- [May 15, 2001] Mary Holstege's "Conversion Tool (XML DTDs to W3C XML Schema)" based upon Perl. With contributions by Yuichi Koike, Dan Connolly, and Bert Bos. See dtd2xsd.pl from www.mathling.com, May 15, 2001 or later. [cache]
- [April 20, 2000] See previous entry. DTD to XML Schema Conversion Tool. Based on Perl: `perl dtd2xsd.pl [-alias] [-prefix p] [-ns n] [file]`. By Bert Bos, Dan Connolly, Yuichi Koike. Said to be 'open source'. [cache]
- [November 09, 1999] "Family Tree of Schema Languages for Markup Languages" referenced from Rick Jelliffe's Schemas & XML document. [local archive copy]
- "Resource Description Framework (RDF) Schema Specification." W3C Working Draft 30-October-1998, WD-rdf-schema-19981030.
- [October 08, 1999] W3C Publishes *The*

Cambridge Communiqué on Web Data Models. The W3C has released a NOTE on Web data models under the title *The Cambridge Communiqué*. Reference: W3C NOTE 7-October-1999, edited by Ralph R. Swick and Henry S. Thompson. The note constitutes "a report of the results of a meeting of a group of W3C Members involved in XML and RDF to advance the general understanding of a unified approach to the expression of Web data models. This document is one response to the Web data architecture discussed in "Web Architecture: Describing and Exchanging Data". In detail: a group "consisting of W3C Member representatives and W3C staff involved in the XML and RDF activities met on August 26 and 27 [1999] to discuss the architectural relationship between the schema work being undertaken within these two activities. The goals of this meeting were to articulate a vision of this relationship for the Web community, to feed input into the XML Schema Working Group and other W3C activities in support of this vision, and to resolve issues raised in the Member review of the RDF Schema Proposed Recommendation concerning overlap with XML work." The document presents nine (provisional) 'Observations and Recommendations', the first of which states: "The XML data model is the XML Information Set being specified by the XML Information Set Working Group. Other data models exist, both generic and application-specific. RDF is an example of one such generic data model. The XML Schema and RDF Schema languages are separate languages based on different data models and do not need to be merged into a single comprehensive language..."

- [July 05, 1999] "Understanding XML Schemas. [Schemas for XML.]" By Norman Walsh. From XML.com. July 01, 1999. "In May, the [W3C] XML Schema Working Group (WG) published its first Working Draft (WD). Schemas will have a broad impact on the future of XML for two reasons: first because they will define what it means for an XML document to be valid and second because they are a radical departure from Document Type Definitions (DTDs), the existing schema mechanism inherited from SGML. In this article, I'll explore what schemas are, what validity means, how schemas differ from DTDs, and what new functionality will be gained from adopting them. I'll be using the XML Schemas WD from 6 May 1999 to frame the discussion and as the source for concrete examples. . . Looking at the scope and functionality that schemas will provide, they seem like a great improvement over DTDs. Certain kinds of applications, exchanging information between databases, for example, and ecommerce are clearly going to be made simpler and more

interoperable by XML Schema. As I see it, the primary virtue of DTDs today is that they are well understood and they do offer a good way to describe the structure of an document for interchange. It will take some time before XML Schema are as well understood. Until then, we'll be 'flying without a net' to a certain extent, waiting for the final standard and practical, documented methodologies for schema creation to follow." See 'XML Schemas' from the W3C: XML Schema Part 1: Structures and XML Schema Part 2: Datatypes.

- [December 20, 1999] "Writing a data type-checking XML parser with Xerces." By Bob DuCharme. In IBM Developer Library. December 1999. "While most XML parser developers are waiting for the W3C Schema Working Group's proposal to become a Recommendation before they support it, the Xerces parser donated by IBM to the Apache XML project already supports much of the Working Group's September 1999 Working Draft. In particular, it supports basic data-type checking, one of the most eagerly awaited W3C Schema features. In this article, see how your XML Java applications can take advantage of data-type checking when using the Xerces parser... The first parser I know of that provided any support for the W3C Schema Working Group's XML Schema Definition Language (XSDL) was alphaWorks' xml4j-ea1, a special version of the xml4j XML Parser for Java Early Access release (see Resources). It included support for a subset of XSDL that was backward-compatible with XML 1.0 DTDs. In other words, if you rewrote an XML 1.0 DTD as an XSDL schema that didn't take advantage of any of XSDL's new features, an application using the xml4j-ea1 parser could validate a document against that schema. With xml4j-ea2 and the Xerces parser that IBM donated to the Apache project, the IBM XML Technology Group developers added the feature that developers were clamoring the loudest for: type checking. Because the Working Group still has some issues to work out in Part 2 ("Datatypes") of the XSDL Proposal, the Xerces parser doesn't yet support all the data types mentioned in the proposal. But real numbers, integers, booleans, and of course strings are supported, and your applications can take advantage of this support now."
- [September 17, 1999] "UML as a Schema Language for XML based Data Interchange." By David Skogan (Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, N-0316 OSLO, NORWAY). WWW. Paper submitted to UML'99. With 26 references. 1999-05-14. Abstract: "The Unified Modeling Language (UML) is here used as a schema language to define data interchange formats based on the Extensible

Markup Language (XML). UML is a powerful and flexible modeling language and XML is expected to be the next generation data interchange format for the Web. UML's declarative expressiveness and intuitive visual form overcome XML's current declarative powers. The use of UML as a schema language combined with XML as a data representation language addresses both semantic and syntactic interoperability. A mapping from UML to XML is defined and two prototype implementations are presented. The mapping is inspired by Object Management Group's (OMG) XML Metadata Interchange specification (XMI). It is developed as a part of the ongoing standardization work creating an international standard for geographic information (ISO 15046). It is generic and may easily be adapted to other application domains." Cited in connection with a docment on the document Geographic Information BROWSER 1.0.2.1" - 'A prototype XML import/ export facility and browser. Encode, decode, navigate, and edit Geographical Information.' "The GI Browser has been developed in the DISGIS project and is based on ISO CD 15046-18 Geographic Information - Encoding, a standard currently being developed by ISO/TC 211." [local archive copy]

- [December 03, 1999] "Serializing Graphs of Data in XML." By Adam Bosworth, Andrew Layman, and Michael Rys (Microsoft). From the BizTalk Web site. '1999.' "XML is evolving as the standard format of exchanging data among heterogeneous, distributed computer systems and as such is used to represent data of various origins in a common format. Often, this data possesses rich structure and represents relationship among various entities. These relationships form graphs, where the relations are directed from one entity to another (and may have inverses) and where there may be multiple paths to an entity. Thus, an important goal of the encoding of this data is to preserve the exact graph structure in the serialization to XML. The aim of this paper is to describe a specific way to use XML to serialize graphs of data (such as database tables and relations or nodes and edges from directed labeled graphs) in such a way that the graph structure is preserved and can be reconstructed. A graph of data serialized according to the described rules is said to be in canonical form. Other representations of the same data can be mapped into and out of the canonical form as long as they do not lose or add information. Therefore, the canonical form provides a common basis that can be exploited for information integration across multiple sources and it can be used as a common abstraction for data interchange. This paper does not change the fact that every validatable XML

document conforms to a specific grammar. Rather, it proposes a way to mechanically generate, from a database's or graph's schema, a particular grammar that can be used to serialize data from the database or graph, and into which any other serialization of that data can be mapped. The proposal here is not appropriate for every usage of XML (such as documents), but it is appropriate for those usages that are encodings of directed, labeled graphs..." [local archive copy]

- XOL - XML-Based Ontology Exchange Language. XOL is a language for ontology exchange. It is "designed to provide a format for exchanging ontology definitions among a set of interested parties. The ontology definitions that XOL is designed to encode include both schema information (meta-data), such as class definitions from object databases -- as well as non-schema information (ground facts) , such as object definitions from object databases. XOL is similar to other past ontology-exchange languages; its development was inspired by Ontolingua and OML. XOL differs from Ontolingua in having an XML-based syntax rather than a Lisp-based syntax; the semantics of OKBC-Lite are extremely similar to the semantics of Ontolingua. XOL differs from OML in that the semantics of OML are based on Conceptual Graphs, which have a number of differences from OKBC-Lite."
- Business Rules Markup Language (BRML). BRML is an 'XML Rule Interlingua for Agent Communication, based on Courteous/Ordinary Logic Programs.' It is used in connection with 'CommonRules' from IBM, and was developed in connection with IBM's Business Rules for E-Commerce Project. A related proposal is given in the 'Agent Communication Markup Language,' a new XML version of FIPA standards-draft Agent Communication Language.'
- Ontology and Conceptual Knowledge Markup Languages. A communique from Robert E. Kent summarizes new directions for the Ontology and Conceptual Knowledge Markup Languages. Documentation for the Ontology Markup Language (OML) accessible at http://wave.eecs.wsu.edu/CKRMI/OML.html. OML was originally intended to be subservient to the more inclusive CKML (Conceptual Knowledge Markup Language) and to Conceptual Knowledge Processing (CKP). The earlier versions of OML were basically a translation to XML of the SHOE formalism (http://www.cs.umd.edu/projects/plus/SHOE/), with suitable changes and improvements. [The new design] is highly RDF/Schemas compatible, although it has its own solution to the namespace problem; but more importantly, we have incorporated our own

version of the elements and expressiveness of conceptual graphs. In fact, the current version of OML may be the first time a framework using XML and equivalent to predicate logic has been placed on the Internet. For these reasons, at least four versions of OML are being considered, each designed for a different purpose: the full Standard OML is regarded as the most expressive and natural; Abbreviated OML is for interoperability with the conceptual graphs standard CGIF (http://concept.cs.uah.edu/CG/Standard.html); Simple OML is for interoperability with RDF with schemas; and Core OML is for logical simplicity."

- [July 05, 1999] "XML Authority Ends Waiting Game for Schema Developers." By Dale Dougherty. From XML.com. July 01, 1999. [From Extensibility,] "XML Authority is a visual editing environment for creating and testing schemas. It is also good for viewing existing DTDs or other schema examples in a consistent way. For many of us, the syntax of DTDs can be confusing and awkward, and while a visual interface doesn't necessarily spare you from knowing the underlying syntax, it allows you a higher level, interpreted view of the structure you are creating. . . XML Authority regards DTDs and each of the different schema proposals as essentially the same from the developer's point of view. They use different syntax and there are some differences in features but there is a lot in common. Developers can use XML Authority as a consistent interface for building schemas, regardless of whether you want the result saved as a DTD, or in any of the prevalent schema proposals. This allows the developer to be a standards pragmatist and start getting some useful work done today."
- "Meta Content Framework Using XML", NOTE-MCF-XML. Submitted to W3C 6 June 97. Edited by R. V. Guha and Tim Bray.
- "MGML - an SGML Application for Describing Document Markup Languages." By Tim Bray (and others, see details). [local archive copy]
- "Why I Demand Schemata: Element Type Hierarchies for Transparent Document Structure Definition." By Henry S. Thompson (Language Technology Group, University of Edinburgh). Draft date: October 15, 1997. [local archive copy]
- ISO 11179 - Specification and Standardization of Data Elements. According to Frank Olken (Lawrence Berkeley National Laboratory) in a seminar description 'XML-Data/RDF for ISO/IEC 11179', "the W3C XML-Schema Working Group is currently working on several issues which could have an impact on metadata registries in general, and ISO/IEC 11179 in particular."

- XML-Data Schemas Guide. By Andrew Layman. May/July, 1999. "This paper describes the features in the XML-Data schema language implemented by Microsoft's Internet Explorer 5.0 MSXML parser. XML-Data is an XML vocabulary for describing classes of XML documents and components, that is, for defining XML element types, attribute types and declaring rules for their combination into documents (or portions of documents). This paper is a tutorial guide to the features; they are defined more exactly at http://msdn.microsoft.com/xml/XMLGuide/schema-overview.asp, and you should look there for definitive specifications.
- XML-Data Schemas Guide (First Draft, May 25 1999)
- ISO 11179 with X3.285. DTD Element Index. DTD work from ISO 11179 and X3.285 by Terry Allen (Veo Systems); HTML presentation of ISO 11179 with X3.285 facilitated by Norm Walsh's DTDParse. **NB:** this is a transient URL for the ISO 11179 DTD, so please **do not** create public bookmarks to it.
- [May 13, 1999] "XML Notation Schemas." By Rick Jelliffe. May 12, 1999. "This note is for discussion purposes by the W3C Schema Working Group. It provides an alternative characterization of the schema problem. This provides a fromwork for addressing many issues not handled in the first working draft of the XML Schema specification." [local archive copy]
- [December 01, 1999] "Describing your Data: DTDs and XML Schemas." By Simon St. Laurent. From XML.com (December 01, 1999). ['Are you confused about which XML schema syntax to use? Concerned that your XML applications remain interoperable with future XML schema standards? Simon St. Laurent guides us through the maze of XML schema languages, focusing on DTDs and XML Schemas.'] "If you've been developing with XML for even a short period of time, you are likely to have reached the point of wanting to describe your XML data structures. Document Type Definitions (DTDs) and XML Schemas are key technologies in this area. Although neither are strictly required for XML development, both DTDs and XML Schemas are important parts of the XML toolbox. DTDs have been around for over twenty years as a part of SGML, while XML Schemas are relative newcomers. Though they use very different syntax and take different approaches to the task of describing document structures, both mechanisms definitely occupy the same turf. The W3C seems to be grooming XML Schemas as a replacement for DTDs, but it isn't yet clear that how quickly the transition will be made. DTDs are

here-and-now, while XML Schemas, in large part, are for the future..."

- [May 12, 1999] "Alternatives to XML DTDs: Four Proposals." By Bob DuCharme. In <*TAG*> Volume 13, Number 4 (April 1999), pages 5-7. "In a follow-up to his analysis last month of XML's Document Type Definition (DTD) declaration syntax, Bob DuCharme focuses on the status of four alternative DTD schemas proposed by the W3C: XML-Data, XML Document Content Description (DCD), Schema for Object-oriented XML (SOX), and Document Definition Markup Language (DDML). In particular, DuCharme outlines the history and priorities behind each schema, and considers the functionality each affords to applications that manipulate metadata structured in XML. The article discusses some features that the W3C Schema Working Group members could add to the schema proposal that they'll draft after studying the four existing proposals." [See now also "World Wide Web Consortium Releases First Working Drafts of XML Schema Specification. W3C Members Collaborate to Improve and Standardize Needed Technology."]
- [March 30, 1999] "If Not DTDs, Then What?" By Bob DuCharme. In <*TAG*> Volume 13, Number 3 (March 1999), pages 1-3. "On an XML discussion mailing list, someone once claimed that no one would use DTDs if they were optional. Why bother, he asked, with something that just restricts your freedom when creating documents? XML Specification coeditor Tim Bray replied that the opposite effect had happened: people complained that DTDs did not allow enough restrictions. This article corrects most peoples' misconceptions that the current work on schemas in the W3C don't constitute alternatives to the DTD, but different ways of representing the DTD. [. . .] None of the four [current schema] proposals will ever 'win' as the accepted alternative to traditional DTD syntax. Instead, the W3C has assembled an XML Schema Working Group to evaluate the proposals and then construct a new proposal combining their best features, and probably adding some new ones as well. The Working Group's membership includes at least two authors, editors, or contributors involved in the creation of each of the original four proposals."
- Knowledge Interchange Format (KIF)
- A Discussion of the Relationship Between RDF-Schema and UML." NOTE-rdf-uml-19980804. By: Walter W. Chang (Advanced Technology Group, Adobe Systems). "This note summarizes the relationship between RDF-Schema and UML, the generic industry standard object-oriented modeling framework for information systems modeling. This note will briefly describe these

systems then relate them to each other."

- W3C XML Schema Working Group. Co-chaired by Dave Hollander of Hewlett-Packard and C. M. Sperberg-McQueen of the University of Illinois at Chicago.
- "Document Definition Markup Language (DDML) Specification, Version 1.0." NOTE-ddml-19990119. 19-Jan-1999. [local archive copy]
- DCD - Document Content Description for XML
- SOX - Schema for Object-oriented XML
- "Universal Commerce Language and Protocol (UCLP)"
- XML-Data
- Resource Description Framework (RDF) Schema Specification WD-rdf-schema-19981030, W3C Working Draft 30 October 1998. ". . . this document defines a schema specification language. More succinctly, the RDF Schema mechanism provides a basic type system for use in RDF models. It defines resources and properties such as Class and subClassOf that are used in specifying application-specific schemas." See also the W3C Resource Description Framework (RDF)
- XML Authority - "a graphical design tool accelerating the creation and enhancing the management of schemas for XML. With support for data typing, solutions for data interchange and document oriented applications converge. XML Authority includes a toolset to help convert existing application and document structures to schemas, defining the basis for well formed XML documents and enabling valid XML. Beta, 1999-03-25. See XML & Schemas.
- [January 15, 1999] "XML Schema Languages." By Ronald Bourret. January, 1999. Presentation slides. Summary: "I recently gave a presentation on XML schema languages. Due to the recent questions about these on XML-Dev, I've annotated my slides and made them available on the Web. A majority of my audience didn't know XML, so the presentation starts with a brief description of XML. It then discusses why you might want XML schema languages, their basic syntax, and what the major differences between the four existing languages (XSchema, DCD, SOX, and XML-Data) are. It ends with a summary of what I think a schema language should have today and what languages I think you should use for what purpose. . . This presentation briefly reviews the current (January 1999) state of XML schema languages -- why we have them, how to use them, and what each language offers. Disclaimer 1: I am one of the co-authors of XSchema. Because of this, there is likely to be some bias, especially in the 'Existing XML

Schema Languages' and 'Summary' sections. If you have complaints, comments, or suggestions, please send me email. Disclaimer 2: Because I know XSchema, it is used as the sample language for illustrating most schema language concepts. DCD, SOX, or XML-Data could have been used equally well." Also available in Powerpoint format. **Note:** The W3C has recently chartered an XML Schema Working Group, co-chaired by Dave Hollander of Hewlett-Packard and C. M. Sperberg-McQueen of the University of Illinois at Chicago.

- Object Management Group (OMG) and XML Metadata Interchange Format (XMI)
- [October 28, 1999] **First Working Draft of ISO/ WD 10303-28: XML Representation of EXPRESS Driven Data.** A first working draft of ISO 10303-28 has been developed by Eurostep and Monsell EDM for BSI. This is: *ISO/WD 10303-28:1999(E). Product data representation and exchange: Implementation methods: XML representation of EXPRESS-driven data.* Reference: ISO TC184/SC4/WG10 N285 and ISO TC184/SC4/WG11 N090, Date: 1999-10-24. This "first rough draft of part 28" specifies "the way in which XML can be used to encode both EXPRESS schemas and corresponding data." An accompanying document explaining the use of Use of Architectural Forms is also available: *"Use of Architectural Forms for Early to Late Bound Mapping WG11 N91.* These two documents have been sent to SOLIS as WG11/N90 and N91, and are intended to form the basis for discussion at an upcoming meeting in New Orleans. The goal of the project in this new work item is explained in the introduction to the proposed standard: "ISO 10303 is an International Standard for the computer-interpretable representation of product information and for the exchange of product data. The objective is to provide a neutral mechanism capable of describing products throughout their life cycle. This mechanism is suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases, and as a basis for archiving. This part of ISO 10303 specifies means by which data and schemas specified using the EXPRESS language (ISO 10303-11) can be encoded using XML. XML provides a basic syntax that can be used in many different ways to encode information. In this part of ISO 10303, the following uses of XML are specified: a) A late bound XML architectural Document Type Declaration (DTD) that enables any EXPRESS schema to be encoded; b) An extension to the late bound DTD to enable data corresponding to any EXPRESS schema to be encoded as XML; c) A canonical form for the late bound DTD that is derived from the architectural

DTD; d) The use of SGML architectures to enable early binding XML forms to be defined that are compatible with the late binding. The use of architectures allows for different early bindings to be defined that are compatible with each other and can be processed using the architectural DTD." The Architectural Forms document (by Robin La Fontaine) "explains the basics of SGML Architectures as needed to represent the relationship between the early-bound and late-bound XML formats for Express-driven data... Given a document in XML which corresponds with a particular DTD, architectural forms provide a standard mechanism for viewing it as if it were consistent with another DTD (the meta-DTD or base architecture). This is being used within STEP to allow one or more early-bound data sets to be viewed as if they were defined in terms of the standard late-bound DTD. Thus software written against the late-bound DTD can, without modification, process data that complies with any compliant early-bound DTD. 'Compliant' here means that the early-bound DTD has the late-bound DTD as its base architecture. This gives some flexibility in defining early-bound DTDs which can be optimised for different purposes, *e.g.*, for display, for data exchange, for compactness." Persons interested in the activity of this ISO group may contact the Nigel Shaw (Project Leader, Eurostep Limited) or Robin La Fontaine (Project Editor, Monsell EDM). For background on this proposed new work item, see "SGML/XML and STEP"; see also (tangentially) "Product Data Markup Language (PDML)."

- [March 30, 2000] "XML Schemas: Setting Rules for XML Documents." By Simon St.Laurent. March 2000. Slideset presentation (24 slides). "Why schemas? (1) Common Vocabularies: Establishing common vocabularies makes it easy to build software that processes information according to a clearly defined set of rules. The larger the audience using the same vocabulary, the larger the audience. (2) Formal Sets of Rules: Because machines (computers) will be doing most of the XML processing, expressing those vocabularies in a form that computers can understand is important. The formal description must be regular, unambiguous, and relatively easy to process. (3) Building Contracts: On the human side of the information interchange equation, formal descriptions of vocabularies can provide a core set of rules for all participants in a series of transactions. Schemas can make it clear which kinds of information are required or optional for which kinds of transactions. ...it only covers what I could say in 90 minutes."
- [January 12, 1999] The ISO TC 184/SC4 [Industrial data] Secretariat has issued an ISO

New Work Item Ballot for "XML representation for EXPRESS-driven data." The NWI 'specifies the representation according to the syntax of Extensible Markup Language (XML) of data defined using ISO 10303-11 (the EXPRESS language) and/or for EXPRESS schemas. The mappings from the EXPRESS language to the syntax of the representation are specified. Any EXPRESS schema or schemas and the data they describe can be represented.' The current proposal 'arises out of the preliminary work item on SGML and Industrial Data (commonly refered to as 'STEP/SGML Harmonisation') and is seen as an important part of that initiative. The use of XML will enable increased flexibility with respect to future changes to EXPRESS schemas. The result of the NWI will enable the generalized use of XML and SGML tools and web browser technology with EXPRESS-driven data and schemas.' The facility would 'enable the use of the recommended syntax for data exchange on the World Wide Web to be applied to instances of EXPRESS-driven data, enable the use of the recommended syntax for data exchange on the World Wide Web to be applied to EXPRESS schemas, and enable EXPRESS schemas to be exchanged together with data instances they describe.'

- ISO/IEC 11404 INTERNATIONAL STANDARD. Information technology -- Programming languages, their environments and system software interfaces -- Language-independent datatypes. First edition 1996-12-15. "This International Standard provides the specification for the Language-Independent Datatypes. It defines a set of datatypes, independent of any particular programming language specification or implementation, that is rich enough so that any common datatype in a standard programming language or service package can be mapped to some datatype in the set." [local archive copy]
- "Beyond the SGML DTD." By François Chahuneau. Posting submitted to the W3C WG discussion forum.
- Extensible Type Specifications for RDF and XML Schemas." By Frank Olken. Lawrence Berkeley National Laboratory. September 11, 1998. DRAFT 1.0. "This document addresses issues of extensible type specifications for use in XML and RDF schemas, i.e., schemas which describe information encoded as either XML or RDF documents. XML documents are described by XML schemas such as XML-Data and DCD (Document Content Definitions). RDF documents are described by W3C RDF schemas. We are particulary concerned with two issues: 1) decomposing the descriptation of basic types; 2) extensibility of type specifications." [local archive

copy]

- "Adding Strong Data Typing to SGML and XML", by Tim Bray. May [21,] 1997. archive copy, May 21, 1997; or: previous archive copy, May 15, 1997]. Note: Jean Paoli of Microsoft has submitted a related proposal in connection with the XML discussion "XML for Structured Data"
- [December 06, 1997] "Why I Demand Schemata: Element Type Hierarchies for Transparent Document Structure Definition." By Henry S. Thompson (Language Technology Group, University of Edinburgh). Draft date: October 15, 1997. Overview: "In this paper I describe the XML-Data schemata proposal, concentrating on the motivation for and nature of the provision of an element-type hierarchy, in which element types can inherit attribute declarations and positions in content models from ancestors in the hierarchy. I argue that this represents a major improvement over the use of parameter entities to structure and maintain DTDs." [local archive copy]
- [September 03] "Tips and Techniques." By [<TAG> Online Staff]. In *<TAG>* Volume 13, Number 7 (July, 1999), pages 8-10. "This month's article shows you how to create an XML schema in accordance with the current draft spec from the W3C XML Schema Working Group. 1. Create XML schema, 2. Create document according to schema, 3. Point to schema using namespaces. Remember that this article is based on a working draft of the XML schema specification. The final recommendation might look slightly or completely different from this, but the concepts will remain the same. . . The example shows the structure of our Joke Markup Language. The root element is 'Joke', which contains three elements (Setup, PunchLine, and OneLiner) and two attributes (Type and FirstUsed)." [See http://architag.com/newsletter/joke.xsd.]
- "MGML - an SGML Application for Describing Document Markup Languages." SGML '96 (?) [local archive copy]
- "Meta Content Framework Using XML", NOTE-MCF-XML. Submitted to W3C 6 June 97. Edited by R. V. Guha and Tim Bray. "This document provides the specification for a data model for describing information organization structures (metadata) for collections of networked information. It also provides a syntax for the representation of instances of this data model using XML, the Extensible Markup Language."

[CR: 19980217]

Chahuneau, François. "SGML and Meta-information: From SGML DTDs to XML-DATA."

Pages 337-340 in *SGML/XML '97 Conference Proceedings*. SGML/XML '97. "SGML is Alive, Growing, Evolving!" The Washington Sheraton Hotel, Washington, D.C., USA. December 7 - 12, 1997. Sponsored by the Graphic Communications Association (GCA) and Co-sponsored by SGML Open. Conference Chairs: Tommie Usdin (Chair, Mulberry Technologies), Debbie Lapeyre (Co-Chair, Mulberry Technologies); Michael Sperberg-McQueen (Co-Chair, University of Illinois). Alexandria, VA: Graphic Communications Association (GCA), 1997. Extent: 691 pages, CDROM; print volume contains author and title indexes, keyword and acronym lists. Author's affiliation: [François Chahuneau]: AIS (Advanced Information Systems) S.A., 17 Rue Remy Dumoncel, Paris, France F-75014; Email: fcha@ais.berger-levrault.fr; WWW: http://www.ais.berger-levrault.fr/.

Abstract: "This paper studies, from an historical perspective, the relationship between SGML and data modeling concerns. SGML did not the invent the concept of structural document models, or 'schemata'. Nevertheless, through the notion of DTDs, it made this powerful concept available and understandable to a large number of people with little or no data modeling experience.

"With the evolutionary trend towards 'content oriented' DTDs, the emergence of well-described methodologies to design them and the appearance of specialized 'case' tools to manipulate them, the potential of SGML as a data modeling methodology became clear, and some SGML enthusiasts suggested to use it as a general purpose tool.

"However, because an SGML DTD intimately mixes the notion of a 'grammar' and that of a 'schema', these two concepts remained partly confused, at least in the 'orthodox' SGML approach. This original characteristic caused some misunderstandings and raised many suspicions from the 'traditional' data modeling world. This largely precluded, so far, the use of SGML as a general data modeling tool outside the restricted arena of structured documents.

"By introducing a simplified syntax with a fixed grammar, XML isolated the role of DTDs as 'pure schemata', and also made them unnecessary for pure recognition of the 'de facto' document structure.

Finally, recent proposals such as MCF and XML-data suggest to use the XML syntax itself to encode document schemata, therefore making 'traditional' DTDs obsolete.

At the same time, they propose several extensions to the SGML data modeling semantics, by incorporating object-oriented concepts. Will such an evolution allow XML to become the official, well-accepted and ubiquitous way to exchange structured data and associated models, and bring SGML power much beyond its original application niche?"

[Extract from the section "The Dual Nature of DTDs"]: "With the benefit of hindsight, after ten years of practice, the design of SGML appears as an unlikely and unique mixture of many brilliant ideas and a few mistakes, and strikes [one] by its total lack of references to data modeling or language design theories which had already emerged in computer science at the time it was designed. A major point of originality is the central SGML DTD concept itself: a DTD is *both a generative grammar* for the markup language which will be used to tag corresponding instances, and a *schema* which characterizes a document class: it assigns names to things and defines rules stating what structural patterns shall or shall not be not possible/required in an SGML document (modeled as a tree of typed nodes with attributes) which belongs to the class. In the same set of statements, one is instructed that 'the end tag for AUTHOR can be omitted' and that 'the document must have a title and a single one', although these two pieces of information admittedly belong to totally different areas of concern. This dual nature of DTD should not necessarily lead to confusing the two notions. Unfortunately, this is largely what happened in the SGML community..."

Document URL: http://xml.coverpages.org/schemas.html — Legal stuff

---

TOPICS
Business
Databases
Graphics
Metadata
Mobile
Programming
Schemas
Style
Web
Web Services

ESSENTIALS
Annotated XML
What is XML?
What is XSLT?
What is XSL-FO?
What is XLink?
What is XML Schema?
What is XQuery?
What is RDF?
What is RSS?
What are Topic Maps?
What are Web Services?
What are XForms?
XSLT Recipe of the Day

Manage Your Account
Forgot Your Password?

FIND
Search
Article Archive

COLUMNS
<taglines/>

Parsing the Atom

Print
Email article link

by Leigh Dodds
April 25, 2001

This week XML-DEV has been considering some interesting twists on XML data processing, prompted by the use of regular expressions in the W3C XML Schema specification to define complex data types.

Complexity

While the world may be increasingly surrounded by pointy brackets, the majority of data exchanged isn't XML, which will be true for some time to come (if not always). Yet even in situations where data has been generated as XML, there is a near infinite variety of forms which that markup can take. Hence the numerous initiatives to define horizontal and vertical XML standards; these schemas limit the acceptable forms of XML documents to those deemed suitable for particular application or business uses.

However the core flexibility of XML -- the ability for anyone to quickly define and produce their own formats -- will mean that a variety of document types will evolve and coexist. There will never be a single blessed way to markup a single piece of information, just acceptable forms for particular processing contexts.

Where does that leave the Desperate XML Hacker? How does she deal with the following variety of document fragments:

```
<date>30-4-1972</date>
<date day="30" month="04" year="1972">30th April 1972</date>
<date>
 <day>30</day>
 <month>4</month>
 <year>1972</year>
</date>
```

All these fragments are legal XML, they merely differ in the granularity of their markup. The last fragment is obviously the most granular, and the most convenient for processing with XSLT, for example. But what about this fragment of Scalable Vector Graphics Markup (SVG)?

```
 <path
d="M159.213,120.358H12.287V13.104h146.926v107.254z"/>
```

Or perhaps this example of CSS styling?

Dive into XML
Hacking the Library
Jon Udell
Perl and XML
Practical XQuery
Python and XML
Rich Salz
Sacré SVG
Standards Lowdown
Transforming XML
XML Q&A
XML-Deviant

GUIDES
XML Resources
Buyer's Guide
Events Calendar
Standards List
Submissions List

TOOLBOX
Syntax Checker


```
<a href="http://www.w3.org/"
          style="{color: #900}
          :link {background: #ff0}
          :visited {background: #fff}
          :hover {outline: thin red solid}
          :active {background: #00f}">...</a>
```

Not quite so amenable to processing and certainly not with XSLT's limited string handling capabilities. The only resort then is to fall back on application code to parse, and process this information. Or perhaps not.

The second XML Schema specification describes the means to define simple and complex datatypes in W3C XML schema. This goes a long way toward standardizing the legal forms of datatypes in XML documents; the specification itself defines many simple types such as numbers and dates, etc. An important aspect to these definitions are the use of regular expressions to define the legal forms of type values. The specification refers to the definitions of these type values as atoms, which are used to validate the contents of elements declared to have specific types.

But what if we could do more than simply validate this content? What if we could use these regular expressions to extract atoms of data from element and attribute values? These are the questions that XML-DEV has been considering.

Splitting the Atom

Simon St. Laurent began the discussion, unhappy with the current simple and complex types in W3C XML Schemas.

> *It seems as if regular expressions could be used not just for validation of typed content, but for fragmentation of typed molecules into smaller atoms. Instead of binding users to a particular (ISO 8601) date format, this approach would let users provide their own rules for fragmenting date strings into the parts we need for processing -- year, month, day, etc.*
>
> *It would also open up the prospect of treating other compounds -- like the CSS style attribute, some of the path information in SVG, and various other places where the principle of one chunk, one string has been violated -- as a set of atoms which could themselves be validated and/or transformed and/or typed.*
>
> *This leads to another kind of post-processing infoset, where the atoms are available as an ordered set of child nodes, but it seems like a promising road.*

Instead of relying on sections of application code to process this information, instead defining a mechanism to pull out atoms of data from within "molecules" of information, St. Laurent argued that interoperability would be increased.

> *[B]y making it possible to access that information in multiple environments, you preserve a lot of the interoperability that XML promises. Working with atoms lets us avoid a lot of not-very-portable application code.*

St. Laurent was not alone. Steve Rosenberry suggested a similar mechanism to the Schema Working Group earlier in the year. Rosenberry was particularly interested in being able to define values that included units of measurement.

> *My motivation was to specify an attribute as a numeric value with units of measure attached to it for absolute clarity. (Ask NASA how important this might be. They lost a Mars probe because numbers had no units associated with it and one group assumed metric while the other group was specifying English.)*
>
> *...Since the only guideline for when one should use attributes vs. elements is "It depends upon the application", I don't see this further structuring,*

> *parsing, and use of attributes as fundamentally wrong given that it has clear functionality for certain developers and users of XML.*

This highlights another aspect to the debate. If there *are* several atoms of information within a given value, then surely they must be explicitly identified. Henry Thompson was one XML-DEV member with this viewpoint:

> *Strong disagreement (speaking personally). We have a way in XML to express compound objects -- it's called elements-and-attributes. The mistake, in my opinion, was giving in to the SQL people and having _any_ kind of date or time as simple types -- they should _all_ have gone in to the type library as complex types.*

Thompson suggests here that there are many simple XML Schema types that may better have been explicitly defined as complex types. Indeed, Thompson (again expressing his personal viewpoint) believes that the most granular form of markup is the correct approach.

Recognizing the utility of regular expression processing of information atoms for simple examples, Michael Brennan had some concerns over a possible trend toward sparsely marked-up data.

> *...I hope there is not going to be too much of a trend toward doing this sort of thing. In my mind, if a datatype has some structure to it, why not just make it a complex type and leverage XML syntax to convey that structure? Isn't that really the whole point of XML -- a standardized syntax for conveying structure?*

Anders Tell had similar reservations.

> *That doesn't mean that standardizing [dates and times] is bad, but they should be standardized as compound types -- otherwise, it will encourage people to make other things into "simple" types with their own special parsing rules (For example, phone numbers, ZIP codes, UK Postcodes...*

While philosophically in agreement with this viewpoint, St. Laurent believed that the trend toward data clumps was already in evidence, citing several examples. St. Laurent stated his goal as dealing with the variety of data already being produced:

> *I'm afraid the trend's already happened...*
>
> *Given that situation, I'd like very much to have a means of breaking into different lexical forms representing such compounds without having to revert to full-scale XML Schema processing.*
>
> *It's not so much that I want to encourage such things, but that they already exist and that I'm not especially impressed with current models for processing and handling them.*

Expressing similar sentiments, Jonathan Borden highlighted the role of St. Laurent's proposal within a typical XML processing chain and suggested that the feature need not be limited to XML Schemas.

> *Generally I agree with this sentiment that markup is the best way to represent structured data. The problem exists with getting stuff into the proper form -- especially when the data you are handed isn't organized in an ideal fashion. I like to use a processing chain in such cases, and to the extent that regular expression matching/parsing can be integrated into an XML processing chain, we might use standard XML techniques such as XSLT transforms to "clean up" such data into a properly structured form.*
>
> *I'm not sure I need this facility in XML Schema per se, what I would really like is an XSLT/XPath regular expression function to include variable bindings. Recursive parsing or character data in XSLT is a fairly ugly*

proposition at the moment, but something that is frequently needed in practical applications.

Microparsing and Beyond

Use of regular expressions is only the beginning, although in most cases they're likely to hit the 80/20 sweet spot for most functions. Complex data molecules such as XPath expressions and SVG paths may require more complex parsing rules. Extracting information in this manner is often termed 'microparsing'. It is possible to identify an alternative, more verbose XML syntax for this information that may be easier to manipulate with the available tools.

Also in XML-Deviant

The Courtship of Atom

Politics By Any Other Name

PyCon 2004: Making Python Faster and Better

Semantic Web Interest Group

Community Developments

This is an interesting parallel to the recent effort to define an XML encoding for XPath. Robin Berjon also posted a simple example demonstrating the generation of SVG paths from an alternative path vocabulary. One can view this as a downward translation, or compression, into a more readable, succinct format. Obviously it is desirable that this format can also be uncompressed or upwardly translated into its equivalent form. Len Bullard observed that these kinds of encoding have been a common feature of standardization efforts.

> *This points out something that recurs a lot and was noted often in the days before people were trained to think of markup languages as a single DTD or schema which everyone implements: the need for an up/down transformable language spec. This approach was well-understood and documented in the past.*
>
> *For a standard language there should be:*
>
> 1. A form that is an up translation target. The maximum information form into which and out of which other forms can be created.
> 2. Specifications for compressed format. These should be normatively expressed as the transform itself.
>
> *... SVG with minimized paths is a compressed form, one of the possible normative compressed forms which could be documented by transform.*

To conclude, it seems there may be some mileage in defining some finer-grained utilities for manipulating data within XML markup. Although regular expressions have long been a part of SGML and XML markup, as they form the basis for XML schema languages in general, it seems that their formal addition to XML Schemas opens up some additional possibilities. Perl programmers may feel justifiably smug. C developers may wish to look at Hackerlab Rx-XML, which is a regular expression matcher that processes XML Schema regular expressions

Whether the individual design decisions that have lead to complex data formats within elements and attributes are themselves questionable is a moot point (and has previously been debated on XML-DEV). The very real situation is that there are a many varieties of data and markup to deal with, and it's always handy to have a few extra tools in the toolbox to handle them.

Hintergrund

Der W3C-Standard *XML-Schema* ist eine XML-Sprache zur Definition von XML-Vokabularen.
Als Nachfolger der bekannten Document Type Definitions (DTD) markiert XML Schema den entscheidenden Wendepunkt von der bisherigen dokumentenorientierten Sichtweise hin zu einer datenorientierten .
Technisch gesehen erweitern sich durch XML Schema die Ausdrucksmöglichkeiten bei der Formulierung neuer XML-Vokabulare entscheidend. Einerseits hinsichtlich inhaltlicher Merkmale wie Datentypen; andererseits auch in Richtung struktureller Merkmale. So halten bekannte Konzepte aus den Programmier- und Datenbanksprachen Einzug.
Inzwischen haben namhafte Firmen und Initiativen ihre Unterstützung des am 2. Mai 2001 durch das W3C verabschiedeten Standards erklärt; zu Lasten anfänglich konkurrierender proprietärer Vorschläge.

Diese Seite trägt einige Informationen zum Themengebiet zusammen; dabei berücksichtigt sie neben dem offiziellen W3C-Standard auch andere Ansätze und Ideen.

Parser und Werkzeuge mit Schema-Unterstützung

- Autorenwerkzeug
- IBM and the Apache project (Xerces J)
  unterstützt Schema-Recommendation (2001-05-02)
- IBM and the Apache projekt (Xerces C)
  unterstützt Untermenge der Schema-Recommendation (2001-05-02)
- MSXML
  unterstützt Schema-Recommendation (2001-05-02) ab Version 4.0 B2
- Multi-Schema Validator (MSV)
  unterstützt eine Untermenge von XML-Schema Teil 1
  ferner werden unterstützt: RELAX NG, RELAX Namespace, RELAX Core, TREX, DTDs
- Oracle (XML Schema Prozessor for C)
  unterstützte Schema-Version: 2000-04-07
- Oracle (XML Schema Prozessor for C++)
  unterstützte Teile der Schema-Recommendation (2001-05-02)
- Oracle (XML Schema Prozessor for Java)
  unterstützte Schema-Version: 2000-10-24
- TIBCO Extensibility (XML Authority)
  unterstützt Schema-Recommendation (2001-05-02)

- Trang --- Ein Werkzeug zur Konversion zwischen verschiedenen Schemasprachen (derzeit: RELAX NG (compact und XML-Syntax), XML DTDs und W3Cs XML Schema)
- University of Edinburgh and W3C (XSV)
  unterstützt Schema-Recommendation (2001-05-02)
- XBuilder Entwurfs- und Dokumentationswerkzeug
- XML4C
  unterstützt Teil der Schema-Recommendation (2001-05-02)
- XML4J
  unterstützt Schema-Recommendation (2001-05-02)
- XML Authority
  unterstützte Schema-Version: 2000-10-24
- XMLSpy (XML Spy)
  unterstützt Schema-Recommendation (2001-05-02)
- XML Spy
  unterstützte Schema-Version: 2001-05-02
- XSDComp (XML Schema compilation) Schema-Validierung mit XSLT
  unterstützte Schema-Version: 2000-04-07

Links

- A. Layman: *XML Schema NG Guide*
- Artikel von Simon St. Laurent @ XML.com über DTDs und Schemata
- Ashvin Radiya: The basics of using XML Schema to define elements
- Best Practices
- Cambridge Communique
- Comparative Analysis of Six XML Schema Languages
- Datatypes for DTDs (DT4DTD)
- deutsche Übersetzung der XML Recommendation
- Document Content Definition (DCD)
- Document Description Markup Language (DDML)
- Document Structure Description (DSD)
- John Lam: XSL Transformations: XSLT Alleviates XML Schema Incompatibility
- Leigh Dodds: Schema round-up
- Leigh Dodds: *Spotlight on Schemas*
- Leigh Dodds: XML Schema revisited
- Projekt zur Übersetung der XML-Schema-Spezifikation ins Deutsche
- Reasoning about XML Schema Languages using Formal Language Theory
- RELAX (REgular LAnguage description for XML)
  Auf dem Wege zum ISO-Standard (ISO/IEC DTR 22250-1)
  RELAX NG integriert RELAX und TREX zu einer eigenständigen Schemasprache
  - englische Übersetzung der Spezifikation
  - Formale Semantik
  - Jing -- eine RELAX NG Implementierung
  - Learning to RELAX
  - Regular Language description for XML
  - RELAX NG Schema für RELAX NG
  - RELAX NG-Schema für W3C's XML-Schema
  - RELAX NG Technical Committee @ Oasis
  - Tutorial
- Rick Jeliffe: XML Schemas Endgame
- Roger Costello: Schema Tutorial
- Schema for object oriented XML (SOX)
- Schema Parser

- Schema Questionaire/Anwenderumfrage
- Schematron
  - Chimezie Ogbuji: *Validating XML with Schematron*
  - Specification
  - Article @ SunWorld
  - Topologi's Validator(unterstützt auch W3Cs XML-Schema)
- Tim Bray: *Adding Strong Data Typing to SGML and XML*
- TREX - Tree Regular Expressions for XML
- TREX -- Tree Regular Expressions for XML
  - PyTREX -- Python Implementierung
  - Spezifikation
  - Tutorial
- W3C's XML Schema
  - Using W3C XML Schema
  - XML Schema Part 0: Primer
  - XML Schema Part 1: Structures
  - XML Schema Part 2: Data Types
  - XML-Schema mailing list @ W3C
  - XML Schema Requirements
  - XSV -- Schema Validator @ W3C
  - Schema Converter(konvertiert ältere Schemadokumente in aktuellen Stand)
  - Mailingliste @ Yahoo
  - Mailingliste @ W3C
  - Kevin Williams: *Soapbox: Why XML Schema beats DTDs hands-down for data*
- W3C Schema Tests
- XML @ W3C
- XML 1.0 Recommendation
- XML4J(validierender XML Parser der XML Schema unterstützt)
- XML Authoritiy(graphisches Schemadesign-Werkzeug mit Unterstützung verschiedener Schema-Dialekte)
- XML-Data
- XML Schema Part 1 API (Nikita Vinokurov)
- XML schema requirements
- Zusammenstellung verschiedener Schema Referenzen

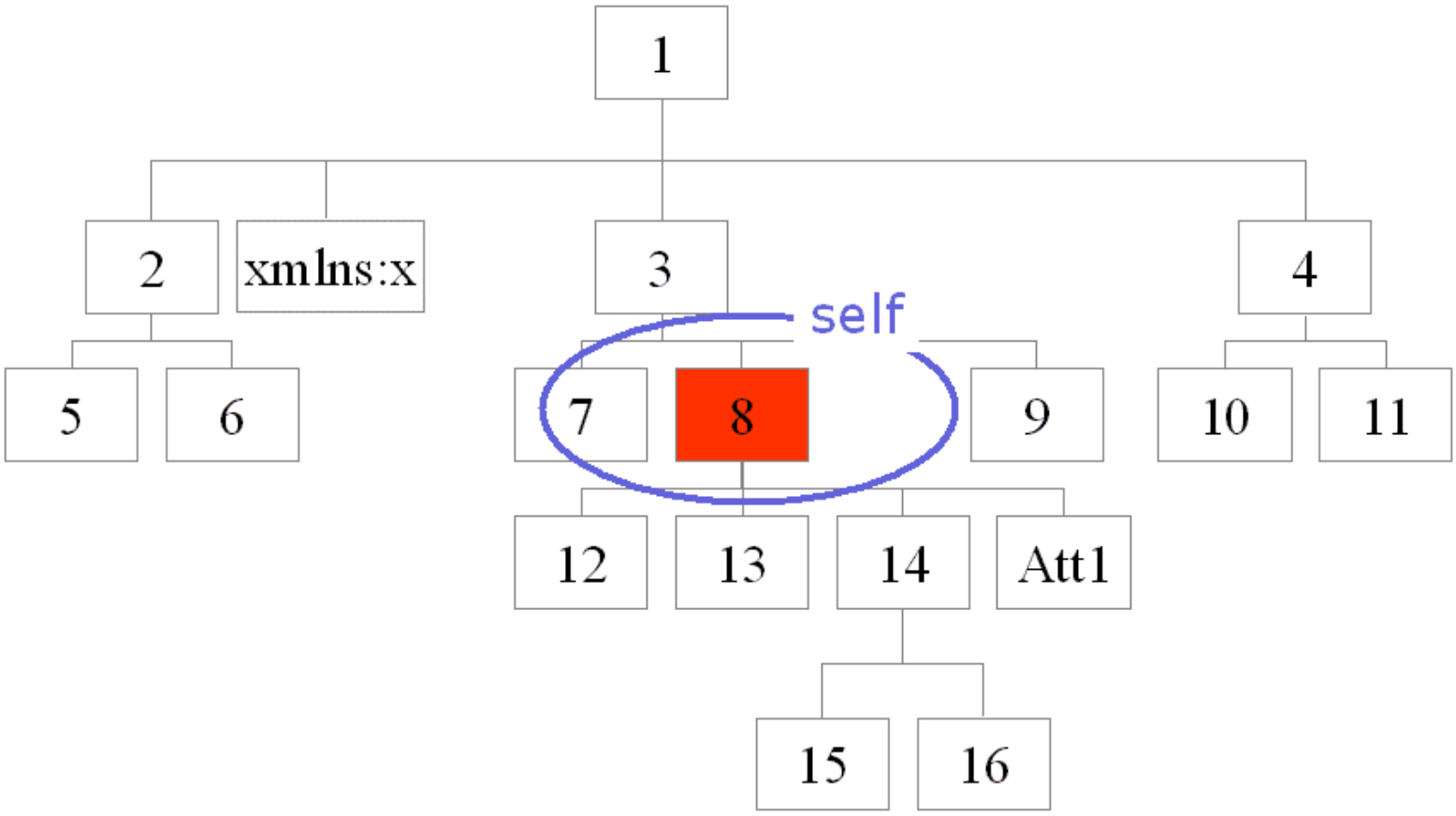
Service provided by Mario Jeckle

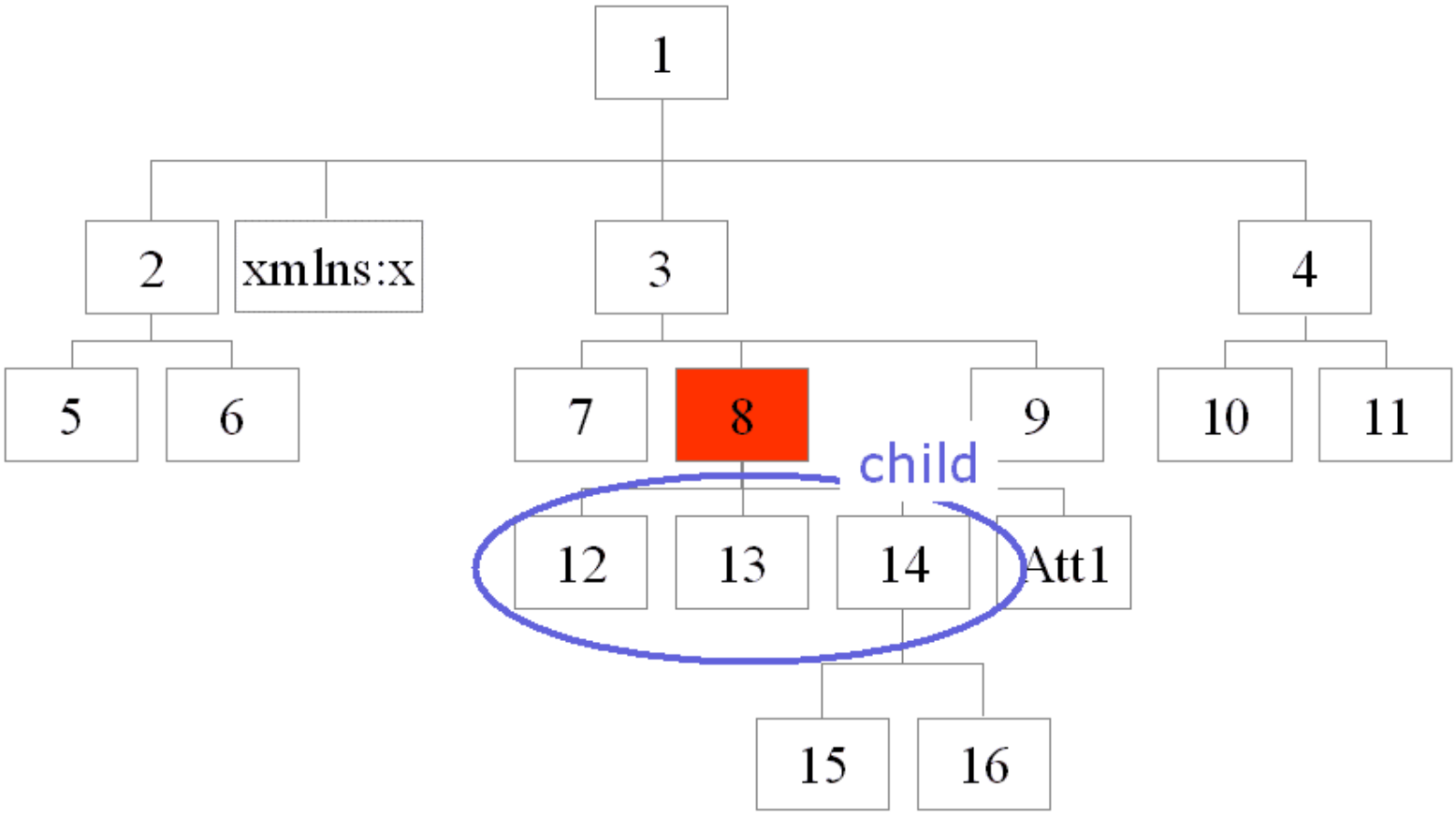
Generated: 2004-05-24T13:35:17+01:00

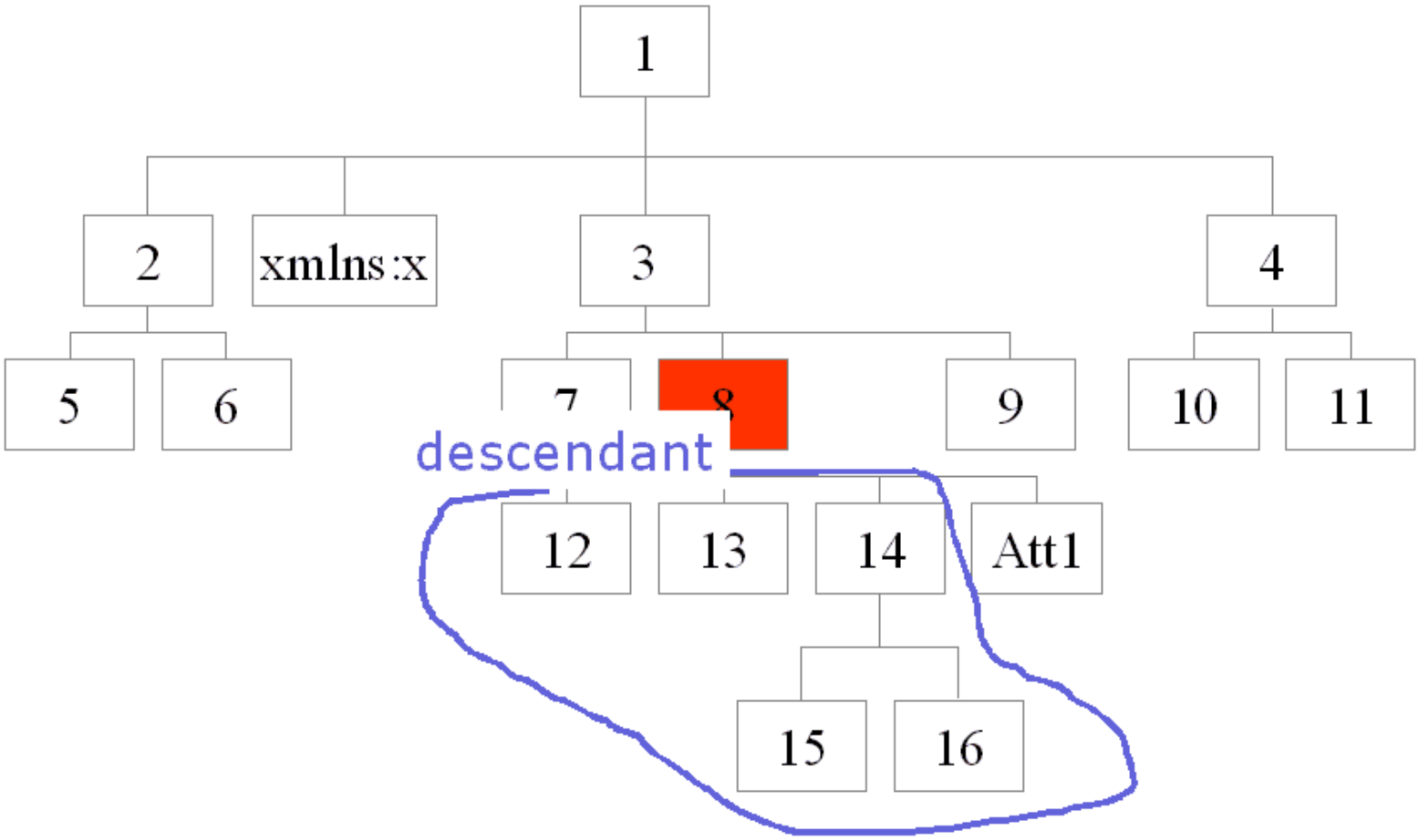
Feedback SiteMap

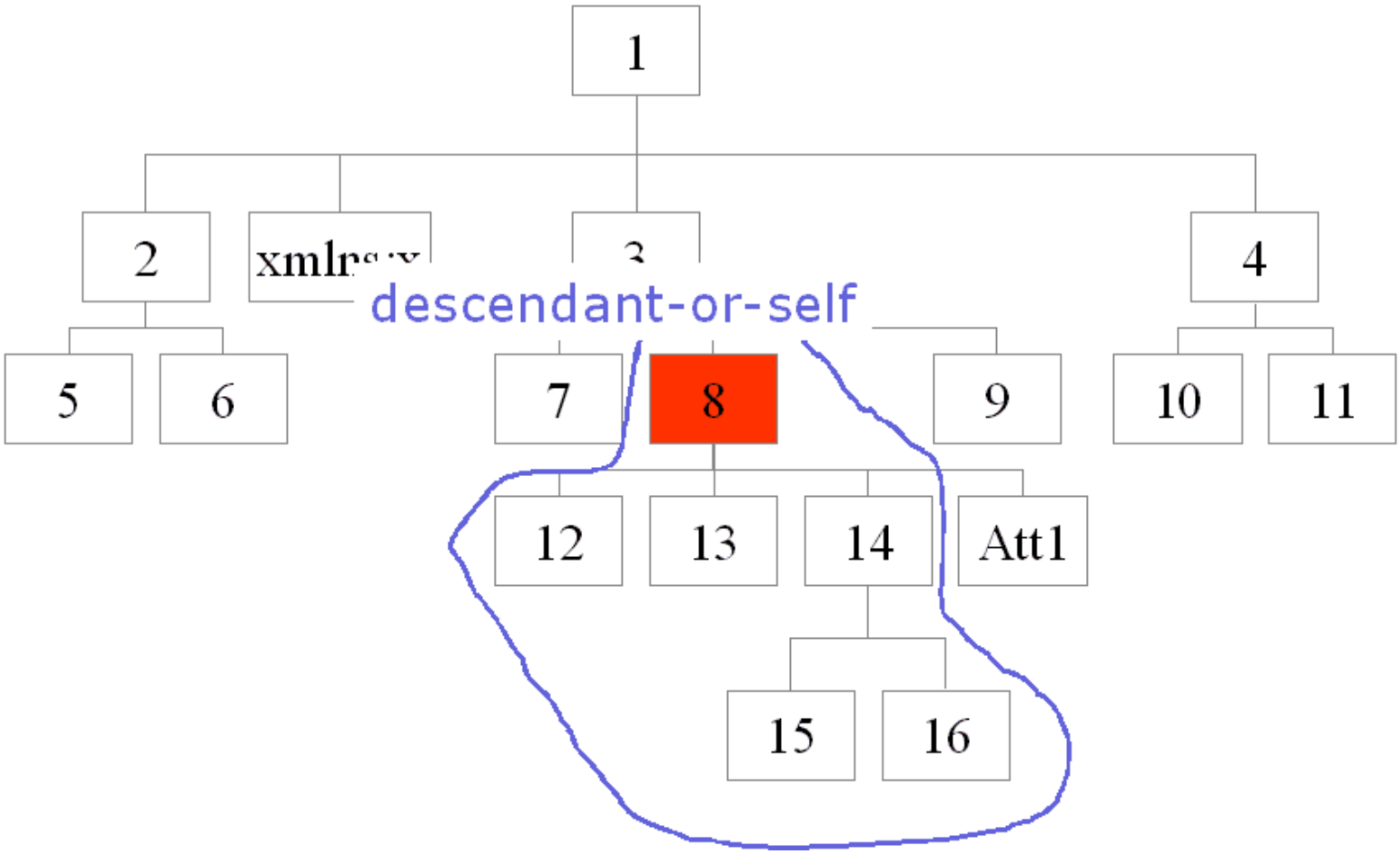
This page's original location: http://www.jeckle.de/xml/schema.html

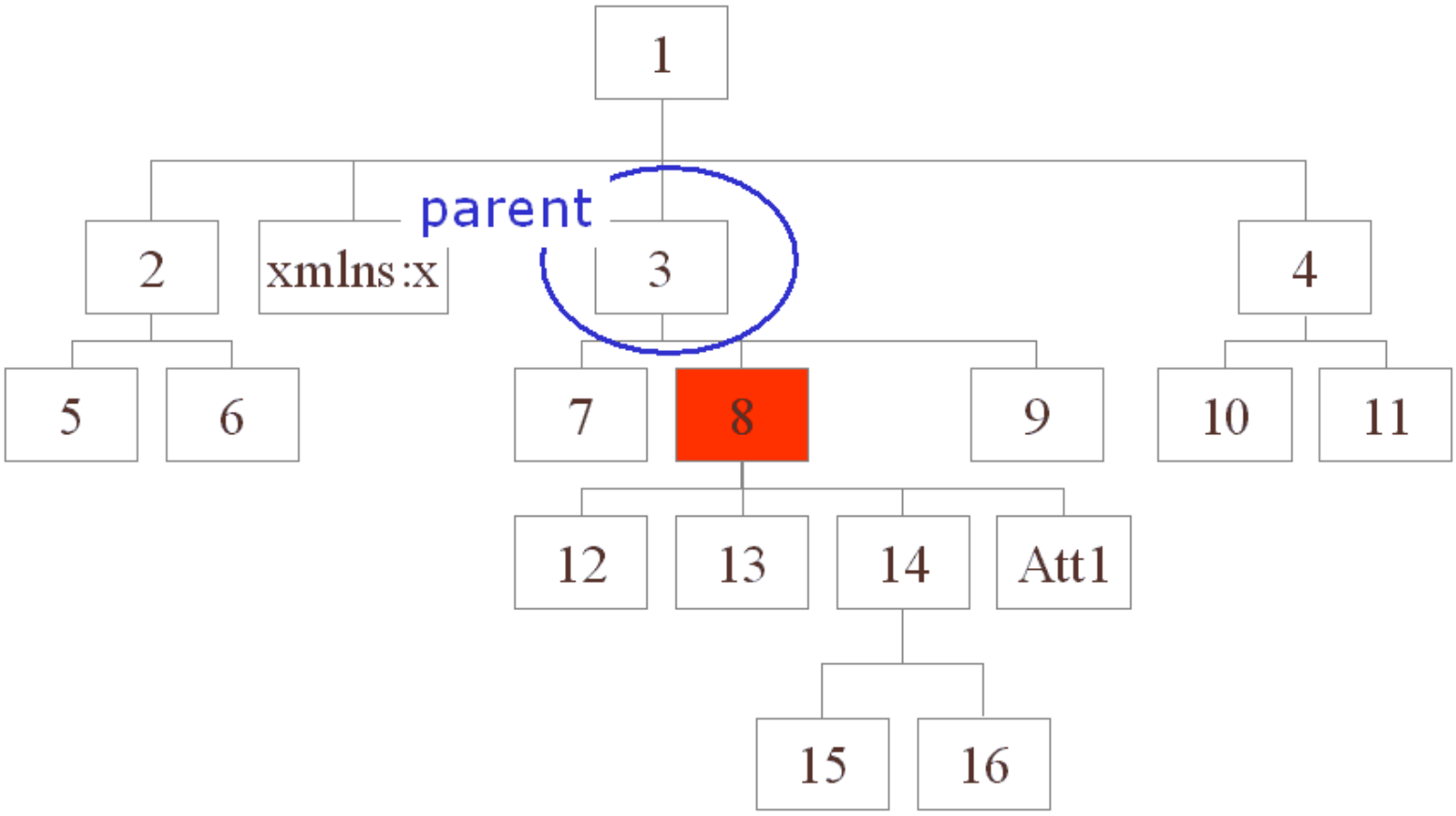
RDF description for this page

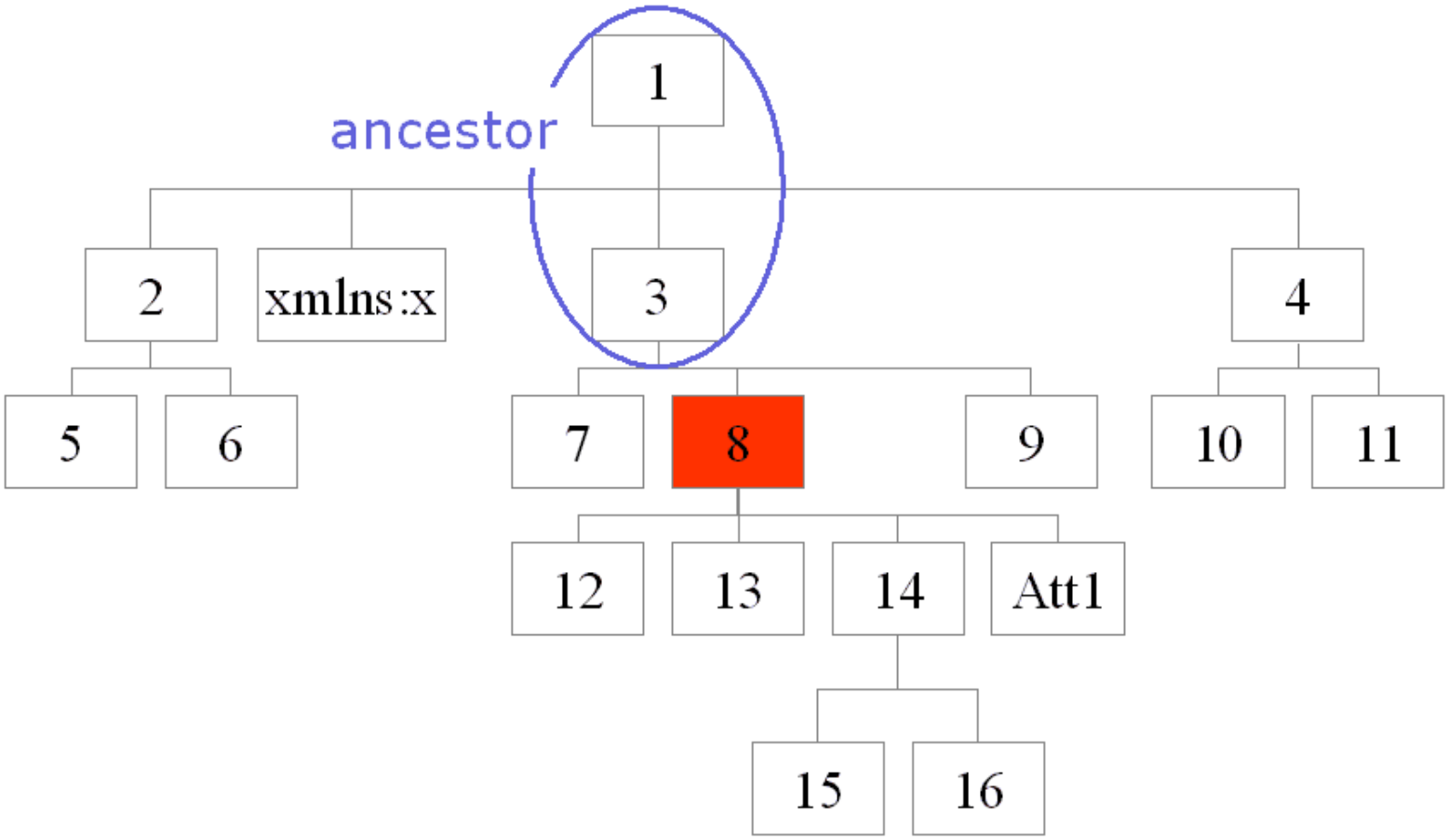

1
2
xmlns:x
3
4
self
5
6
7
8
9
10
11
12
13
14
Att1
15
16

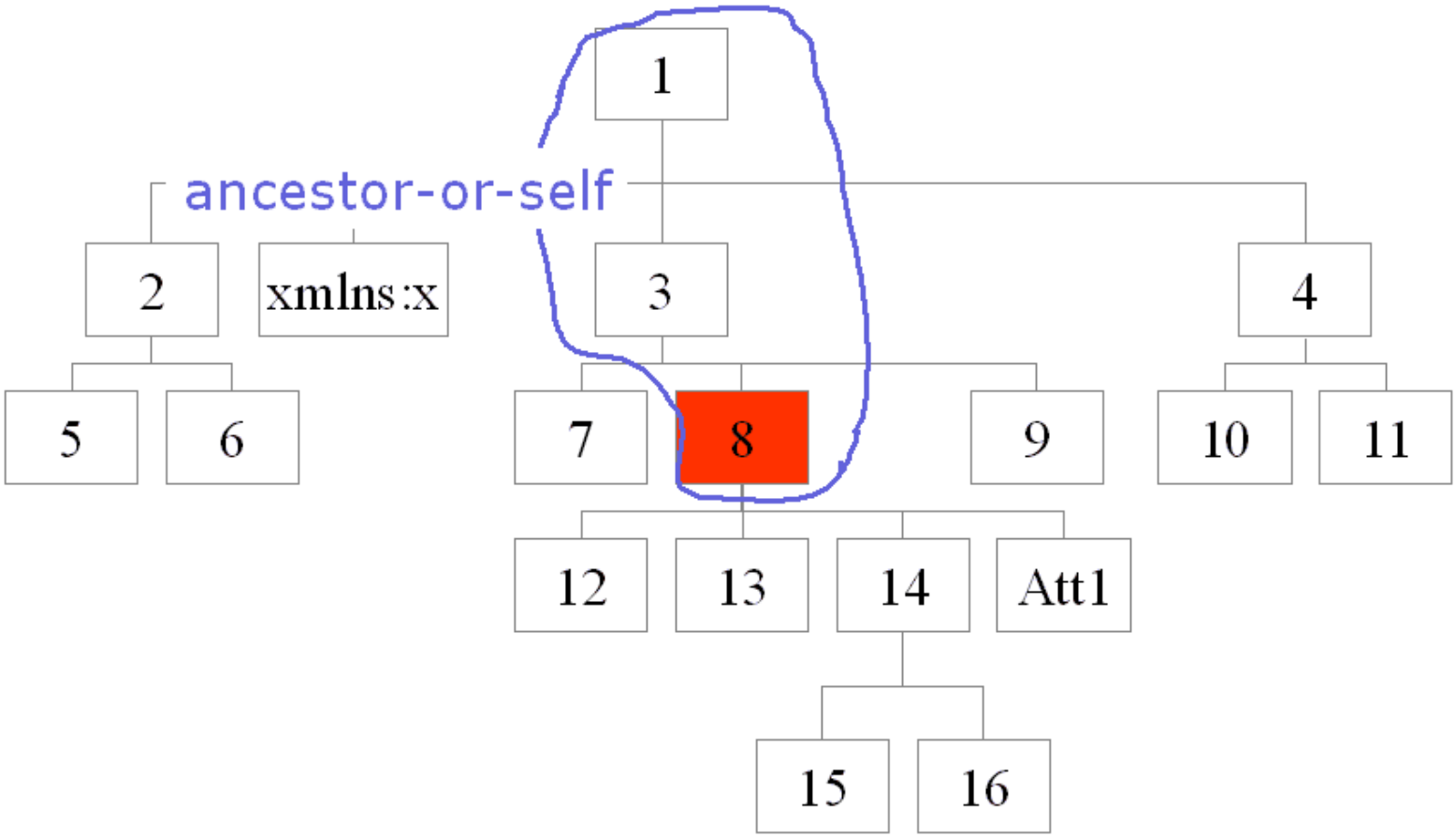

1
2
xmlns:x
3
4
5
6
7
8
9
10
11
child
12
13
14
Att1
15
16

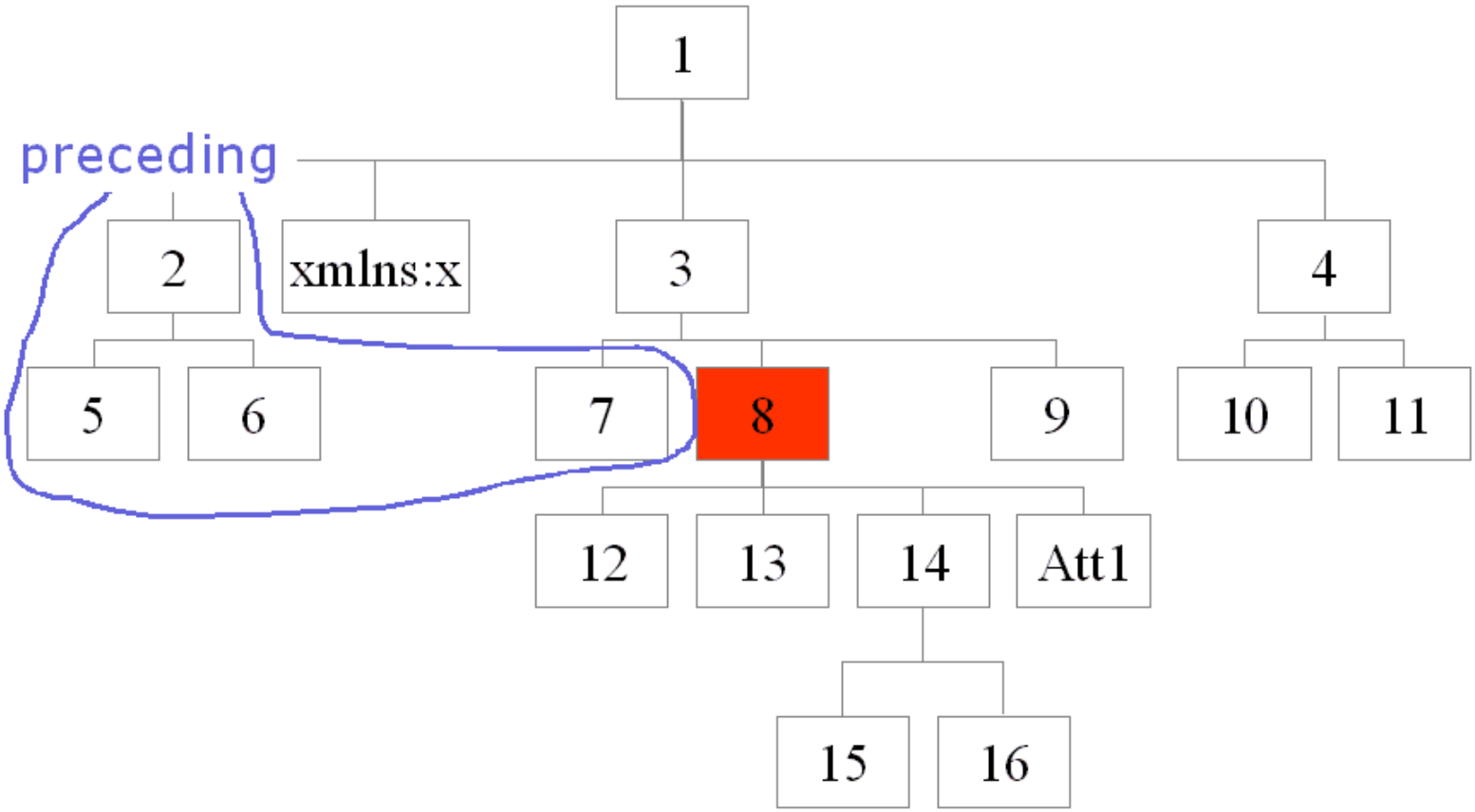

1
2
xmlns:x
3
4
5
6
7
8
9
10
11
descendant
12
13
14
Att1
15
16

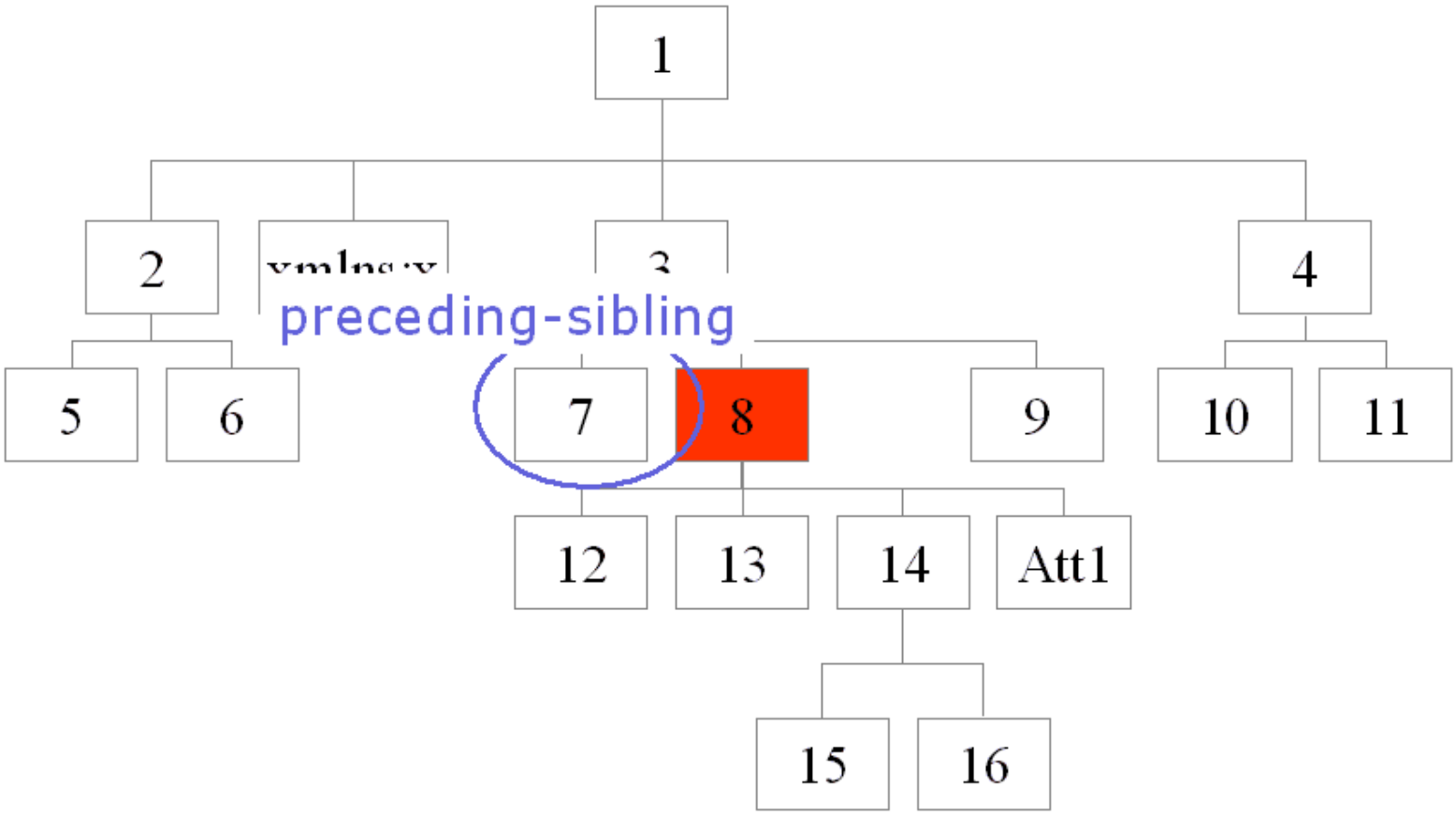

1
2
xmlns:x
3
4
descendant-or-self
5
6
7
8
9
10
11
12
13
14
Att1
15
16

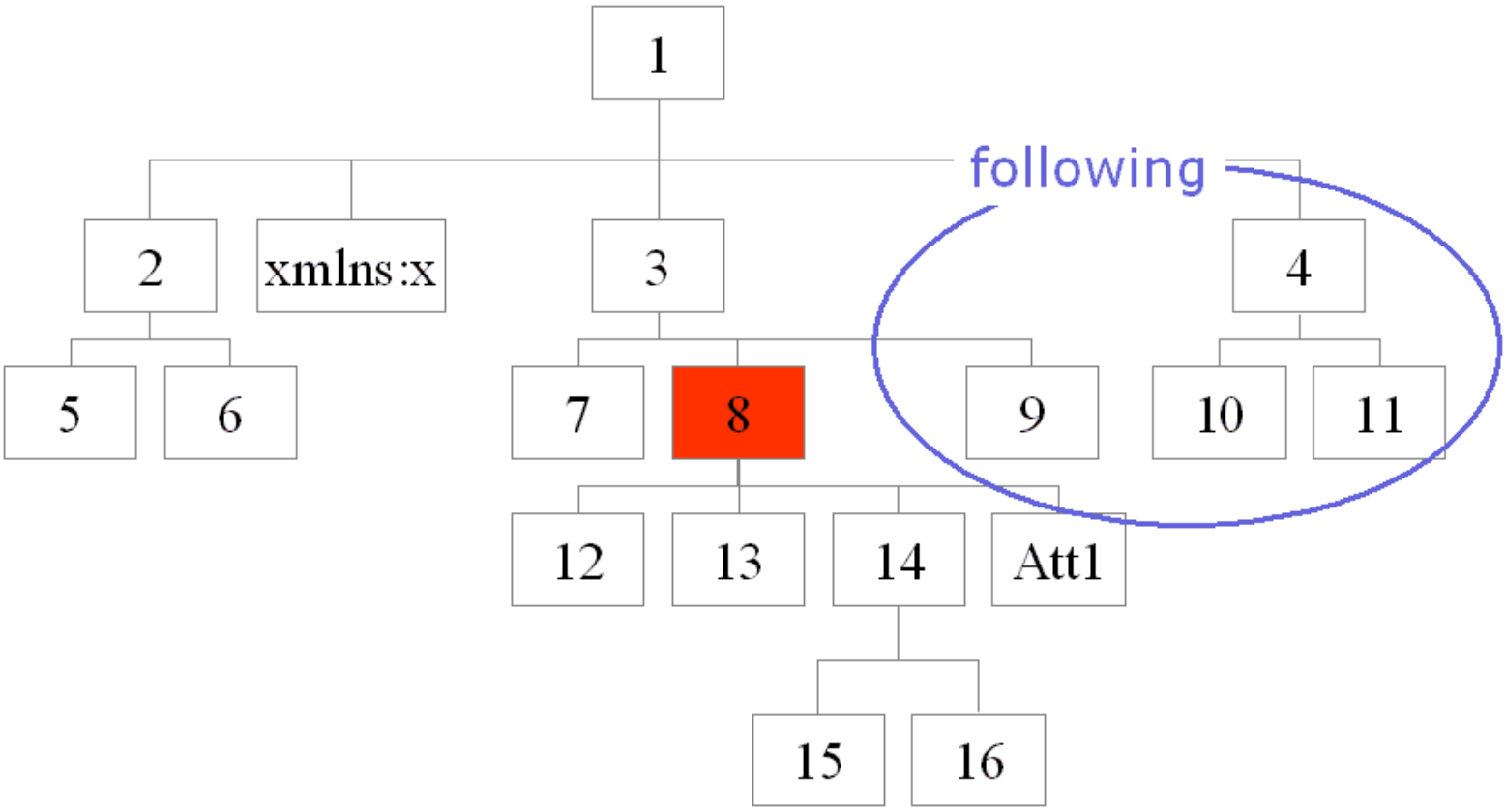

1
parent
2
xmlns:x
3
4
5
6
7
8
9
10
11
12
13
14
Att1
15
16

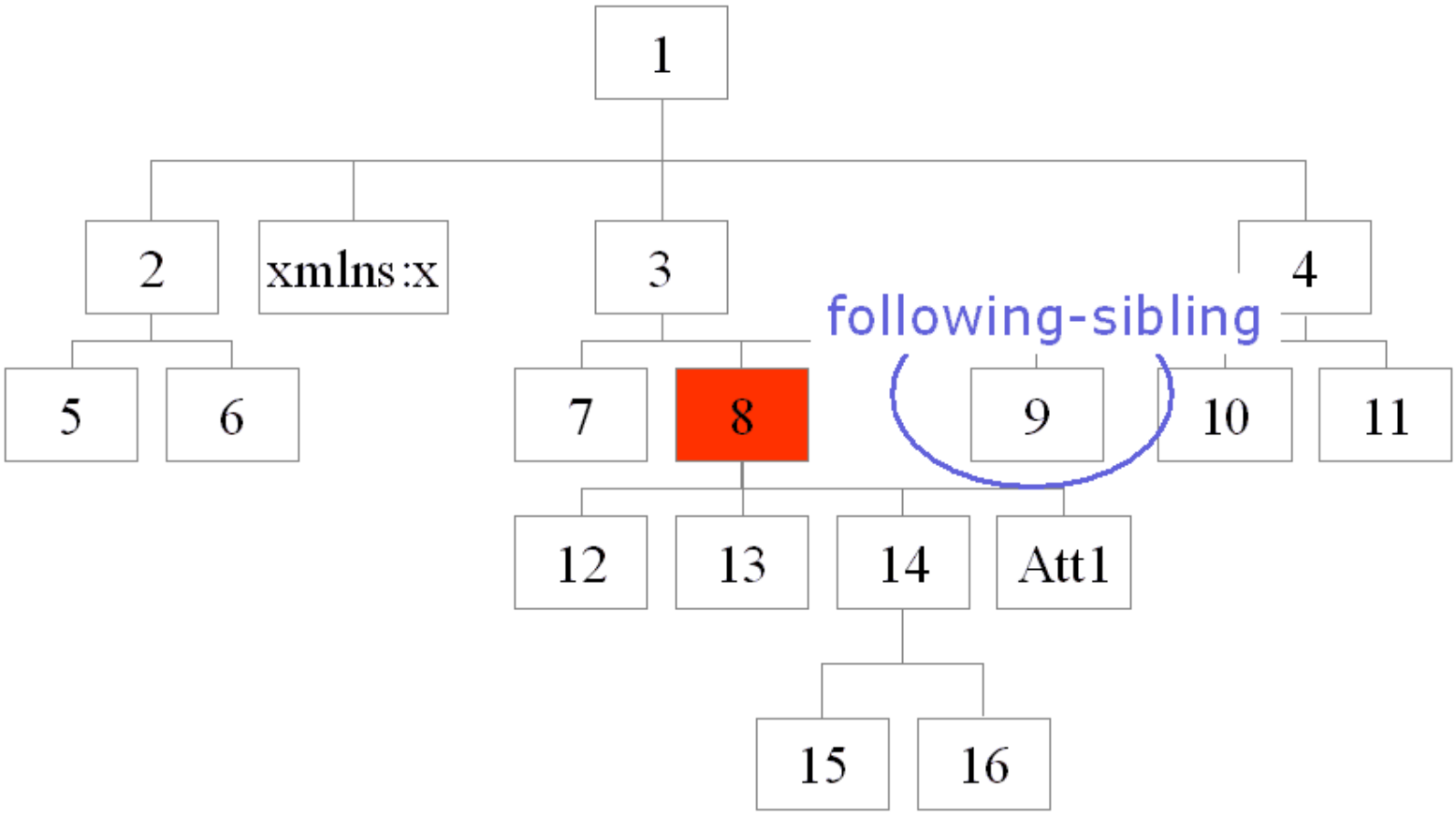

ancestor
1
2
xmlns:x
3
4
5
6
7
8
9
10
11
12
13
14
Att1
15
16

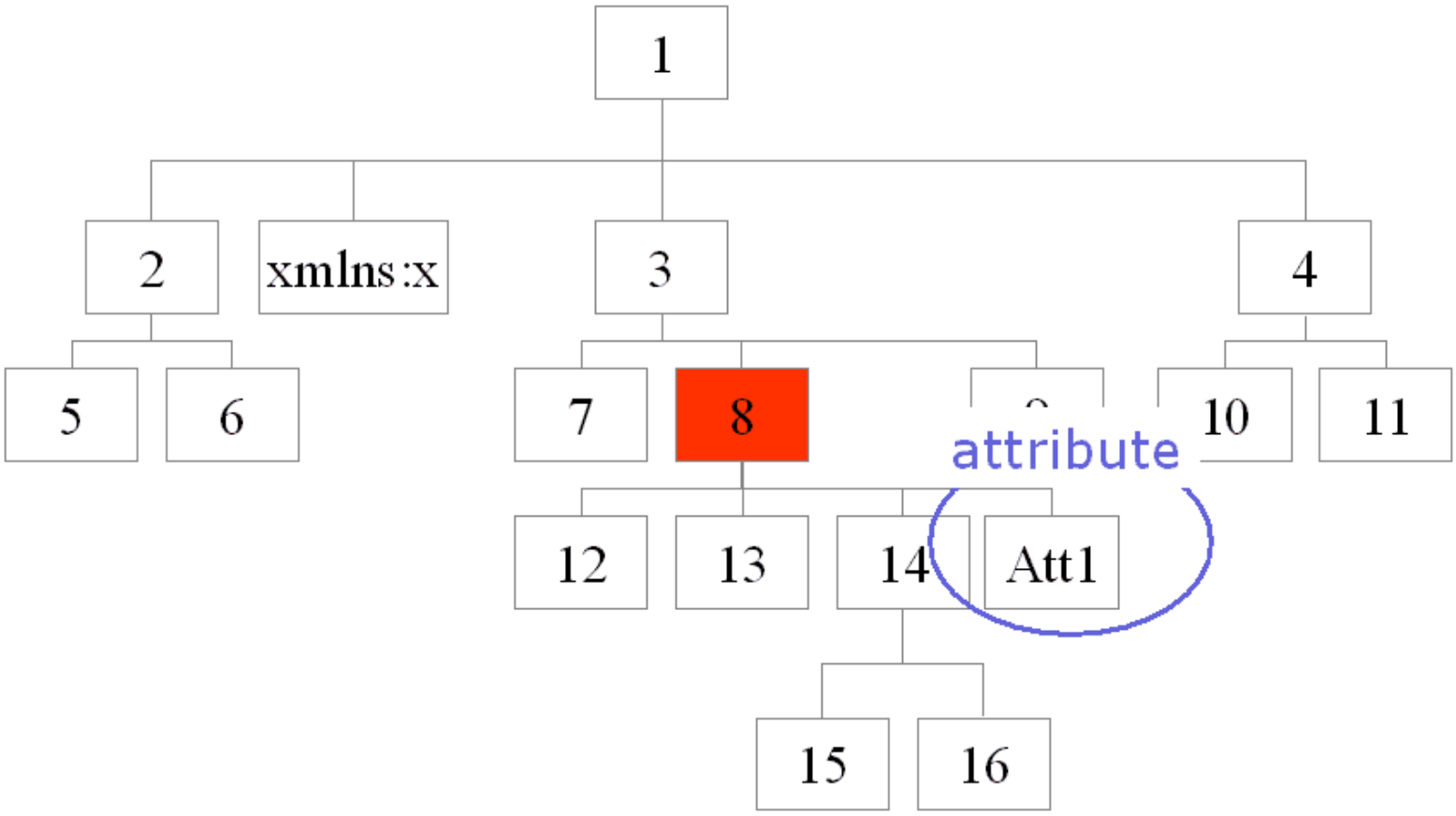

1
ancestor-or-self
2
xmlns:x
3
4
5
6
7
8
9
10
11
12
13
14
Att1
15
16

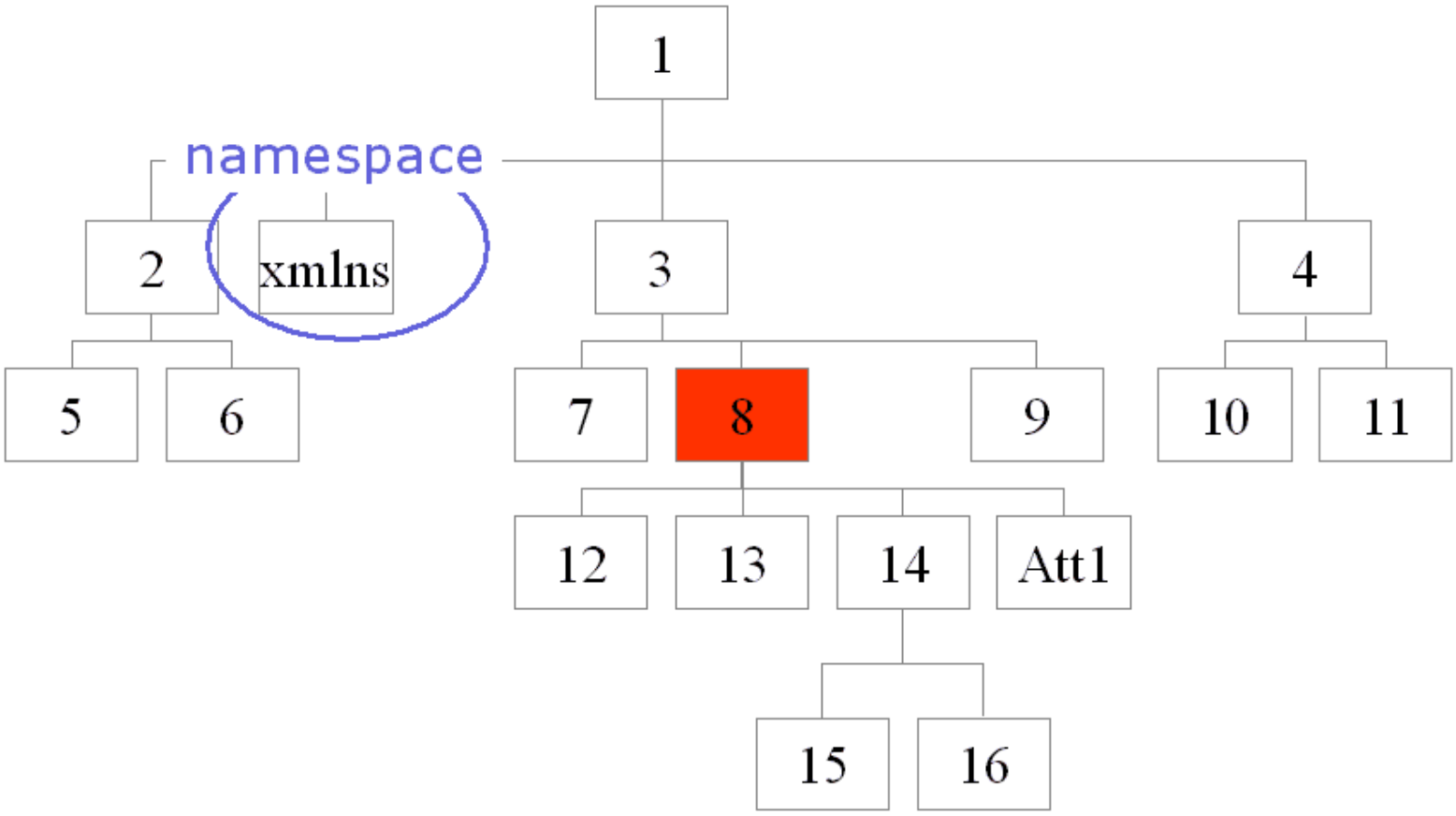

1
preceding
2
xmlns:x
3
4
5
6
7
8
9
10
11
12
13
14
Att1
15
16

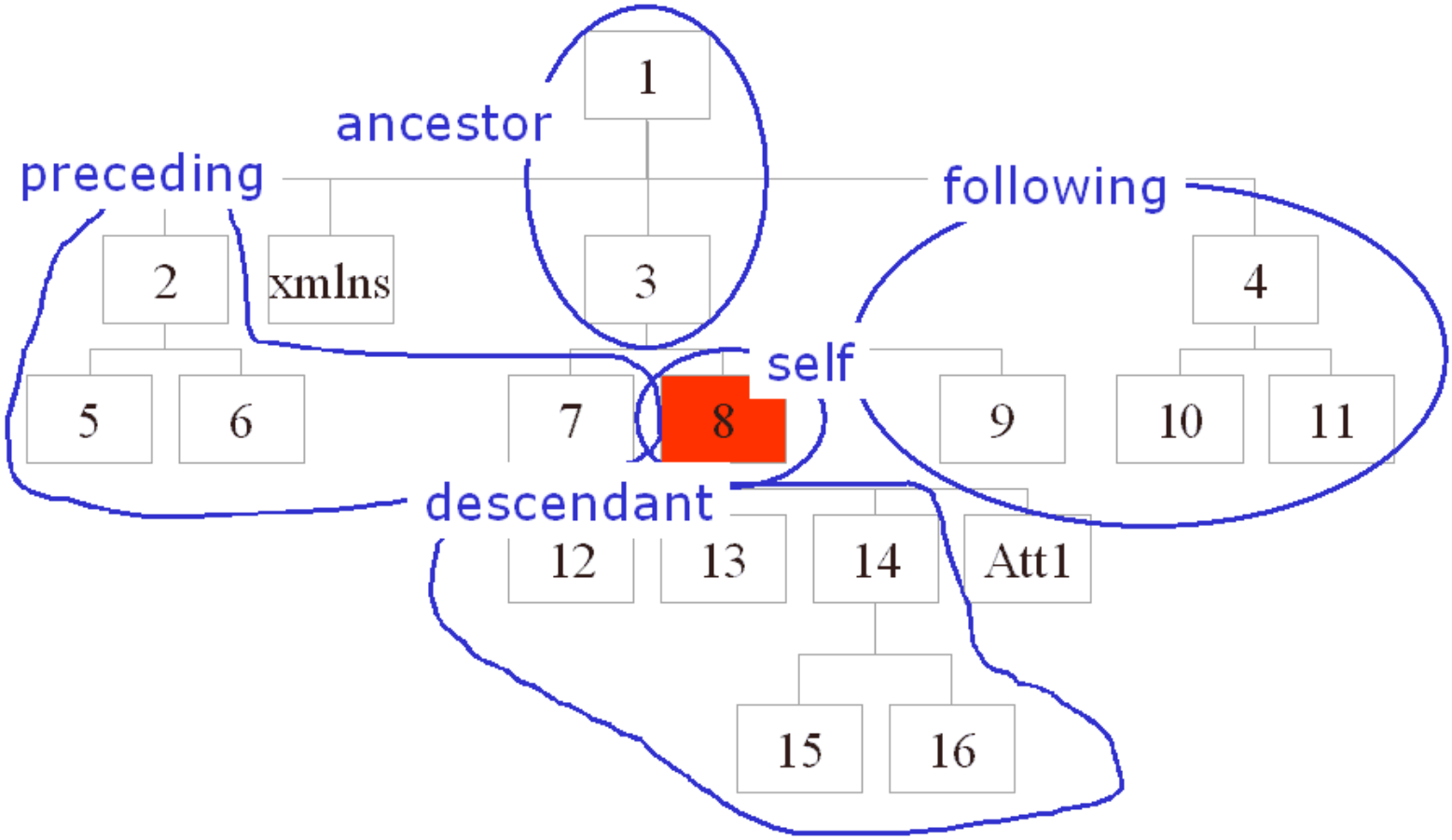

1
2
4
preceding-sibling
5
6
7
8
9
10
11
12
13
14
Att1
15
16

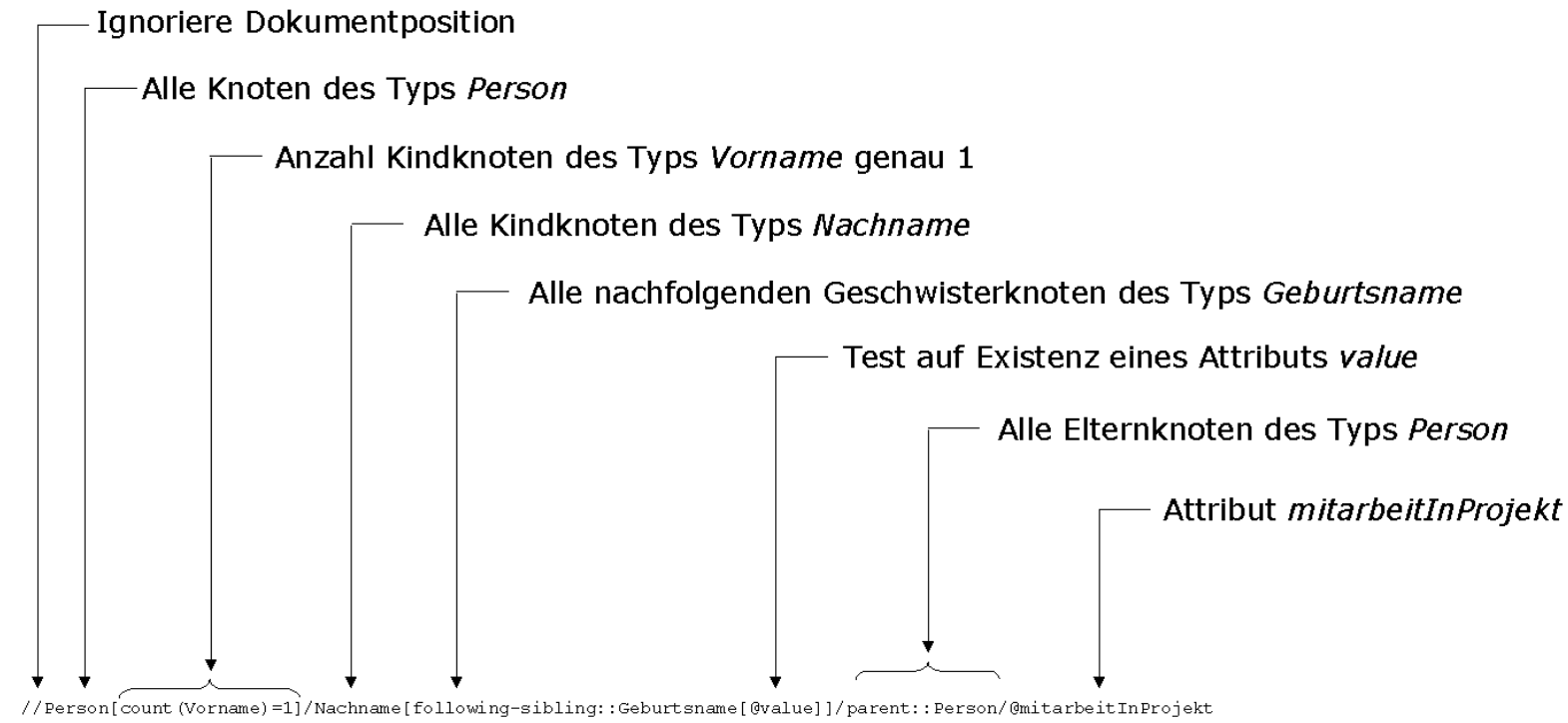

1
following
2
xmlns:x
3
4
5
6
7
8
9
10
11
12
13
14
Att1
15
16


1
2
xmlns:x
3
4
following-sibling
5
6
7
8
9
10
11
12
13
14
Att1
15
16


1
2
xmlns:x
3
4
5
6
7
8
10
11
attribute
12
13
14
Att1
15
16


1
namespace
2
xmlns
3
4
5
6
7
8
9
10
11
12
13
14
Att1
15
16


1
ancestor
preceding
following
2
xmlns
3
4
self
5
6
7
8
9
10
11
descendant
12
13
14
Att1
15
16


Ignoriere Dokumentposition
Alle Knoten des Typs *Person*
Anzahl Kindknoten des Typs *Vorname* genau 1
Alle Kindknoten des Typs *Nachname*
Alle nachfolgenden Geschwisterknoten des Typs *Geburtsname*
Test auf Existenz eines Attributs *value*
Alle Elternknoten des Typs *Person*
Attribut *mitarbeitInProjekt*
//Person[count(Vorname)=1]/Nachname[following-sibling::Geburtsname[@value]]/parent::Person/@mitarbeitInProjekt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
	xmlns:xsd="http://www.w3.org/2001/XMLSchema"
	elementFormDefault="qualified"
	attributeFormDefault="unqualified">
<xsd:element name="ProjektVerwaltung">
	<xsd:complexType>
		<xsd:sequence>
			<xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/>
			<xsd:element name="Projekt" type="ProjektType" maxOccurs="unbounded"/>
		</xsd:sequence>
		<xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
	</xsd:complexType>
	<xsd:unique name="uniquenessPersID">
		<xsd:selector xpath="Person"/>
		<xsd:field xpath="@PersID"/>
	</xsd:unique>
</xsd:element>

<xsd:complexType name="PersonType">
	<xsd:attribute name="PersID" type="xsd:token"/>
</xsd:complexType>

<xsd:complexType name="ProjektType"/>

</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
	xmlns:xsd="http://www.w3.org/2001/XMLSchema"
	elementFormDefault="qualified"
	attributeFormDefault="unqualified">
<xsd:element name="ProjektVerwaltung">
	<xsd:complexType>
		<xsd:sequence>
			<xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/>
			<xsd:element name="Projekt" type="ProjektType" maxOccurs="unbounded"/>
		</xsd:sequence>
		<xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
	</xsd:complexType>
	<xsd:unique name="uniquenessPersID">
		<xsd:selector xpath="Person"/>
		<xsd:field xpath="Vorname"/>
		<xsd:field xpath="Nachname"/>
	</xsd:unique>
</xsd:element>

<xsd:complexType name="PersonType">
	<xsd:sequence>
		<xsd:element name="Vorname" type="xsd:token" minOccurs="1"
maxOccurs="unbounded"/>
		<xsd:element name="Nachname" type="xsd:token" maxOccurs="1"/>
	</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ProjektType"/>

</xsd:schema>
```

XML Path Language (XPath) Version 1.0

Deutsche, kommentierte Übersetzung

26. Februar 2002

Dies ist die deutsche Übersetzung der W3C-Empfehlung "XML Path Language (XPath)" vom 16. November 1999. Bitte beachten Sie, dass dieses Dokument Übersetzungsfehler enthalten kann. Die normative englische Version des Dokuments befindet sich unter http://www.w3.org/TR/1999/REC-xpath-19991116

Bitte schicken Sie Fehler in dieser Übersetzung oder Verbesserungsvorschläge an den Übersetzer.

Diese Version:
http://www.obqo.de/w3c-trans/xpath-de-20020226
http://www.edition-w3c.de/TR/1999/REC-xpath-19991116
(verfügbar als XML oder HTML)
Aktuelle Version:
http://www.obqo.de/w3c-trans/xpath-de
http://www.edition-w3c.de/TR/xpath
Vorherige Version:
http://www.obqo.de/w3c-trans/xpath-de-20010910
Übersetzer:
Oliver Becker <ob@obqo.de>

XML Path Language (XPath) Version 1.0

Empfehlung des W3C, 16. November 1999

Diese Version:
http://www.w3.org/TR/1999/REC-xpath-19991116

(verfügbar als XML oder HTML)

Aktuelle Version:

http://www.w3.org/TR/xpath

Vorherige Versionen:

http://www.w3.org/TR/1999/PR-xpath-19991008
http://www.w3.org/1999/08/WD-xpath-19990813
http://www.w3.org/1999/07/WD-xpath-19990709
http://www.w3.org/TR/1999/WD-xslt-19990421

Herausgeber:

James Clark <jjc@jclark.com>
Steve DeRose (Inso Corp. and Brown University) <Steven_DeRose@Brown.edu>

Zusammenfassung

Die Sprache XPath dient zur Adressierung von Teilen eines XML-Dokuments. Sie wurde für die Verwendung sowohl in XSLT als auch in XPointer entworfen.

Status dieses Dokuments

Dieses Dokument wurde von Mitgliedern des W3C und anderen Interessierten geprüft und vom Direktor als W3C-Empfehlung gebilligt. Es ist ein abgeschlossenes Dokument und darf als Referenzmaterial verwendet oder als normative Referenz von einem anderen Dokument zitiert werden. Die Rolle des W3C bei der Erstellung dieser Empfehlung ist es, die Spezifikation bekannt zu machen und ihre breite Anwendung zu fördern. Dies erhöht die Funktionsfähigkeit und Interoperabilität des Web.

Die Liste der bekannten Fehler in dieser Spezifikation ist unter http://www.w3.org/1999/11/REC-xpath-19991116-errata verfügbar.

Anmerkungen zu dieser Spezifikation können an www-xpath-comments@w3.org geschickt werden; alle Anmerkungen sind in einem Archiv verfügbar.

Die englische Version dieser Spezifikation ist die einzig normative Version. Allerdings werden Übersetzungen dieses Dokuments unter http://www.w3.org/Style/XSL/translations.html aufgeführt.

Aktuelle W3C-Empfehlungen und weitere technische Dokumente sind unter http://www.w3.org/TR zu finden.

Diese Spezifikation ist das Ergebnis der gemeinsamen Arbeit der XSL- und der XML-Linking-Arbeitsgruppen und damit Teil der W3C Style Activity und der W3C XML Activity.

Inhaltsverzeichnis

Anhang

---

1 Einleitung

XPath ist das Ergebnis der Bemühungen, eine gemeinsame Syntax und Semantik für jene Funktionen bereitzustellen, die sowohl von XSL Transformations [XSLT] als auch von XPointer [XPointer] genutzt werden. Die primäre Aufgabe von XPath besteht in der Adressierung von Teilen eines XML-Dokuments [XML]. Zur Unterstützung dieser Aufgabe werden außerdem einfache Hilfsmittel für die Manipulation von Zeichenketten, Zahlen und booleschen Werten bereitgestellt. XPath benutzt eine kompakte Nicht-XML-Syntax, um die Verwendung von XPath-Ausdrücken innerhalb von URIs und XML-Attributen zu erleichtern. XPath operiert auf der abstrakten, logischen Struktur eines XML-Dokuments, nicht auf seiner äußerlichen Syntax. Seinen Namen erhält XPath durch die Verwendung einer auch in URLs genutzten Pfad-Notation (path), mit der sich durch die hierarchische Struktur eines XML-Dokuments navigieren lässt.

Neben der Verwendung für die Adressierung wurde XPath so gestaltet, dass eine natürliche Teilmenge davon zum Matching (Testen, ob ein Knoten auf ein Muster passt) genutzt werden kann. Diese Anwendung von XPath ist in XSLT beschrieben.

Anmerkung des Übersetzers:

Diese Teilmenge wird in XSLT Muster (pattern) genannt. Obwohl diese Muster durch eine eigene Grammatik definiert werden, ist jedes Muster auch ein XPath-Ausdruck.

Neben XSLT und XPointer existieren weitere Spezifikationen, die XPath nutzen. Als Beispiel sei hier der Arbeitsentwurf des W3C für die XML-Abfragesprache XQuery [XQuery] genannt, deren Syntax erweiterte XPath-Ausdrücke verwendet.

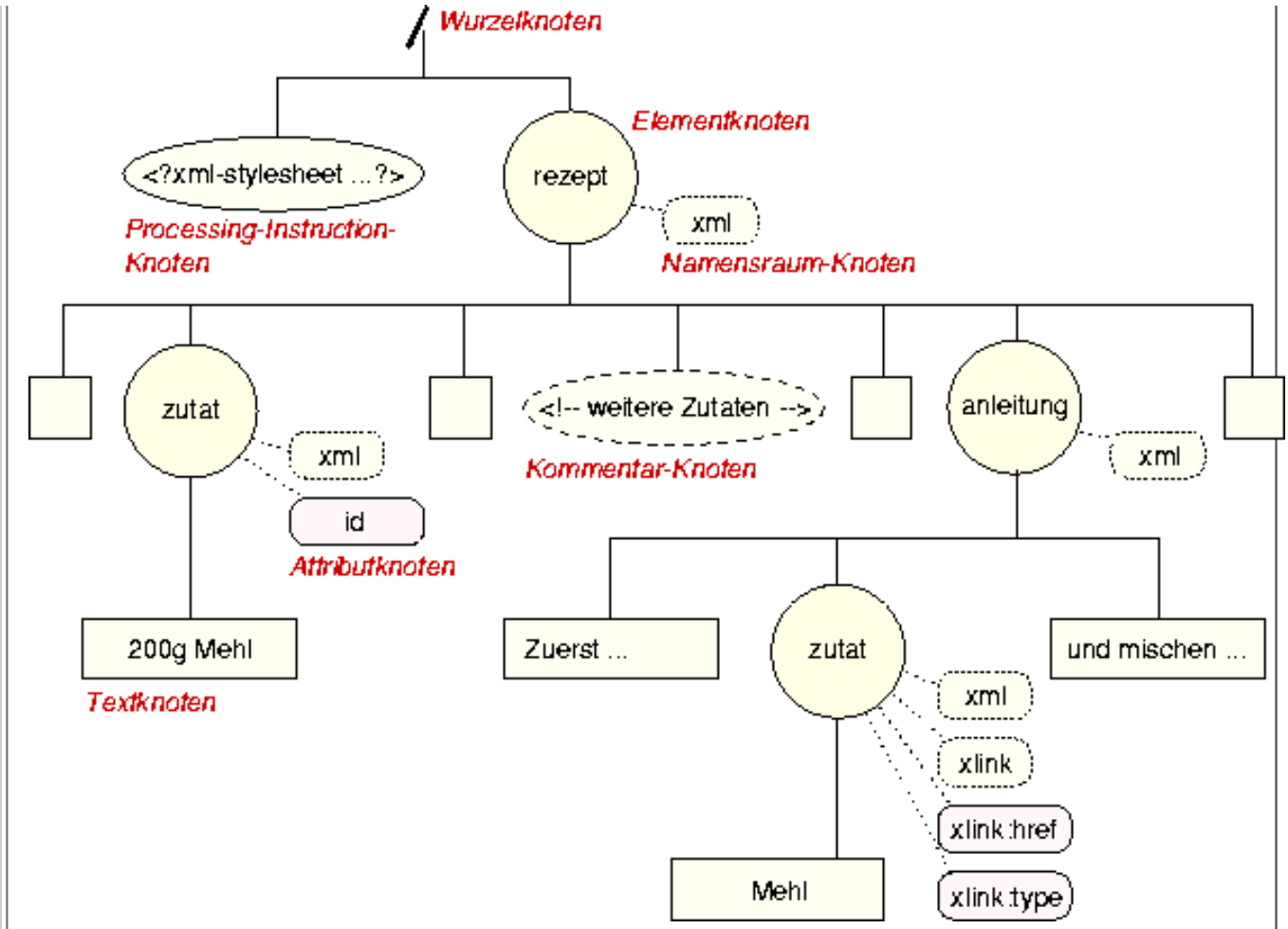
XPath modelliert ein XML-Dokument als einen Baum, der aus Knoten besteht. Es gibt verschiedene Knotentypen, unter anderem Elementknoten, Attributknoten und Textknoten. XPath definiert, wie der Zeichenkettenwert für jeden Knotentyp berechnet wird. Einige Knoten besitzen zusätzlich einen Namen. XPath unterstützt in vollem Umfang XML-Namensräume [XML Names]. Daher wird der Name eines Knotens als ein Paar aus einem lokalen Bestandteil und einem gegebenenfalls leeren Namensraum-URI modelliert – dieses wird erweiterter Name genannt. Das Datenmodell ist detailliert in [**5 Datenmodell**] beschrieben.

Anmerkung des Übersetzers:

Zur Veranschaulichung der durch XPath modellierten Baumstruktur soll folgendes Beispiel dienen:

```
<?xml version="1.0"?>
<!DOCTYPE rezept SYSTEM "rezept.dtd">
<?xml-stylesheet href="style.xsl" type="text/xml"?>
<rezept>
   <zutat id="mehl">200g Mehl</zutat>
   <!-- weitere Zutaten -->
   <anleitung>
      Zuerst nehmen Sie das
      <zutat xmlns:xlink="http://www.w3.org/1999/xlink"
             xlink:type="simple" xlink:href="mehl">Mehl</zutat>
      und mischen es mit ...
   </anleitung>
</rezept>
```

Dieses recht kurze XML-Dokument besitzt bereits eine aus 23 Knoten bestehende Baumrepräsentation:



Die XML-Deklaration sowie die Dokumenttyp-Deklaration finden sich in der Baumdarstellung nicht wieder, auf sie kann über einen XPath-Ausdruck auch nicht zugegriffen werden. Alle anderen Bestandteile des Dokuments (Wurzel, Elemente, Textinhalt, Attribute, Namensräume, Kommentare, Processing Instructions) werden durch entsprechende Knoten des Baumes repräsentiert. Die leeren Quadrate symbolisieren Textknoten, die ausschließlich Leerraumzeichen enthalten. Inwieweit sich solcher Leerraum in eigenen Textknoten wiederfindet, wird von der XPath nutzenden Anwendung bestimmt. Schließlich sei auf die implizit zu jedem Element gehörenden Namensraumknoten für den `xml`-Namensraum hingewiesen.

Das Document Object Model [DOM] definiert ebenfalls eine Baumrepräsentation, die allerdings in einigen Punkten von der in XPath verwendeten abweicht. Auf diese Unterschiede wird innerhalb des Kapitels [**5 Datenmodell**] eingegangen.

Das primäre syntaktische Konstrukt in XPath ist der Ausdruck. Ein Ausdruck lässt sich aus der Produktion Expr ableiten. Die Auswertung eines Ausdrucks ergibt ein Objekt, das zu einem der folgenden vier Grundtypen gehört:

- *node-set* (eine ungeordnete Menge von Knoten ohne Duplikate)
- *boolean* (*wahr* oder *falsch*)
- *number* (eine Gleitkommazahl)
- *string* (eine Zeichenkette bestehend aus UCS-Zeichen)

Anmerkung des Übersetzers:

Das sind die vier Grundtypen, die XPath definiert. Tatsächlich können darauf aufbauende Spezifikationen weitere Typen definieren, sodass die Auswertung eines XPath-Ausdrucks ein Objekt dieses neuen Typs ergeben kann. Insbesondere können im Ausdruck enthaltene Variablen Objekte anderer Typen aufnehmen.

Wie man sieht, gibt es keinen Datentyp für Knoten. Ein einzelner Knoten kann aber als Knotenmenge (*node-set*) dargestellt werden, die genau ein Element enthält.

Die innerhalb von Zeichenketten erlaubten UCS-Zeichen (Universal Multiple-Octet Coded Character Set) sind in [ISO/IEC 10646] bzw. [ISO/IEC 10646, 2nd Edition] beschrieben.

Ein Ausdruck wird immer bezüglich eines Kontextes ausgewertet. XSLT und XPointer spezifizieren, wie dieser Kontext bei der Verwendung von XPath-Ausdrücken in XSLT bzw. XPointer bestimmt wird. Der Kontext besteht aus:

- einem Knoten (dem **Kontextknoten**)
- einem Paar von positiven ganzen Zahlen ungleich 0 (der **Kontextposition** und der **Kontextgröße**)
- einer Menge von Variablenbelegungen
- einer Funktionsbibliothek
- den Namensraumdeklarationen, in deren Gültigkeitsbereich der Ausdruck liegt

Die Kontextposition ist immer kleiner oder gleich der Kontextgröße.

Die Variablenbelegungen bestehen aus einer Abbildung von Variablennamen auf Variablenwerte. Der Wert einer Variablen ist ein Objekt, welches von jedem beliebigen Typ sein kann, der für Ausdrücke möglich ist. Daneben sind auch weitere Typen möglich, die hier nicht spezifiziert werden.

Anmerkung des Übersetzers:

In der XSLT-1.0-Spezifikation [XSLT] wird beispielsweise als neuer Typ *Ergebnisteilbaum* (*result tree fragment*) eingeführt. Werte dieses Typs entstehen im Ergebnis des Transformationsprozesses. Durch die folgende Variablenvereinbarung wird z.B. eine Variable namens `antwort` erzeugt, deren Wert ein *Ergebnisteilbaum* ist:

```
<xsl:variable name="antwort">
   <antwort><xsl:value-of select="6 * 7" /></antwort>
</xsl:variable>
```

Dieser Typ wird voraussichtlich in zukünftigen XSLT-Versionen nicht mehr existieren, da das Ergebnis einer Transformation dann eine normale Knotenmenge vom Typ *node-set* sein wird, siehe [XSLT 2.0].

Die folgende Anweisung erzeugt dagegen eine Variable, deren Wert sich aus der Berechnung eines XPath-Ausdrucks *expression* ergibt:

```
<xsl:variable name="var" select="expression" />
```

An dieser Stelle sei darauf hingewiesen, dass die Erzeugung der Variablenbelegungen für

antwort und var ein XSLT-Sprachmittel ist. XPath selbst stellt hierfür keinerlei Konstrukte bereit. Neben xsl:variable besitzt XSLT für diesen Zweck xsl:param.

Die XPointer-Spezifikation [XPointer], die ebenfalls XPath nutzt, sieht keine Möglichkeiten für das Erzeugen von Variablenbelegungen vor. Die Verwendung von Variablenreferenzen innerhalb eines XPointer-Ausdrucks führt daher zu einem syntaktischen Fehler. Die als Arbeitsentwurf des W3C vorgelegte Abfragesprache für XML-Dokumente XQuery [XQuery] benutzt zur Erzeugung von Variablenbelegungen so genannte FLWR-Ausdrücke (gesprochen "flower", eine Abkürzung für FOR-, LET-, WHERE- und RETURN-Klauseln).

Andere auf XPath aufbauende Spezifikationen müssen in analoger Weise definieren, wie Variablenbelegungen für einen Kontext erzeugt werden.

Die Funktionsbibliothek besteht aus einer Abbildung von Funktionsnamen auf Funktionen. Jede Funktion besitzt null oder mehr Argumente und liefert einen einzelnen Wert. Diese Spezifikation definiert eine Bibliothek von Grundfunktionen, die von allen XPath-Implementationen unterstützt werden muss (siehe [**4 Bibliothek der Grundfunktionen**]). Bei einer Grundfunktion gehören die Argumente und das Ergebnis einem der vier Grundtypen an. Sowohl XSLT als auch XPointer erweitern XPath um zusätzliche Funktionen, von denen einige auf den vier Grundtypen, andere auf zusätzlichen, durch XSLT und XPointer definierten Typen operieren.

Namensraumdeklarationen bestehen aus einer Abbildung von Präfixen auf Namensraum-URIs.

Anmerkung des Übersetzers:

Angenommen, ein Element enthält in seinem Start-Tag folgende Namensraumdeklaration: xmlns:xlink="http://www.w3.org/1999/xlink". Ein Kontext, für den diese Deklaration gültig ist, enthält dann eine Abbildung des Präfixes xlink auf den URI http://www.w3.org/1999/xlink.

Die Variablenbelegungen, die Funktionsbibliothek und die Namensraumdeklarationen, die benutzt werden, um einen Teilausdruck zu berechnen, sind immer dieselben, die auch für den umgebenden Ausdruck benutzt werden. Der Kontextknoten, die Kontextposition und die Kontextgröße, die zur Berechnung eines Teilausdrucks benutzt werden, sind dagegen zuweilen verschieden von denen des umgebenden Ausdrucks. Mehrere Arten von Ausdrücken ändern den Kontextknoten, aber nur Prädikate ändern die Kontextposition und die Kontextgröße (siehe [**2.4 Prädikate**]). Bei der Beschreibung, wie bestimmte Ausdrücke zu berechnen sind, wird immer explizit angegeben, ob sich der Kontextknoten, die Kontextposition oder die Kontextgröße bei der Berechnung von Teilausdrücken ändert. Wird nichts über Kontextknoten, Kontextposition und Kontextgröße ausgesagt, bleiben sie bei der Berechnung von Teilausdrücken dieser Ausdrücke gleich.

Anmerkung des Übersetzers:

Zur Erklärung hier ein kleiner Vorgriff:

Innerhalb eines Lokalisierungspfades ändert jeder Schritt den Kontextknoten, der für die Berechnung der folgenden Schritte relevant ist. Durch ein Prädikat werden Knoten aus einer Knotenmenge herausgefiltert, sodass die Kontextgröße sich in der Regel verkleinert und die Position der gefilterten Knoten sich entsprechend ändert.

XPath-Ausdrücke erscheinen häufig in XML-Attributen. Die in diesem Abschnitt spezifizierte Grammatik wird auf Attributwerte nach ihrer Normalisierung gemäß XML 1.0 angewendet. Wenn beispielsweise die Grammatik das Zeichen < verwendet, darf dieses nicht in der XML-Quelle als < auftreten, sondern muss gemäß den XML-1.0-Regeln notiert werden, zum Beispiel durch Eingabe als `<`. Innerhalb von Ausdrücken werden Zeichenkettenliterale durch einfache oder doppelte Anführungszeichen begrenzt, die ebenfalls zur Begrenzung von XML-Attributen verwendet werden. Um zu vermeiden, dass ein Anführungszeichen innerhalb eines Ausdrucks durch den XML-Prozessor als Abschluss des Attributwertes interpretiert wird, kann das Anführungszeichen als Zeichenreferenz eingegeben werden (`&quot;` oder `&apos;`). Alternativ können im Ausdruck einfache Anführungszeichen benutzt werden, falls das XML-Attribut durch doppelte Anführungszeichen begrenzt wird oder umgekehrt.

Anmerkung des Übersetzers:

Statt "Zeichenreferenz" muss es hier "Entity-Referenz" heißen.

Abgesehen davon gibt es immer noch Fälle, in denen dieses einfache Kochrezept nicht ausreichend ist – nämlich dann, wenn eine Zeichenkette benötigt wird, die beide Arten von Anführungszeichen enthalten soll.

Angenommen, es soll getestet werden, ob der Inhalt des Elements `para` mit der Zeichenkette »Sie fragte: "Wie geht's?"« übereinstimmt. Möchte man diese Zeichenkette innerhalb eines XPath-Ausdrucks durch doppelte Anführungszeichen begrenzen, könnte man auf die Idee kommen, beispielsweise durch Verwendung der Entity-Referenz `&quot;` Folgendes zu schreiben:

```
para="Sie fragte: &quot;Wie geht's?&quot;"
```

Das ist aber keine Lösung, da ein XML-Parser, der diese Zeichenfolge analysiert, Entity-Referenzen auflöst. Ein darauf aufsetzender XPath-Prozessor kann nicht mehr zwischen »"« und »`&quot;`« unterscheiden. Formuliert man im nächsten Schritt dann nämlich weiter

```
<xsl:if test='para="Sie fragte: &quot;Wie geht&apos;s?&quot;"'>
```

(diese Zeile ist im XML-Sinn wohlgeformt), so führt das zu einem Fehler im XPath-Prozessor, da dieser trotzdem die folgende Zeichenkette auswertet:

```
para="Sie fragte: "Wie geht's?""
```

Eine einfache Lösung für solche Fälle besteht darin, die betreffende Zeichenkette als Wert einer Variablen zu definieren, etwa in XSLT per

```
<xsl:variable name="string">Sie fragte: "Wie geht's?"</xsl:variable>
```

und anschließend diese Variable in XPath-Ausdrücken zu verwenden.

```
<xsl:if test="para=$string">
```

Alternativ kann man die gewünschte Zeichenkette aus mehreren Teilen zusammensetzen (unter Benutzung der noch zu erläuternden Funktion **concat**), wobei jede Teilzeichenkette jeweils nur eine Sorte von Anführungszeichen enthält:

```
para=concat('Sie fragte: "Wie geht', "'", 's?"')
```

Das Anführungszeichen, das zur Begrenzung dieses Ausdrucks innerhalb eines Attributwertes genutzt wird, muss dann durch die dazugehörige Entity-Referenz ersetzt werden. Das Ergebnis sieht zwar unübersichtlich aus, ist aber syntaktisch korrekt:

```
<xsl:if test="para=concat('Sie fragte: &quot;Wie geht',
                          &quot;'&quot;, 's?&quot;')">
```

Diese zweite Variante muss verwendet werden, wenn Variablen nicht erlaubt sind, etwa innerhalb eines XSLT-Musters oder als Bestandteil eines XPointers.

Ein wichtiger spezieller Ausdruck ist der Lokalisierungspfad. Ein Lokalisierungspfad wählt eine Knotenmenge relativ zu einem Kontextknoten aus. Das Ergebnis der Berechnung eines Ausdrucks, der ein Lokalisierungspfad ist, ist genau die Knotenmenge, die die durch den Lokalisierungspfad ausgewählten Knoten enthält. Lokalisierungspfade können Ausdrücke rekursiv enthalten, die zum Filtern von Knotenmengen benutzt werden. Ein Lokalisierungspfad lässt sich aus der Produktion LocationPath ableiten.

Die in der nachfolgenden Grammatik verwendeten Nichtterminale QName und NCName sind in [XML Names] definiert, S ist in [XML] definiert. Die Grammatik verwendet die gleiche EBNF-Notation wie in [XML] (mit der Ausnahme, dass Grammatiksymbole immer mit einem Großbuchstaben beginnen).

Anmerkung des Übersetzers:

Die *Erweiterte Backus-Naur-Form (EBNF)* wird zur Notation von formalen Grammatiken verwendet. Sie wird im Kapitel Notation der XML-Spezifikation [XML, 2nd Edition] näher erläutert.

Ausdrücke werden geparst, indem die Zeichenfolge in einzelne Tokens zerlegt und anschließend die entstehende Folge der Tokens geparst wird. Leerraumzeichen können beliebig zwischen Tokens verwendet werden. Der Zerlegungsprozess wird in **[3.7 Lexikalische Struktur]** beschrieben.

Anmerkung des Übersetzers:

Ein Token ist eine syntaktische Einheit von Zeichen, etwa der Name »`html`«, die Zahl »`3.14159`« oder der Operator »`!=`«. Die zwischen diesen Tokens erlaubten Leerraumzeichen sind gemäß der XML-Spezifikation Folgen aus Leerzeichen (#x20), Tabulatoren (#x9), Zeilenvorschüben (#xA) und Wagenrückläufen (#xD). Innerhalb eines Tokens dürfen keine solchen Leerraumzeichen auftreten, so z.B. nicht zwischen den beiden einzelnen Zeichen »`!`« und »`=`« des Operators »`!=`«.

2 Lokalisierungspfade

Obwohl Lokalisierungspfade nicht das allgemeinste grammatische Konstrukt der Sprache darstellen (ein Lokalisierungspfad ist ein Spezialfall eines Ausdrucks), sind sie doch das wichtigste Konstrukt und werden deshalb als Erstes beschrieben.

Jeder Lokalisierungspfad kann durch eine unkomplizierte und eher verbale Syntax ausgedrückt werden. Daneben gibt es eine Reihe von syntaktischen Abkürzungen, mit denen sich häufige Fälle kurz und prägnant ausdrücken lassen. Dieser Abschnitt erläutert die Semantik von Lokalisierungspfaden anhand der ausführlichen Syntax. Die abgekürzte Syntax wird im Anschluss daran erläutert, indem gezeigt wird, wie diese auf die ausführliche Syntax abgebildet wird (siehe [**2.5 Abgekürzte Syntax**]).

Es folgen einige Beispiele für Lokalisierungspfade unter Benutzung der ausführlichen Syntax:

- `child::para` wählt die Kindelemente `para` des Kontextknotens aus.
- `child::*` wählt alle Kindelemente des Kontextknotens aus.
- `child::text()` wählt alle Textknoten aus, die Kinder des Kontextknotens sind.
- `child::node()` wählt alle Kindknoten des Kontextknotens aus, unabhängig von ihrem Knotentyp.

  Anmerkung des Übersetzers:

  Obwohl sich sämtliche XPath-Lokalisierungspfade für Kontextknoten jedes Typs anwenden lassen, muss an dieser Stelle darauf hingewiesen werden, dass nur der Wurzelknoten sowie Elementknoten Kinder haben können. Diese Kinder können Element-, Text-, Kommentar- und Processing-Instruction-Knoten sein. Attribut- und Namensraumknoten sind niemals Kinder anderer Knoten. Dieses Konzept wird noch einmal ausführlich in Kapitel [**5 Datenmodell**] beschrieben.

- `attribute::name` wählt das Attribut `name` des Kontextknotens aus.
- `attribute::*` wählt alle Attribute des Kontextknotens aus.

Anmerkung des Übersetzers:

Diese beiden Ausdrücke sind nur sinnvoll, wenn der Kontextknoten ein Elementknoten ist. Für alle anderen Knotentypen liefern sie die leere Knotenmenge.

- `descendant::para` wählt die `para`-Elemente aus, die Nachkommen des Kontextknotens sind.
- `ancestor::div` wählt alle `div`-Elemente aus, die Vorfahren des Kontextknotens sind.
- `ancestor-or-self::div` wählt alle `div`-Vorfahren des Kontextknotens sowie auch den Kontextknoten selbst aus, falls dieser ein `div`-Element ist.
- `descendant-or-self::para` wählt alle `para`-Nachkommen des Kontextknotens aus, sowie auch den Kontextknoten selbst, falls dieser ein `para`-Element ist.
- `self::para` wählt den Kontextknoten aus, falls dieser ein `para`-Element ist, und sonst nichts.
- `child::chapter/descendant::para` wählt die `para`-Elemente aus, die Nachkommen der `chapter`-Kindelemente des Kontextknotens sind.
- `child::*/child::para` wählt alle `para`-Enkelelemente des Kontextknotens aus.
- `/` wählt die Wurzel des Dokuments aus (diese ist immer der Vater des Dokumentelements).

Anmerkung des Übersetzers:

Der Begriff *Wurzel* unterscheidet sich hier von dem in der XML-Spezifikation [XML, 2nd Edition] genutzten. Dort bezeichnen Wurzel und Dokumentelement das Gleiche. In der XPath-Terminologie ist das Dokumentelement ein Kind des Wurzelknotens. Der Wurzelknoten entspricht damit dem *Dokument-Entity*, siehe [**5.1 Wurzelknoten**]. Das ist insofern wichtig, als der Wurzelknoten zusätzlich Kommentar- oder Processing-Instruction-Knoten als Kinder haben kann, die damit Geschwister des Dokumentelements sind.

Des Weiteren muss hier hinzugefügt werden, dass durch `/` die Wurzel des Dokuments ausgewählt wird, in dem sich der Kontextknoten befindet. Es lässt sich leicht eine Knotenmenge bilden, die Knoten aus verschiedenen Dokumenten enthält. Die in XSLT definierte Funktion **document** liefert beispielsweise den Wurzelknoten eines anderen Dokuments.

- `/descendant::para` wählt alle `para`-Elemente aus, die im gleichen Dokument wie der Kontextknoten enthalten sind.

> **Anmerkung des Übersetzers:**
>
> Genau genommen werden alle `para`-Elemente ausgewählt, die Nachkommen der Wurzel des Dokuments sind, zu dem der Kontextknoten gehört.

- `/descendant::olist/child::item` wählt alle `item`-Elemente aus, die ein `olist`-Vaterelement besitzen und die sich im gleichen Dokument wie der Kontextknoten befinden.

> **Anmerkung des Übersetzers:**
>
> Die folgenden Beispiele zeigen Lokalisierungspfade, bei denen die ursprünglich ausgewählten Knotenmengen durch nachgestellte Filterausdrücke in eckigen Klammern, so genannte Prädikate, weiter eingeschränkt werden.

- `child::para[position()=1]` wählt das erste `para`-Kindelement des Kontextknotens aus.
- `child::para[position()=last()]` wählt das letzte `para`-Kindelement des Kontextknotens aus.
- `child::para[position()=last()-1]` wählt das vorletzte `para`-Kindelement des Kontextknotens aus.
- `child::para[position()>1]` wählt alle `para`-Kindelemente des Kontextknotens aus, mit Ausnahme des ersten `para`-Kindelements des Kontextknotens.
- `following-sibling::chapter[position()=1]` wählt das nächste `chapter`-Geschwisterelement des Kontextknotens aus.
- `preceding-sibling::chapter[position()=1]` wählt das vorhergehende `chapter`-Geschwisterelement des Kontextknotens aus.

> **Anmerkung des Übersetzers:**
>
> An dieser Stelle sei als kurzer Vorgriff darauf hingewiesen, dass sich die Position eines Knotens in der aktuellen Kontextknotenliste über die Funktion **position** bestimmen lässt und diese Position offenbar von der Blickrichtung abhängt. Von allen vorhergehenden Geschwisterelementen befindet sich der unmittelbare Vorgänger an der Position 1.

- `/descendant::figure[position()=42]` wählt das zweiundvierzigste `figure`-Element im Dokument aus.

Anmerkung des Übersetzers:

Hier werden alle `figure`-Nachkommen des Wurzelknotens betrachtet. Ausgehend von ihrer natürlichen Reihenfolge im Dokument wird aus der gesamten Menge das zweiundvierzigste ausgewählt. Die Definition dieser natürlichen Reihenfolge, der so genannten Dokumentordnung, ist Bestandteil des Kapitels [**5 Datenmodell**].

- `/child::doc/child::chapter[position()=5]/child::section[position()=2]` wählt das zweite `section`-Element des fünften `chapter`-Elements des `doc`-Dokumentelements aus.

- `child::para[attribute::type="warning"]` wählt alle `para`-Kindelemente des Kontextknotens aus, die ein `type`-Attribut mit dem Wert `warning` besitzen.

- `child::para[attribute::type='warning'][position()=5]` wählt das fünfte `para`-Kindelement des Kontextknotens aus, das ein Attribut `type` mit dem Wert `warning` besitzt.

- `child::para[position()=5][attribute::type="warning"]` wählt das fünfte `para`-Kindelement des Kontextknotens aus, wenn dieses Kind ein `type`-Attribut mit dem Wert `warning` besitzt.

Anmerkung des Übersetzers:

Die letzten beiden Beispiele zeigen, wie sich durch ein Prädikat Kontextgröße und -position ändern können. Das Prädikat `[position()=5]` liefert immer eine Knotenmenge, die maximal einen Knoten enthält, je nachdem, ob es einen fünften Knoten gibt oder nicht. Das erste der beiden Beispiele beschränkt zunächst die betrachtete Knotenmenge auf die `para`-Elemente mit dem Attribut `type="warning"` und wählt anschließend aus diesen das fünfte aus. Das zweite Beispiel wählt zuerst das fünfte `para`-Element und anschließend aus der verbleibenden Knotenmenge die Knoten mit dem gewünschten Attribut aus. Besitzt das fünfte `para`-Element also gerade kein Attribut `type="warning"`, so ist das Ergebnis in diesem Fall die leere Knotenmenge.

- `child::chapter[child::title='Introduction']` wählt die `chapter`-Kindelemente des Kontextknotens aus, die wenigstens ein `title`-Kindelement mit einem Zeichenkettenwert gleich `Introduction` besitzen.

- `child::chapter[child::title]` wählt die `chapter`-Kindelemente des Kontextknotens aus, die ein oder mehrere `title`-Kindelemente besitzen.

- `child::*[self::chapter or self::appendix]` wählt die `chapter`- und `appendix`-Kindelemente des Kontextknotens aus.

> **Anmerkung des Übersetzers:**
>
> Hier werden aus allen Kindern (`child::*`) diejenigen ausgewählt, die selbst (`self::`) entweder ein `chapter`- oder ein `appendix`-Element sind. Hier muss man etwas aufpassen, da der Ausdruck in den eckigen Klammern für einen anderen Kontextknoten ausgewertet wird als der Ausdruck vor den Klammern. Der Vergleich mit den beiden vorhergehenden Beispielen verdeutlicht den Unterschied. Die genaue Definition folgt in Kapitel [2.4 Prädikate].

- `child::*[self::chapter or self::appendix][position()=last()]` wählt das letzte `chapter`- oder `appendix`-Kindelement des Kontextknotens aus.

> **Anmerkung des Übersetzers:**
>
> Hier wird zunächst die Menge aller `chapter`- oder `appendix`-Kinder betrachtet und aus dieser dann das letzte ausgewählt. Die Ergebnisknotenmenge enthält damit (maximal) einen Knoten.

Es gibt zwei Arten von Lokalisierungspfaden: relative und absolute Lokalisierungspfade.

Ein relativer Lokalisierungspfad wird durch eine Folge aus einem oder mehreren Lokalisierungsschritten gebildet, die durch `/` voneinander getrennt sind. Die Schritte eines relativen Lokalisierungspfades werden von links nach rechts zusammengesetzt. Jeder Schritt wählt der Reihe nach eine Knotenmenge relativ zu einem Kontextknoten aus. Eine Anfangsfolge von Schritten wird mit einem folgenden Schritt wie folgt zusammengesetzt: Die Anfangsfolge von Schritten wählt eine Knotenmenge relativ zu einem Kontextknoten aus. Jeder Knoten in dieser Menge wird dann als Kontextknoten für den folgenden Schritt benutzt. Die durch diesen Schritt bestimmten Knotenmengen werden vereinigt. Die Vereinigung ist dann genau die Knotenmenge, die durch das Zusammensetzen der Schritte ausgewählt wird. Zum Beispiel wählt `child::div/child::para` die `para`-Kindelemente der `div`-Kindelemente des Kontextknotens aus, oder – mit anderen Worten – die `para`-Enkelelemente, die `div`-Elternelemente haben.

Ein absoluter Lokalisierungspfad besteht aus dem Zeichen `/` und einem optional folgenden relativen Lokalisierungspfad. Ein `/` allein wählt den Wurzelknoten des Dokuments aus, das den Kontextknoten enthält. Falls ein relativer Lokalisierungspfad folgt, so wählt der Lokalisierungspfad die Knotenmenge aus, die ein relativer Lokalisierungspfad relativ zum Wurzelknoten des den Kontextknoten enthaltenen Dokuments auswählen würde.

> **Anmerkung des Übersetzers:**
>
> Hier bietet sich der Vergleich zu Pfaden im Dateisystem an: Sowohl in UNIX-Betriebssystemen als auch innerhalb von URLs wird der Schrägstrich »`/`« als Trennzeichen innerhalb von Dateipfaden benutzt. In ähnlicher Weise kann man sich Pfade in einem XML-Baum vorstellen. Ein relativer Pfad geht immer vom aktuellen Knoten (dem Kontextknoten) aus, ein absoluter Pfad beginnt immer an der Dokumentwurzel. Genau genommen ist ein absoluter Pfad allerdings ebenfalls relativ, nämlich zum Dokument des Kontextknotens. Das spielt eine Rolle, wenn Knoten aus mehreren Dokumenten verarbeitet werden.
>
> Man sollte sich allerdings einer Feinheit bewusst sein: Während beispielsweise durch »`verzeichnis1/verzeichnis2/datei3`« in einem Dateisystem exakt eine Datei adressiert wird, eben jene, die den Namen »`datei3`« trägt und sich in dem

Unterverzeichnis »`verzeichnis1/verzeichnis2`« befindet, wählt ein analoger XPath immer eine Knotenmenge aus, die in der Regel mehrere Knoten enthalten kann. Das sollte aber nicht weiter verwundern, darf ein Element doch durchaus mehrere gleichnamige Kindelemente besitzen.

Lokalisierungspfade

[1]	LocationPath	::=	RelativeLocationPath
			| AbsoluteLocationPath
[2]	AbsoluteLocationPath	::=	'/' RelativeLocationPath?
			| AbbreviatedAbsoluteLocationPath
[3]	RelativeLocationPath	::=	Step
			| RelativeLocationPath '/' Step
			| AbbreviatedRelativeLocationPath

Anmerkung des Übersetzers:

Die obige Grammatik gibt das bisher Beschriebene noch einmal formal wieder. Wie man sieht, können Lokalisierungspfade auch abgekürzte Bestandteile enthalten. Diese werden in Kapitel [**2.5 Abgekürzte Syntax**] beschrieben.

2.1 Lokalisierungsschritte

Ein Lokalisierungsschritt hat drei Bestandteile:

- eine Achse, welche die Beziehung zwischen den durch den Lokalisierungsschritt ausgewählten Knoten und dem Kontextknoten innerhalb des Baumes spezifiziert,
- einen Knotentest, der den Knotentyp und den erweiterten Namen der durch den Lokalisierungsschritt ausgewählten Knoten spezifiziert, sowie
- null oder mehr Prädikate, die mittels beliebiger Ausdrücke die durch den Lokalisierungsschritt ausgewählte Knotenmenge weiter verfeinern können.

Anmerkung des Übersetzers:

Prädikate im mathematischen Sinn sind Aussagen über Eigenschaften, die für die betreffenden Objekte wahr oder falsch sein können. In diesem Sinn wählt ein Prädikat aus einer Knotenmenge genau diejenigen Knoten aus, für die die entsprechende Aussage zutrifft. Aber auch ohne Kenntnis der Begriffswelt der Prädikatenlogik kann man sich anhand der Begriffe »Prädikatswein« oder »Prädikatsexamen« verdeutlichen, dass hier jeweils eine bestimmte Eigenschaft bzw. Qualität des beschriebenen Objekts ausgedrückt wird.

Ein Lokalisierungsschritt besteht syntaktisch aus dem Namen der Achse, gefolgt von zwei Doppelpunkten und dem Knotentest, gefolgt von null oder mehr in eckigen Klammern

eingeschlossenen Ausdrücken. Zum Beispiel enthält `child::para[position()=1]` die Achse `child`, den Knotentest `para` und ein Prädikat `[position()=1]`.

Die durch den Lokalisierungspfad ausgewählte Knotenmenge ergibt sich aus der durch Achse und Knotentest bestimmten Ausgangsknotenmenge, indem dort der Reihe nach die einzelnen Prädikate angewendet werden.

Die Ausgangsknotenmenge enthält alle Knoten, die zum Kontextknoten in der durch die Achse angegebenen Beziehung stehen und die den im Knotentest spezifizierten Knotentyp und erweiterten Namen besitzen. Zum Beispiel wählt der Lokalisierungsschritt `descendant::para` alle `para`-Nachkommen des Kontextknotens aus: `descendant` besagt, dass jeder Knoten in der Ausgangsknotenmenge ein Nachkomme des Kontextknotens sein muss; `para` besagt, dass jeder Knoten in der Ausgangsknotenmenge ein Element mit dem Namen `para` sein muss. Die verfügbaren Achsen werden in [**2.2 Achsen**] beschrieben, die verfügbaren Knotentests in [**2.3 Knotentests**]. Die Bedeutung einiger Knotentests hängt von der jeweiligen Achse ab.

Die Ausgangsknotenmenge wird durch das erste Prädikat gefiltert und ergibt eine neue Knotenmenge. Diese wird anschließend durch das zweite Prädikat gefiltert und so weiter. Die resultierende Knotenmenge ist schließlich die Knotenmenge, die durch den Lokalisierungsschritt ausgewählt wird. Die Achse beeinflusst, wie der Ausdruck in jedem Prädikat berechnet wird. Die Semantik eines Prädikats ist damit bezüglich einer Achse definiert (siehe [**2.4 Prädikate**]).

Anmerkung des Übersetzers:

Dieser letzte Satz bezieht sich auf Prädikate, die die Kontextposition und damit die Näheposition der gefilterten Knoten auswerten.

Zur Verdeutlichung der Vorgehensweise bei mehreren Prädikaten sei auf die beiden bereits diskutierten Beispiele

```
child::para[attribute::type="warning"][position()=5]
```

und

```
child::para[position()=5][attribute::type="warning"]
```

in Kapitel [**2 Lokalisierungspfade**] verwiesen.

Lokalisierungsschritte

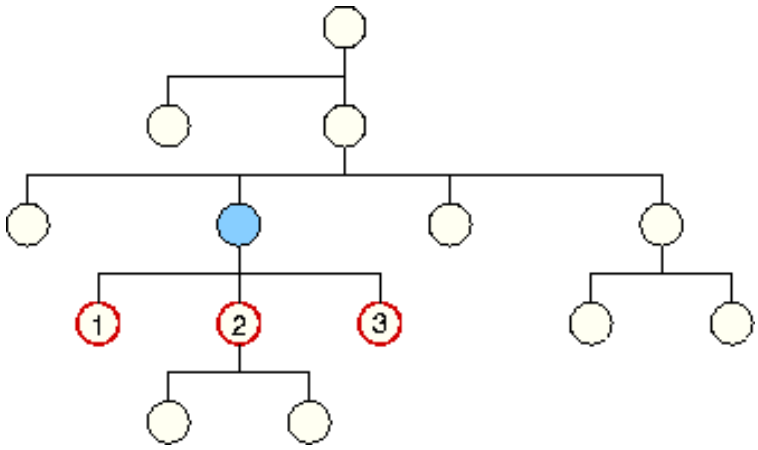
[4]	Step	::=	AxisSpecifier NodeTest Predicate* | AbbreviatedStep
[5]	AxisSpecifier	::=	AxisName '::' | AbbreviatedAxisSpecifier

2.2 Achsen

Es stehen die folgenden Achsen zur Verfügung:

- Die Achse `child` enthält die Kinder des Kontextknotens.

Anmerkung des Übersetzers:

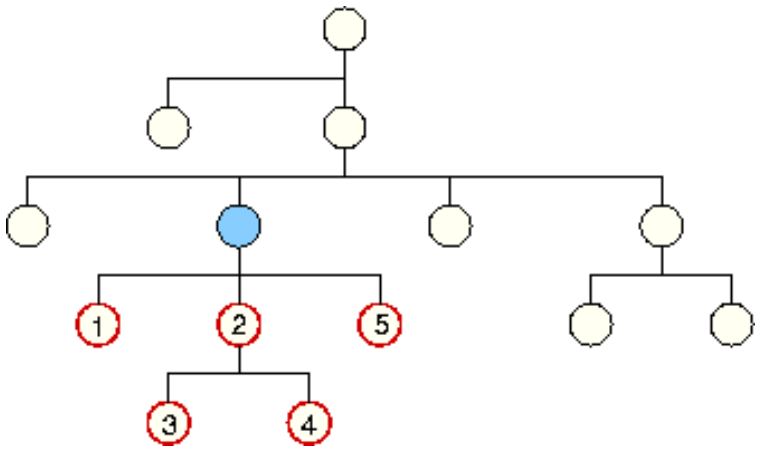


Die Kreise, die hier und bei den folgenden Achsen zur Darstellung der einzelnen Knoten genutzt werden, repräsentieren keine speziellen Knotentypen. Es könnte sich also sowohl um Elemente als auch um Textknoten, Kommentare oder Processing Instructions handeln. Sie repräsentieren allerdings niemals Attribut- oder Namensraumknoten, da diese nur über eigens dafür definierte Achsen erreicht werden können. Insbesondere sind es keine Kinder anderer Knoten.

Die Nummern geben die Näheposition der Knoten in der durch die Achse ausgewählten Knotenmenge an.

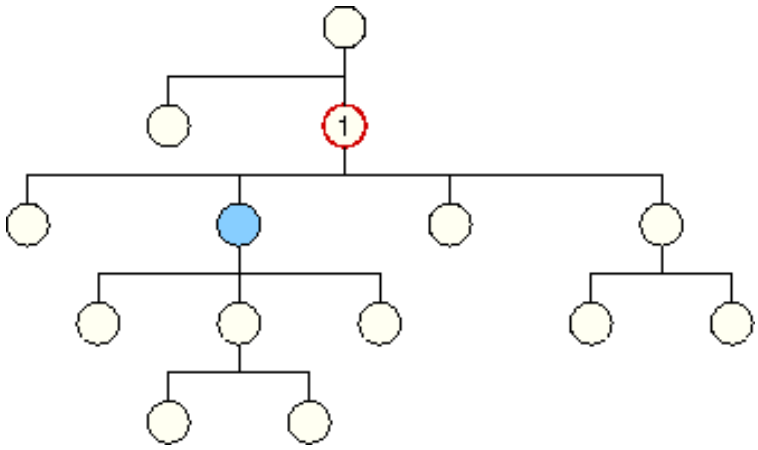
- Die Achse `descendant` enthält die Nachkommen des Kontextknotens; ein Nachkomme ist ein Kind oder ein Kind eines Kindes usw. Die Nachkommenachse enthält niemals Attribut- oder Namensraumknoten.

Anmerkung des Übersetzers:



- Die Achse `parent` enthält den Elternknoten des Kontextknotens, falls es einen gibt.

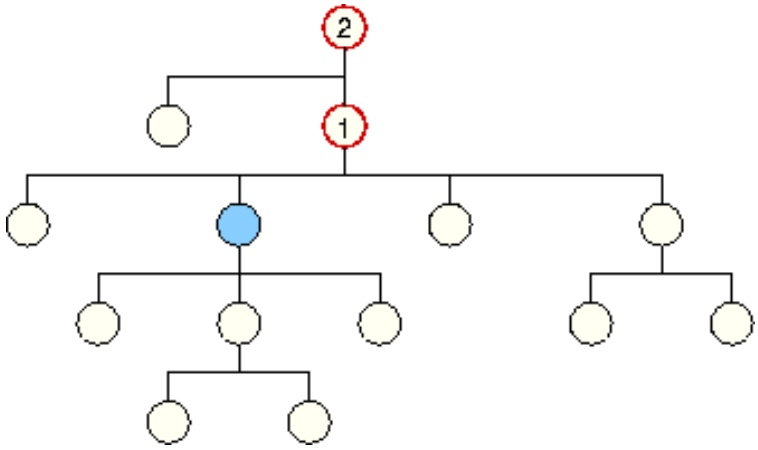
Anmerkung des Übersetzers:



Bis auf den Wurzelknoten besitzt jeder Knoten einen Elternknoten.

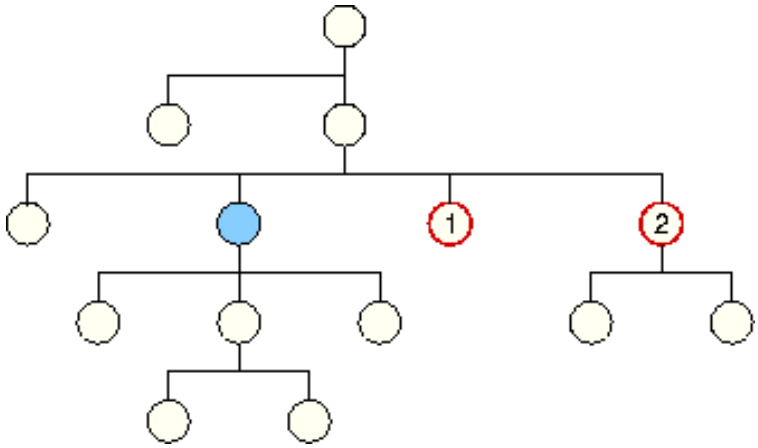
- Die Achse `ancestor` enthält die Vorfahren des Kontextknotens; die Vorfahren des Kontextknotens bestehen aus dem Elternknoten des Kontextknotens, dessen Elternknoten usw. Die Vorfahrenachse enthält somit immer den Wurzelknoten, es sei denn, der Kontextknoten selbst ist der Wurzelknoten.

Anmerkung des Übersetzers:



- Die Achse `following-sibling` enthält alle nachfolgenden Geschwister des Kontextknotens; falls der Kontextknoten ein Attribut- oder Namensraumknoten ist, ist diese Achse leer.

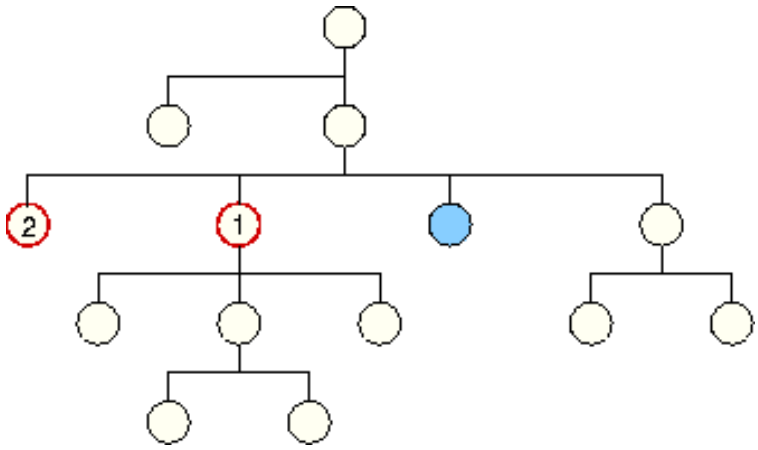
Anmerkung des Übersetzers:



Geschwister sind die Knoten, die den gleichen Elternknoten wie der Kontextknoten besitzen.

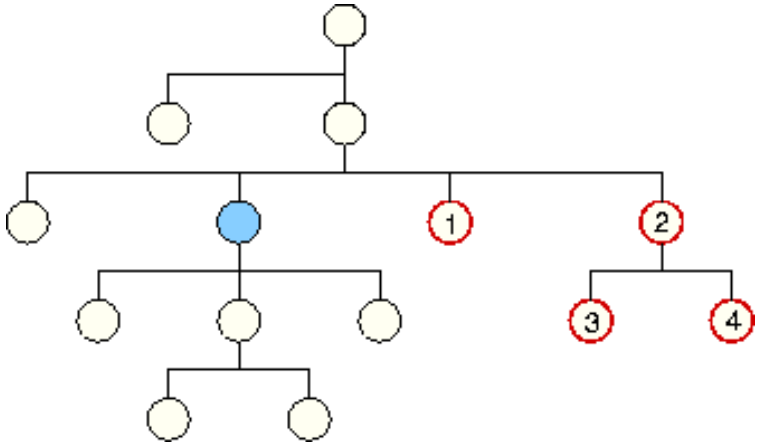
- Die Achse `preceding-sibling` enthält alle vorhergehenden Geschwister des Kontextknotens; falls der Kontextknoten ein Attribut- oder Namensraumknoten ist, ist diese Achse leer.

Anmerkung des Übersetzers:



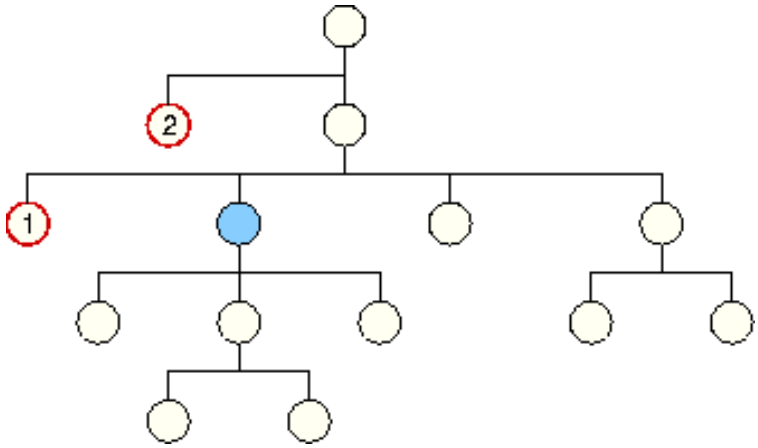
- Die Achse `following` enthält alle Knoten im gleichen Dokument wie der Kontextknoten, die nach dem Kontextknoten in Dokumentordnung auftreten, und zwar ohne seine Nachkommen und ohne Attribut- und Namensraumknoten.

Anmerkung des Übersetzers:



- Die Achse `preceding` enthält alle Knoten im gleichen Dokument wie der Kontextknoten, die vor dem Kontextknoten in Dokumentordnung auftreten, und zwar ohne seine Vorfahren und ohne Attribut- und Namensraumknoten.

Anmerkung des Übersetzers:

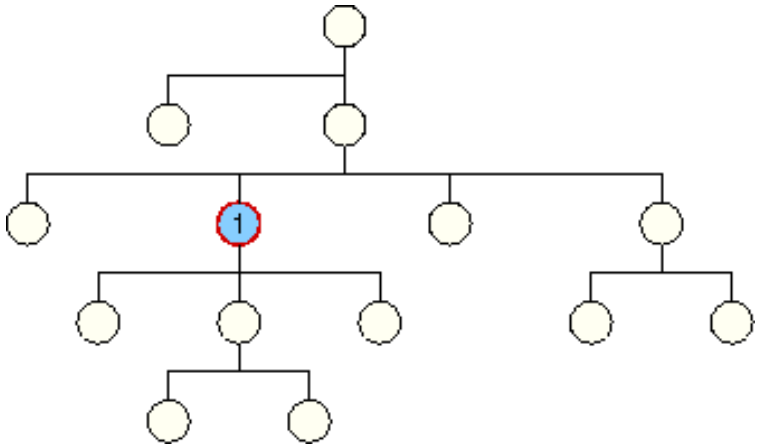


- Die Achse `attribute` enthält die Attribute des Kontextknotens; diese Achse ist leer, es sei denn, der Kontextknoten ist ein Elementknoten.

- Die Achse `namespace` enthält alle Namensraumknoten des Kontextknotens; diese Achse ist leer, es sei denn, der Kontextknoten ist ein Elementknoten.

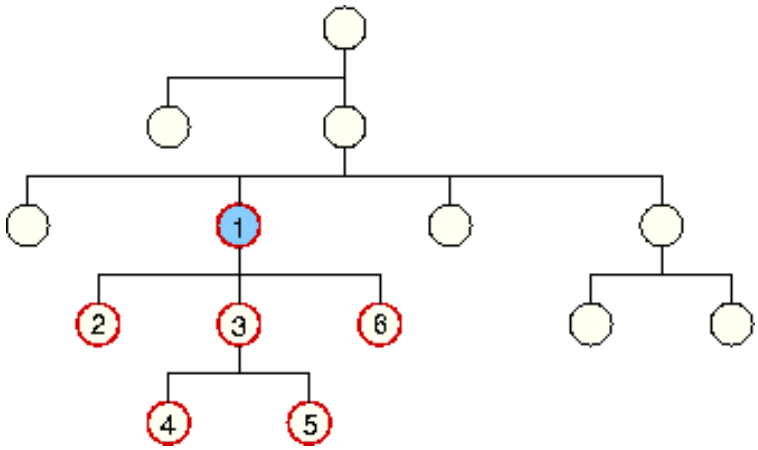
- Die Achse `self` enthält nur den Kontextknoten selbst.

Anmerkung des Übersetzers:



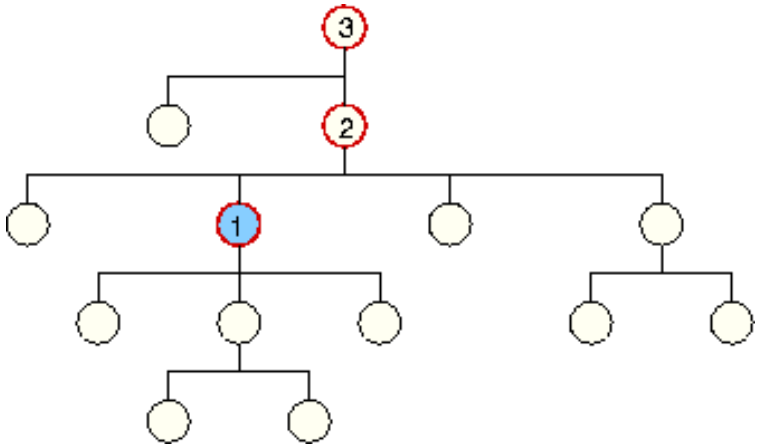
- Die Achse `descendant-or-self` enthält den Kontextknoten sowie die Nachkommen des Kontextknotens.

Anmerkung des Übersetzers:



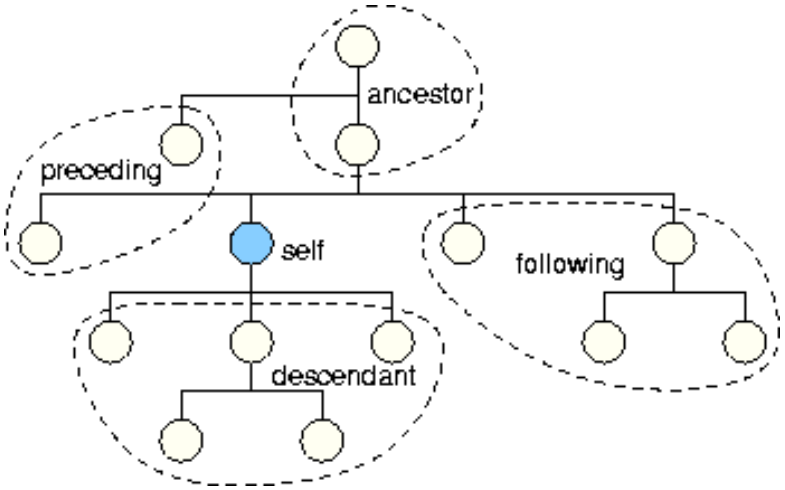
- Die Achse `ancestor-or-self` enthält den Kontextknoten sowie die Vorfahren des Kontextknotens; diese Achse enthält somit immer den Wurzelknoten.

Anmerkung des Übersetzers:



Anmerkung: Die Achsen `ancestor`, `descendant`, `following`, `preceding` und `self` partitionieren ein Dokument (unter Auslassung der Attribut- und Namensraumknoten): sie überschneiden sich nicht und enthalten zusammen alle Knoten des Dokuments.

Anmerkung des Übersetzers:



Achsen

[6] AxisName ::= 'ancestor'
| 'ancestor-or-self'
| 'attribute'
| 'child'
| 'descendant'
| 'descendant-or-self'
| 'following'
| 'following-sibling'
| 'namespace'
| 'parent'
| 'preceding'
| 'preceding-sibling'
| 'self'

Anmerkung des Übersetzers:

Attribut- und Namensraumachse nehmen hier eine Sonderrolle ein. Die adressierten Knoten zeichnen sich nämlich durch einen bestimmten festen Knotentyp aus (den so genannten Hauptknotentyp). Da diese Knoten per Definition keine Kinder anderer Knoten sind, wurden die speziellen Achsen `attribute` und `namespace` definiert. Trotzdem kann sich auch an diese Achsen ein beliebiger Knotentest anschließen (siehe [**2.3 Knotentests**]). Dies birgt die Gefahr, dass zwar syntaktisch korrekte, aber sinnlose Lokalisierungsschritte benutzt werden können, die niemals einen Knoten auswählen. Der Lokalisierungsschritt `attribute::text()`, der über die Attributachse einen Textknoten auswählen soll, liefert beispielsweise immer eine leere Knotenmenge.

2.3 Knotentests

Jede Achse besitzt einen **Hauptknotentyp**. Falls eine Achse Elemente enthalten kann, so ist der Hauptknotentyp der Elementtyp, ansonsten ist es genau der Typ der Knoten, die die Achse enthalten kann. Das bedeutet:

- Für die Attributachse ist der Hauptknotentyp der Attributtyp.

- Für die Namensraumachse ist der Hauptknotentyp der Namensraum-Typ.
- Für alle anderen Achsen ist der Hauptknotentyp der Elementtyp.

Ein Knotentest, der ein QName ist, ist genau dann erfüllt, wenn der Knotentyp (siehe [**5 Datenmodell**]) der Hauptknotentyp ist und einen erweiterten Namen besitzt, der gleich dem erweiterten Namen des QName ist. Beispielsweise wählt `child::para` die `para`-Kindelemente des Kontextknotens aus. Falls der Kontextknoten keine `para`-Kinder besitzt, ist das Ergebnis eine leere Knotenmenge. `attribute::href` wählt die `href`-Attribute des Kontextknotens aus. Falls der Kontextknoten keine `href`-Attribute besitzt, ist das Ergebnis eine leere Knotenmenge.

Anmerkung des Übersetzers:

Das Akronym QName steht für "qualifizierter Name" und bedeutet, dass der entsprechende Name aus einem optionalen Präfix und einem lokalen Bestandteil bestehen kann. Beide Teile werden durch einen Doppelpunkt voneinander getrennt. Durch das Präfix wird der Namensraum bestimmt, zu dem der lokale Name gehört. Im Beispiel `xlink:href` ist `xlink` das Präfix und `href` der lokale Bestandteil. Die obigen Beispiele verwenden Knotentests, die letztendlich nur lokale Namen sind, also keine Namensräume berücksichtigen.

Möchte man auf das `href`-Attribut aus dem `xlink`-Namensraum zugreifen, wäre also zu schreiben: `attribute::xlink:href`. Entsprechend greift man per `preceding-sibling::xhtml:h1` auf die dem Kontextknoten vorhergehenden `h1`-Elemente aus dem `xhtml`-Namensraum zu. Schließlich ein Beispiel für die Namensraumachse: `namespace::xlink` wählt genau den Namensraumknoten aus, der zum Präfix `xlink` gehört. Der qualifizierte Name eines Namensraumknotens enthält übrigens niemals ein Präfix, siehe [**5.4 Namensraumknoten**].

Wie in den Beispielen am Beginn des Kapitels bereits gezeigt wurde, eignet sich die Achse `self`, um innerhalb eines Prädikats bestimmte Elementtypen aus einer Knotenmenge herauszufiltern, wie z.B. in `child::*[self::chapter or self::appendix]`. Leider funktioniert eine analoge Vorgehensweise für Attribut- oder Namensraumknoten nicht. Das erweist sich insbesondere bei negativen Bedingungen als hinderlich. Möchte man z.B. alle Attribute bis auf das `href`-Attribut auswählen, leistet `attribute::*[not(self::href)]` leider nicht das Gewünschte. Dieser Ausdruck liefert nämlich nach wie vor alle Attribute. Da die `self`-Achse Elemente enthalten kann, ihr Hauptknotentyp somit der Elementtyp ist, sucht der Ausdruck `self::href` immer nach einem Element mit dem Namen `href`, niemals nach einem Attribut. Eine Lösung für dieses Problem besteht in der Auswertung des Attributnamens, siehe die Funktionen **name** und **local-name**.

Ein QName im Knotentest wird in einen erweiterten Namen unter Verwendung der Namensraumdeklarationen aus dem Kontext des Ausdrucks expandiert. Dies geschieht in der gleichen Weise wie bei Elementnamen in Start- und End-Tags, allerdings mit der Ausnahme, dass ein mit `xmlns` deklarierter voreingestellter Namensraum nicht genutzt wird: d.h. enthält QName kein Präfix, so ist der Namensraum-URI leer (das ist die gleiche Regel, nach der auch Attributnamen expandiert werden). Es ist ein Fehler, wenn der QName ein Präfix enthält, für das es keine Namensraumdeklaration im Kontext des Ausdrucks gibt.

Anmerkung des Übersetzers:

Qualifizierte Namen werden also gemäß der Namensraum-Empfehlung [XML Names] expandiert. Das bedeutet insbesondere, dass in einem XPath-Ausdruck nicht das gleiche Präfix wie im betrachteten XML-Dokument benutzt werden muss. Wichtig ist nur, dass beide Präfixe den gleichen Namensraum repräsentieren. Wenn beispielsweise das zugrunde liegende XML-Dokument ein XHTML-Dokument ist, dessen Elemente zum Namensraum `http://www.w3.org/1999/xhtml` gehören, so kann beispielsweise das Dokumentelement über den XPath-Ausdruck `/child::xhtml:html` ausgewählt werden – vorausgesetzt, das Präfix `xhtml` wurde im Kontext des Ausdrucks an den gleichen Namensraum gebunden. Dabei ist es völlig unerheblich, welches Präfix im XHTML-Dokument benutzt wurde – es ist sogar möglich, dass dort nur eine Deklaration für den voreingestellten Namensraum vorkommt: `<html xmlns="http://www.w3.org/1999/xhtml">`.

Gerade in diesen Fällen ist Vorsicht geboten. Sehr leicht übersieht man eine solche Namensraumdeklaration, die sich selbstverständlich auch auf alle Kindelemente auswirkt, und ist gewillt, nur den jeweiligen Elementnamen in einem Ausdruck anzugeben, z.B. bei folgender XSLT-Anweisung: `<xsl:value-of select="title" />`. Richtig wäre stattdessen (der Vollständigkeit halber mit Namensraumdeklaration): `<xsl:value-of select="xhtml:title" xmlns:xhtml="http://www.w3.org/1999/xhtml">`. Entsprechend müssen in Lokalisierungpfaden alle Schritte vollständig qualifiziert sein: `xhtml:div/xhtml:p`.

Stellt man fest, dass XPath-Ausdrücke nach dem Einfügen einer DTD in das Originaldokument nicht mehr die richtigen Knoten zurückliefern (was sich beispielsweise darin äußert, dass ein XSLT-Stylesheet nicht mehr die erwartete Ausgabe liefert), ist in der Regel ebenfalls die Deklaration eines voreingestellten Namensraums die Ursache. Eine Deklaration innerhalb der DTD der Form `<!ATTLIST html xmlns CDATA #FIXED "http://www.w3.org/1999/xhtml">` versetzt alle Kindelemente von `html` ohne Präfix und auch `html` selbst in den XHTML-Namensraum. Ohne die DTD gehören sie keinem Namensraum an. In solchen Fällen sollte man die Deklaration des voreingestellten Namensraums in das XML-Dokument aufnehmen, damit XPath-Ausdrücke unabhängig von der Auswertung einer DTD immer das gleiche Ergebnis liefern.

Schließlich sei noch einmal darauf hingewiesen, dass sich ein voreingestellter Namensraum im Kontext eines XPath-Ausdrucks nicht auf den Ausdruck auswirkt. Ein qualifizierter Name ohne Präfix adressiert damit immer ein Element oder ein Attribut, das zu keinem Namensraum gehört. Die folgende Variante ist damit *keine* Alternative zum obigen Beispiel: `<xsl:value-of select="title" xmlns="http://www.w3.org/1999/xhtml">`.

Ein Knotentest `*` ist für jeden Knoten des Hauptknotentyps erfüllt. Beispielsweise wählt `child::*` alle Kindelemente und `attribute::*` alle Attributknoten des Kontextknotens aus.

Anmerkung des Übersetzers:

Handelt es sich beim Kontextknoten um einen Elementknoten, so liefert der Schritt `parent::*` nur dann die leere Knotenmenge, wenn dieser das Dokumentelement repräsentiert. Der Elementknoten für das Dokumentelement besitzt zwar ebenfalls einen Elternknoten, nämlich die Wurzel `/`, diese ist aber nicht vom Hauptknotentyp der Achse `parent`, dem Elementtyp.

Ein Knotentest kann in der Form NCName:* auftreten. In diesem Fall wird das Präfix wie bei einem QName unter Verwendung der Namensraumdeklarationen des Kontextes expandiert. Es ist ein Fehler, wenn es für das Präfix keine Namensraumdeklaration im Kontext des Ausdrucks gibt. Der Knotentest ist erfüllt für jeden Knoten des Hauptknotentyps, dessen erweiterter Name den Namensraum-URI besitzt, zu dem das Präfix expandiert, unabhängig vom lokalen Bestandteil des Namens.

Anmerkung des Übersetzers:

Beispielsweise wählt `descendant::xhtml:*` alle Nachkommen aus dem `xhtml`-Namensraum und `attribute::xlink:*` alle Attribute aus dem `xlink`-Namensraum aus.

Diese Form von Knotentests kann in einem XSLT-Stylesheet innerhalb eines Musters z.B. dazu genutzt werden, alle Elemente eines bestimmten Namensraums in der gleichen Weise zu behandeln. Eingebettete XHTML-Elemente in einem beliebigen zu transformierenden XML-Dokument können auf diese Weise einfach in die Ausgabe kopiert werden, ohne dass diese Elemente explizit benannt werden müssen.

Der Knotentest `text()` ist erfüllt für jeden Textknoten. Zum Beispiel wählt `child::text()` alle Textknoten aus, die Kinder des Kontextknotens sind. Analog ist der Knotentest `comment()` für jeden Kommentarknoten erfüllt und der Knotentest `processing-instruction()` für jede Processing Instruction. Dem Test `processing-instruction()` kann ein Literal als Argument übergeben werden. In diesem Fall ist der Test für jede Processing Instruction erfüllt, deren Name gleich dem Wert des übergebenen Literals ist.

Anmerkung des Übersetzers:

Offensichtlich werden Kommentare nicht einfach ignoriert, sondern durch eigene Knoten innerhalb des XML-Baumes repräsentiert. Auf diese Weise können XPath-Ausdrücke mittels `comment()` auch auf Kommentarknoten zugreifen. Ein Stylesheet kann damit XML-Kommentare verarbeiten und diese geeignet darstellen.

Für Processing Instructions gibt es ebenfalls entsprechende Knoten im XML-Baum. Jede Processing Instruction besitzt einen Namen (ein Ziel), z.B. `xml-stylesheet` in `<?xml-stylesheet href=... ?>`. Wird dieser Name im Knotentest `processing-instruction()` angegeben, ist der entsprechende Test nur für Processing Instructions mit gleichem Namen erfüllt. Namen von Processing Instructions werden dabei nicht von Namensraumdeklarationen beeinflusst. Das Argument für `processing-instruction()` ist damit kein qualifizierter Name, sondern ein Zeichenkettenliteral. Der Knotentest für die zitierte Processing Instruction lautet z.B. `processing-instruction('xml-stylesheet')`.

Der Knotentest `node()` ist für alle Knoten jedes beliebigen Typs erfüllt.

Anmerkung des Übersetzers:

Möchte man alle Knoten eines Dokuments auswählen, kann man z.B. die folgenden Knotenmengen vereinigen: `/descendant-or-self::node()`, `/descendant-or-self::node()/attribute::*` und `/descendant-or-self::node()/namespace::*`.

Es wurde bereits auf die Sonderrolle von Attribut- und Namensraumknoten hingewiesen. Diese werden nicht durch einen Knotentest, sondern durch entsprechende Achsen ausgewählt. Das hat zur Folge, dass sich zwar einfach prüfen lässt, ob der Kontextknoten z. B. ein Textknoten ist (per `self::text()`; analog für Element-, Kommentar- und Processing-Instruction-Knoten), allerdings funktioniert diese Methode nicht bei Attribut- und Namensraumknoten. Hier muss man stattdessen überprüfen, ob sich der Kontextknoten in der Menge der Attribut- bzw. Namensraumknoten des Elternknotens befindet, siehe [**3.3 Knotenmengen**].

[7] NodeTest ::= NameTest
| NodeType '(' ')'
| 'processing-instruction' '(' Literal ')'

2.4 Prädikate

Eine Achse ist entweder vorwärts- oder rückwärtsgerichtet. Eine vorwärtsgerichtete Achse enthält immer nur den Kontextknoten oder Knoten, die nach dem Kontextknoten im Dokument auftreten. Eine rückwärtsgerichtete Achse enthält immer nur den Kontextknoten oder Knoten, die vor dem Kontextknoten im Dokument auftreten. Demzufolge sind die Achsen `ancestor`, `ancestor-or-self`, `preceding` und `preceding-sibling` rückwärtsgerichtete Achsen. Alle anderen Achsen sind vorwärtsgerichtet. Da die Achse `self` immer höchstens einen Knoten enthält, hat es keine Bedeutung, ob sie als vorwärts- oder rückwärtsgerichtete Achse betrachtet wird. Die **Näheposition** eines Knotens in einer Knotenmenge bezüglich einer Achse ist definiert als die Position des Knotens in dieser Knotenmenge, welche in Dokumentordnung geordnet ist, wenn es sich um eine vorwärtsgerichtete Achse handelt, und welche in umgekehrter Dokumentordnung geordnet ist, wenn es sich um eine rückwärtsgerichtete Achse handelt. Die erste Position ist 1.

Anmerkung des Übersetzers:

Diese recht komplizierte Definition bedarf einer Erläuterung. Die Knoten in einer Knotenmenge sind ungeordnet – es handelt sich schließlich um eine Menge. Allerdings kann über die Funktion **position** die Position eines Knotens in der aktuellen Knotenliste bestimmt werden. Die Reihenfolge der Knoten orientiert sich an der Reihenfolge, in der die Knoten im XML-Dokument auftreten. Bei vorwärtsgerichteten Achsen wird diese Ursprungsreihenfolge beibehalten und man spricht von Dokumentordnung. Für rückwärtsgerichtete Achsen wird die Reihenfolge umgekehrt und man spricht von umgekehrter Dokumentordnung.

Die obige Definition hat zur Folge, dass innerhalb eines Lokalisierungsschrittes immer der nächstgelegene Knoten die Position 1 hat. Abhängig von der Blickrichtung kann es sich um einen unmittelbaren Vorgänger (`preceding`) bzw. Vorfahren (`ancestor`) oder aber um einen unmittelbaren Nachfolger (`following`) bzw. Nachkommen (`descendant`) handeln. Der Begriff Näheposition beschreibt damit die Nähe zum Kontextknoten. Die Abbildungen

zu den verschiedenen Achsen in Kapitel [**2.2 Achsen**] verdeutlichen die Näheposition der jeweiligen zur Achse gehörenden Knoten. Genau genommen ist die `parent`-Achse ebenfalls rückwärtsgerichtet, allerdings enthält diese wie `self` maximal einen Knoten.

Die Näheposition ist nur innerhalb von Prädikaten in Lokalisierungsschritten von Bedeutung, da hier die ausschlaggebende Achse bekannt ist. Die in [**4.2 Zeichenkettenfunktionen**] vorgestellte Funktion **string** (ebenso wie die Funktionen **number** und **boolean**) konvertiert dagegen bei einer Knotenmenge immer den ersten Knoten bezüglich der Dokumentordnung, unabhängig davon, auf welchem Weg diese Knotenmenge gebildet wurde.

Für vorwärtsgerichtete Achsen gilt daher folgende Gleichheit (am Beispiel `following`):

```
string(following::p) = string(following::p[position()=1])
```

Dies ist bei rückwärtsgerichteten Achsen nicht der Fall. Stattdessen gilt hier (am Beispiel `preceding`):

```
string(preceding::p) = string(preceding::p[position()=last()])
```

Die XSLT-Anweisung `xsl:value-of` verwendet implizit die Funktion **string**, um einen Textknoten zu generieren. Will man also auf den Wert des ersten Knotens einer rückwärtsgerichteten Achse zugreifen, muss man diesen immer explizit durch ein Prädikat auswählen. Die Ausgabe des Wertes eines Attributs `id` des Vorgängerknotens erfolgt daher z.B. durch `<xsl:value-of select="preceding::p[position()=1]/@id" />`.

Für Attribute und Namensraumknoten ist die Diskussion um Position und Richtung bedeutungslos. Das liegt daran, dass die Reihenfolge, in der Attribute und Namensraumdeklarationen im Start-Tag eines Elements angegeben wurden, als irrelevant angesehen wird. Informationen darüber sind daher nicht mehr im XML-Baum enthalten.

Ein Prädikat filtert eine Knotenmenge bezüglich einer Achse und produziert damit eine neue Knotenmenge. Für jeden zu filternden Knoten der Knotenmenge wird der dazugehörige Ausdruck PredicateExpr berechnet, und zwar mit diesem Knoten als Kontextknoten, der Anzahl der Knoten der Knotenmenge als Kontextgröße und mit der Näheposition des Knotens in der Knotenmenge bezüglich der Achse als Kontextposition. Falls die Berechnung von PredicateExpr für diesen Knoten *wahr* ergibt, wird der Knoten in die Ergebnisknotenmenge aufgenommen, andernfalls nicht.

Anmerkung des Übersetzers:

Diese Definition soll anhand eines Beispiels veranschaulicht werden:

```
ancestor::person[position() >= 2]
```

Achse und Knotentest `ancestor::person` liefern die Menge aller `person`-Elemente, die Vorfahren des Kontextknotens sind. Für jeden dieser Elementknoten wird nun der Ausdruck `position() >= 2` berechnet. Die Kontextgröße ist dabei die Anzahl aller `person`-Vorfahren. Da `ancestor` eine rückwärtsgerichtete Achse ist, werden die Knoten entgegen der Originalreihenfolge nummeriert. Damit werden alle `person`-Elementknoten bis auf den ersten (d.h. den nächsten) in die Ergebnisknotenmenge aufgenommen. Da als Knotentest des Lokalisierungsschrittes `person` verwendet wurde (und nicht `*`), muss es sich bei

diesem ersten `person`-Elementknoten nicht um den Elternknoten handeln. Würde sich ein weiteres Prädikat an den obigen Ausdruck anschließen, wäre die gerade berechnete Ergebnisknotenmenge Ausgangspunkt für dieses Prädikat.

Falls der Ausdruck eines nachfolgenden Prädikats nicht auf Kontextgröße oder -position zugreift, kann dieser Ausdruck bereits im ersten Prädikat berechnet und über den logischen Operator `and` mit dem dortigen Ausdruck verbunden werden. Der Lokalisierungsschritt

```
child::chapter[child::title][attribute::type="warning"]
```

liefert damit die gleiche Knotenmenge wie

```
child::chapter[child::title and attribute::type="warning"]
```

Ein PredicateExpr wird durch Berechnung des Expr und anschließender Konvertierung des Ergebnisses in einen booleschen Wert bestimmt. Falls das Ergebnis eine Zahl war, wird es für den Fall, dass diese Zahl gleich der Kontextposition ist, in den Wert *wahr* konvertiert, ansonsten in den Wert *falsch*. Wenn das Ergebnis keine Zahl war, dann wird es so konvertiert wie bei einem Aufruf der Funktion **boolean**. Damit ist ein Lokalisierungspfad `para[3]` äquivalent zu `para[position()=3]`.

Anmerkung des Übersetzers:

Da die abgekürzte Syntax erst im folgenden Kapitel beschrieben wird, sollte als Beispiel hier besser `child::para[3]` verwendet werden.

Für Zahlen als Wert eines PredicateExpr gilt hier eine Sonderregel. Diese ermöglicht eine Schreibweise, die dem Zugriff auf Feldelemente in anderen Programmiersprachen gleicht. Allerdings muss beachtet werden, dass ein Ausdruck `child::para[$num]` nur dann gleichbedeutend mit `child::para[position()=$num]` ist, wenn die Variable `num` tatsächlich eine Zahl, also ein Objekt vom Typ *number* enthält. Wenn sie stattdessen nur eine geeignete Zeichenkette enthält, wird der Variableninhalt gemäß der Funktion **boolean** in einen booleschen Wert konvertiert. Das Gleiche gilt für jedes andere Objekt, das sich in eine Zahl konvertieren ließe.

Prädikate

[8]	Predicate	::=	'[' PredicateExpr ']'
[9]	PredicateExpr	::=	Expr

2.5 Abgekürzte Syntax

Zunächst einige Beispiele für Lokalisierungspfade, die die abgekürzte Syntax benutzen:

- `para` wählt die `para`-Kindelemente des Kontextknotens aus.
- * wählt alle Kindelemente des Kontextknotens aus.

- `text()` wählt alle Textknoten aus, die Kinder des Kontextknotens sind.
- `@name` wählt das Attribut `name` des Kontextknotens aus.
- `@*` wählt alle Attribute des Kontextknotens aus.
- `para[1]` wählt das erste `para`-Kindelement des Kontextknotens aus.
- `para[last()]` wählt das letzte `para`-Kindelement des Kontextknotens aus.
- `*/para` wählt alle `para`-Enkelelemente des Kontextknotens aus.
- `/doc/chapter[5]/section[2]` wählt das zweite `section`-Element des fünften `chapter`-Elements von `doc` aus.
- `chapter//para` wählt die `para`-Elemente aus, die Nachkommen der `chapter`-Kindelemente des Kontextknotens sind.
- `//para` wählt alle `para`-Nachkommen der Dokumentwurzel aus und somit alle `para`-Elemente im gleichen Dokument wie der Kontextknoten.
- `//olist/item` wählt all die `item`-Elemente aus dem gleichen Dokument wie der Kontextknoten aus, die ein `olist`-Elternelement besitzen.
- `.` wählt den Kontextknoten aus.
- `.//para` wählt die `para`-Elemente aus, die Nachkommen des Kontextknotens sind.
- `..` wählt den Elternknoten des Kontextknotens aus.
- `../@lang` wählt das Attribut `lang` des Elternknotens des Kontextknotens aus.
- `para[@type="warning"]` wählt alle `para`-Kindelemente des Kontextknotens aus, die ein Attribut `type` mit dem Wert `warning` besitzen.
- `para[@type="warning"][5]` wählt das fünfte `para`-Kindelement des Kontextknotens aus, das ein Attribut `type` mit dem Wert `warning` besitzt.
- `para[5][@type="warning"]` wählt das fünfte `para`-Kindelement des Kontextknotens aus, falls dieses Kind ein Attribut `type` mit dem Wert `warning` besitzt.
- `chapter[title="Introduction"]` wählt die `chapter`-Kindelemente des Kontextknotens aus, die ein oder mehrere `title`-Kindelemente mit einem Zeichenkettenwert gleich `Introduction` besitzen.
- `chapter[title]` wählt die `chapter`-Kindelemente des Kontextknotens aus, die ein oder mehrere `title`-Kindelemente besitzen.
- `employee[@secretary and @assistant]` wählt alle `employee`-Kindelemente des Kontextknotens aus, die sowohl ein Attribut `secretary` als auch ein Attribut `assistant` besitzen.

Die wichtigste Abkürzung besteht darin, dass `child::` in einem Lokalisierungsschritt weggelassen werden kann. Die Standardachse ist also `child`. So steht beispielsweise ein Lokalisierungspfad `div/para` abkürzend für `child::div/child::para`.

Für Attribute gibt es ebenfalls eine Abkürzung: `attribute::` kann zu @ abgekürzt werden. Ein Lokalisierungspfad `para[@type="warning"]` steht beispielsweise abkürzend für `child::para[attribute::type="warning"]` und wählt damit `para`-Kindelemente mit einem Attribut `type` aus, dessen Wert gleich `warning` ist.

`//` ist die Abkürzung für `/descendant-or-self::node()/`. Zum Beispiel steht `//para` abkürzend für `/descendant-or-self::node()/child::para` und wählt damit alle `para`-Elemente im Dokument aus (selbst ein `para`-Element, das ein Dokumentelement ist, wird durch `//para` ausgewählt, da der Dokumentelementknoten ein Kind des Wurzelknotens ist). `div//para` steht abkürzend für `div/descendant-or-self::node()/child::para` und wählt daher alle `para`-Nachfolger von `div`-Kindern aus.

Anmerkung des Übersetzers:

Hier enthält das Originaldokument einen kleinen Fehler. Der vollständige Lokalisierungspfad für das letzte Beispiel muss `child::div/descendant-or-self::node()/child::para` lauten.

Anmerkung: Der Lokalisierungspfad `//para[1]` bedeutet *nicht* das Gleiche wie `/descendant::para[1]`. Der zweite wählt das erste Nachkommenelement `para` aus, der erste wählt alle `para`-Nachkommen aus, die das erste Kind ihrer Eltern sind.

Anmerkung des Übersetzers:

Davon kann man sich durch Bestimmung des vollständigen Ausdrucks leicht überzeugen:

```
//para[1] = /descendant-or-self::node()/para[1]
          = /descendant-or-self::node()/child::para[1]
```

Das Prädikat `[1]` wirkt damit auf die durch die `child`-Achse im zweiten Lokalisierungsschritt bestimmte Knotenmenge, wogegen es in `/descendant::para[1]` zur `descendant`-Achse gehört.

Ein Lokalisierungsschritt `.` steht abkürzend für `self::node()`. Das ist insbesondere in Verbindung mit `//` nützlich. Der Lokalisierungspfad `.//para` steht zum Beispiel abkürzend für

```
self::node()/descendant-or-self::node()/child::para
```

und wählt daher alle `para`-Elemente aus, die Nachkommen des Kontextknotens sind.

Analog steht der Lokalisierungsschritt `..` abkürzend für `parent::node()`. Zum Beispiel steht `../title` abkürzend für `parent::node()/child::title` und wählt damit die `title`-Kindelemente des Elternknotens des Kontextknotens aus.

Anmerkung des Übersetzers:

Die Kombination aus `.` und `/` ist unnötig und kann weggelassen werden. Ausdrücke der Form `./title` oder `./@name` können kürzer als `title` bzw. `@name` geschrieben werden.

Ein Blick in die unten stehende Grammatik zeigt, dass es sich bei `.` und `..` um vollständige Lokalisierungsschritte handelt. Ihnen können also keine Prädikate folgen. Ausdrücke der Form `.[self::par]` oder `..[@name='foo']` sind nicht zulässig.

Abkürzungen

[10]	AbbreviatedAbsoluteLocationPath	::=	'//' RelativeLocationPath
[11]	AbbreviatedRelativeLocationPath	::=	RelativeLocationPath '//' Step
[12]	AbbreviatedStep	::=	'.'
			| '..'
[13]	AbbreviatedAxisSpecifier	::=	'@'?

3 Ausdrücke

3.1 Grundlagen

Eine Variablenreferenz (VariableReference) ergibt den Wert, der an den Variablennamen innerhalb der Menge der Variablenbelegungen des Kontexts gebunden ist. Es ist ein Fehler, falls dem Variablennamen in der Menge der Variablenbelegungen aus dem Kontext des Ausdrucks kein Wert zugewiesen wurde.

Anmerkung des Übersetzers:

Der Name einer Variablen ist ein qualifizierter Name, kann also ein Präfix enthalten, das auf einen Namensraum verweist. Mittels des Zeichens $ wird auf den Wert einer Variablen zugegriffen.

Der letzte Satz des obigen Abschnitts bedeutet, dass Variablen vor ihrer Benutzung definiert worden sein müssen. Wie bereits erwähnt, sieht XPath dafür keinerlei Sprachelemente vor. In [XSLT] verwendete Variablen haben immer einen definierten Wert. So wird durch das leere Element `<xsl:variable name="foo:var" />` eine Variable namens `foo:var` definiert und mit der leeren Zeichenkette belegt.

Zum Gruppieren können runde Klammern verwendet werden.

[14]	Expr	::=	OrExpr
[15]	PrimaryExpr	::=	VariableReference
			| '(' Expr ')'
			| Literal
			| Number

| FunctionCall

3.2 Funktionsaufrufe

Ein Ausdruck, der ein Funktionsaufruf (FunctionCall) ist, wird ausgewertet, indem anhand des Funktionsnamens (FunctionName) die Funktion in der Funktionsbibliothek des Ausdruckskontexts bestimmt, jedes der Argumente (Argument) berechnet und in den von der Funktion erwarteten Typ konvertiert und schließlich die Funktion mit den konvertierten Argumenten aufgerufen wird. Es ist ein Fehler, wenn eine falsche Anzahl von Argumenten übergeben wird oder eines der Argumente nicht in den geforderten Typ konvertiert werden kann. Das Ergebnis des Funktionsaufrufes (FunctionCall) ist der von der Funktion zurückgelieferte Wert.

Die Konvertierung eines Arguments in den Typ *string* geschieht so wie beim Aufruf der Funktion **string**. Die Konvertierung eines Arguments in den Typ *number* geschieht so wie beim Aufruf der Funktion **number**. Die Konvertierung eines Arguments in den Typ *boolean* geschieht so wie beim Aufruf der Funktion **boolean**. Ein Argument, das nicht vom Typ *node-set* ist, kann nicht in eine Knotenmenge konvertiert werden.

Anmerkung des Übersetzers:

Dieser letzte Satz stimmt insofern, als es keine automatische Konvertierung in eine Knotenmenge gibt. Zusätzliche Funktionen können durchaus Werte anderer Typen als Parameter entgegennehmen und eine Knotenmenge zurückliefern. In vielen XSLT-1.0-konformen Prozessoren existiert beispielsweise eine Erweiterungsfunktion `node-set`, die für einen *Ergebnisteilbaum* dessen äquivalente Knotenmenge zurückliefert.

[16]	FunctionCall	::=	FunctionName '(' (Argument (',' Argument)*)? ')'
[17]	Argument	::=	Expr

3.3 Knotenmengen

Ein Lokalisierungspfad kann als Ausdruck benutzt werden. Ein solcher Ausdruck liefert die durch den Pfad ausgewählte Knotenmenge.

Der Operator | berechnet die Vereinigung seiner Operanden, welche jeweils Knotenmengen sein müssen.

Anmerkung des Übersetzers:

Da es sich um eine Vereinigung von Mengen handelt, ist ein identischer Knoten in beiden Operanden in der Ergebnisknotenmenge auch nur einmal vorhanden. Zusammen mit der Funktion **count** (siehe [**4.1 Funktionen auf Knotenmengen**]) lässt sich so die Identität zweier Knoten feststellen.

Die Knotenmenge $a ist in der Knotenmenge $b enthalten, falls `count($b) = count ($a | $b)`. $a und $b sind identisch, wenn $a in $b und $b in $a enthalten ist. Es handelt sich um einzelne Knoten, wenn außerdem `count($a) = 1` ist. Achtung: Der Operator = bestimmt die Gleichheit der Zeichenkettenwerte zweier Knoten (siehe [**3.4 Boolesche Werte**]), nicht deren Identität.

XPath definiert keine Operatoren für die Bestimmung von Durchschnitt und Differenz zweier Knotenmengen. Basierend auf dem Teilmengentest lassen sich diese Operationen allerdings berechnen:

Durchschnitt von $a und $b:

```
$a[count(.|$b) = count($b)]
```

Differenz von $a und $b:

```
$a[count(.|$b) != count($b)]
```

Damit lässt sich nun auch testen, ob der Kontextknoten ein Attributknoten ist (analog für Namensraumknoten):

```
count(.|../@*) = count(../@*)
```

Die bedingte Auswahl einer Knotenmenge aus zwei Alternativen abhängig von einem logischen Ausdruck kann durch folgende Konstruktion erreicht werden:

```
node-set1[boolean-test] | node-set2[not(boolean-test)]
```

In den Programmiersprachen C, C++ und Java stellt der Fragezeichenoperator `?:` diese Funktionalität bereit. Eine analoge Anwendung für Zeichenketten wird im Zusammenhang mit der Funktion **substring** in Kapitel [**4.2 Zeichenkettenfunktionen**] vorgestellt.

In XSLT-Mustern, die eine Teilmenge der XPath-Ausdrücke bilden, wird der Operator | verwendet, um mögliche Alternativen anzugeben.

Prädikate werden zum Filtern von Ausdrücken in der gleichen Weise wie in Lokalisierungspfaden benutzt. Es ist ein Fehler, falls das Ergebnis des zu filternden Ausdrucks keine Knotenmenge ist. Das Prädikat filtert die Knotenmenge bezüglich der Kindachse.

> **Anmerkung:** Die Bedeutung eines Prädikats hängt entscheidend davon ab, welche Achse angewendet wird. Zum Beispiel liefert `preceding::foo[1]` das erste `foo`-Element in *umgekehrter Dokumentordnung*, weil die für das Prädikat `[1]` anzuwendende Achse die Vorgängerachse (preceding) ist. Demgegenüber liefert `(preceding::foo)[1]` das erste `foo`-Element in *Dokumentordnung*, weil die Achse, die in diesem Fall für das Prädikat `[1]` gilt, die Kindachse ist.

Anmerkung des Übersetzers:

Auf diesen Unterschied soll noch einmal deutlich hingewiesen werden: Prädikate, die Bestandteil eines Lokalisierungsschrittes sind, filtern eine Knotenmenge bezüglich der im Lokalisierungsschritt verwendeten Achse. Für jeden Knoten der Knotenmenge ist daher dessen Näheposition relevant. Prädikate, die auf einen XPath-Ausdruck angewendet werden, interpretieren die betreffenden Knoten immer in Dokumentordnung, da laut Definition in solch einem Fall die Kindachse anzuwenden ist.

Im obigen Beispiel `preceding::foo[1]` ist das Prädikat `[1]` Bestandteil des Lokalisierungsschrittes `preceding::foo[1]`, während bei `(preceding::foo)[1]` das Prädikat `[1]` auf den Ausdruck `(preceding::foo)` angewendet wird (welcher in diesem Fall ein Lokalisierungsschritt ohne Prädikat ist).

Das folgende Beispiel stellt den Sachverhalt aus einer praxisnäheren Sicht dar. Es gibt zwar die Achse `ancestor-or-self`, welche neben allen Vorfahren auch den Kontextknoten auswählt, es gibt aber keine entsprechende Achse `preceding-or-self`, die alle Vorgänger inklusive des Kontextknotens auswählen könnte. Die gewünschte Knotenmenge muss also durch eine Vereinigung konstruiert werden: `preceding::node() | self::node()`. Bei der Anwendung eines oder mehrerer Prädikate auf die entstehende Menge, etwa `(preceding::node() | .)[@id][1]`, muss man beachten, dass die Knoten nun in Dokumentordnung gefiltert werden. Der Ausdruck liefert damit den ersten Knoten im Dokument, der ein Vorgänger des Kontextknotens ist und ein Attribut `id` besitzt, und nicht den zum Kontextknoten nächsten Knoten mit dieser Eigenschaft.

Bei der Betrachtung des Beispiels `(preceding-sibling::* | following-sibling::*)[1]` wird klar, dass für solche Ausdrücke eine von den beteiligten Achsen abhängende Knotenordnung nicht praktikabel ist.

An dieser Stelle sei darauf hingewiesen, dass über die in [XSLT] definierte Funktion **document** auch Knotenmengen aus verschiedenen Dokumenten miteinander vereinigt werden können. In diesem Fall gibt es keine definierte Dokumentordnung für die Vereinigungsmenge mehr. Die Anwendung eines entsprechenden Prädikats ist damit implementationsabhängig. Entsprechendes gilt bei einer Knotenmenge, die die Attribute eines Elements enthält.

Die Operatoren `/` und `//` verbinden einen Ausdruck und einen relativen Lokalisierungspfad. Es ist ein Fehler, wenn die Berechnung des Ausdrucks keine Knotenmenge ergibt. Der Operator `/` arbeitet dabei in der gleichen Weise wie in einem Lokalisierungspfad. Ebenso wie in Lokalisierungspfaden steht `//` abkürzend für `/descendant-or-self::node()/`.

Es gibt keine Objekte, die in eine Knotenmenge konvertiert werden können.

Anmerkung des Übersetzers:

Angenommen, eine Variable namens `divs` enthält eine Knotenmenge von diversen `div`-Elementen (`div1`, `div2`, etc). Dann kann mittels `$divs[1]` auf das erste dieser Elemente zugegriffen werden. `$divs/@id` liefert die Menge der `id`-Attributknoten der Elemente aus `$divs`, `$divs//image` liefert die Menge aller `image`-Elemente, die Nachkommen eines in `$divs` enthaltenen `div`-Elements sind.

Dabei ist zu beachten, dass sich einem Ausdruck anschließende Lokalisierungsschritte immer auf die Position der Knoten im XML-Dokument beziehen und nicht auf die durch den Ausdruck berechnete Knotenmenge. Beispielsweise bestimmt `$divs[1]/preceding-sibling::*` den nachfolgenden Geschwisterknoten des ersten Knotens aus `$divs` im XML-Dokument, und *nicht* den "Nachfolger" in `$divs`, also `$divs[2]`. Allgemein gesprochen gibt es keinen Weg, der es erlaubt, ausgehend von einem Kontextknoten, der Element einer zuvor bestimmten Knotenmenge ist, auf die anderen Knoten dieser Menge zuzugreifen. Das betrifft insbesondere Ausdrücke, die in Prädikaten oder im Körper der XSLT-Anweisung `xsl:for-each` auftreten.

Ausdrücke, speziell Variablen, dürfen nur vor `/` und `//` auftreten. Es ist nicht möglich, Ausdrücke dynamisch zusammenzusetzen, wie man es etwa mit `/root/$element` versuchen könnte. Eine Möglichkeit, innerhalb eines Pfades dynamisch ein bestimmtes Element auszuwählen, wird im Zusammenhang mit der Funktion **name** vorgestellt.

Wie schon gesagt wurde, muss ein Ausdruck, dem ein Prädikat oder einer der Operatoren `/` und `//` folgt, immer eine Knotenmenge als Ergebnis liefern. Da Objekte anderer Typen nicht in eine Knotenmenge konvertiert werden können, führt die Auswertung eines Ausdrucks, der keine Knotenmenge ergibt, in diesem Fall zu einem Fehler. XPath nutzende Spezifikationen und Implementationen können jedoch explizit zusätzliche Funktionen definieren, die beliebige Objekte in Knotenmengen überführen.

[18]	UnionExpr	::=	PathExpr
			| UnionExpr '|' PathExpr
[19]	PathExpr	::=	LocationPath
			| FilterExpr
			| FilterExpr '/' RelativeLocationPath
			| FilterExpr '//' RelativeLocationPath
[20]	FilterExpr	::=	PrimaryExpr
			| FilterExpr Predicate

3.4 Boolesche Werte

Ein Objekt vom Typ *boolean* kann zwei Werte annehmen, *wahr* und *falsch*.

Die Berechnung eines `or`-Ausdrucks erfolgt, indem jeder der Operanden berechnet und in einen booleschen Wert wie beim Aufruf der Funktion **boolean** konvertiert wird. Das Ergebnis ist der Wert *wahr*, wenn einer der beiden Werte *wahr* ist, und andernfalls *falsch*. Der rechte Operand wird nicht mehr ausgewertet, wenn der linke Operand *wahr* ergibt.

Die Berechnung eines `and`-Ausdrucks erfolgt, indem jeder der Operanden berechnet und in einen booleschen Wert wie beim Aufruf der Funktion **boolean** konvertiert wird. Das Ergebnis ist der Wert *wahr*, wenn beide Werte *wahr* sind, und andernfalls *falsch*. Der rechte Operand wird

nicht mehr ausgewertet, wenn der linke Operand *falsch* ergibt.

Anmerkung des Übersetzers:

Mit dieser Regelung lässt sich die Berechnung von Teilausdrücken und der Aufruf enthaltener Funktionen verhindern. Abgesehen von Performance-Aspekten hat sie im Zusammenhang mit XSLT allerdings nicht die gleiche Bedeutung wie in anderen Programmiersprachen.

So arbeiten alle XSLT-Funktionen ohne Seiteneffekte (sie produzieren weder Ausgaben noch ändern sie Variableninhalte); Funktionsparameter werden automatisch in den geforderten Typ konvertiert und mathematische Operationen liefern immer einen definierten Wert. Lediglich Typfehler können auftreten, falls ein Ausdruck als Operand oder Funktionsparameter eine Knotenmenge verlangt. Allerdings gibt es in XPath keine Möglichkeit festzustellen, ob ein Teilausdruck vom Typ Knotenmenge ist.

Da sich über Erweiterungsmechanismen jedoch Funktionen definieren lassen, die die genannten Eigenschaften nicht mehr besitzen, kann über eine Verknüpfung mit `or` oder `and` der Aufruf solcher Funktionen bei Bedarf verhindert werden.

Die Berechnung eines EqualityExpr-Ausdrucks (der nicht allein ein RelationalExpr-Ausdruck ist) oder eines RelationalExpr-Ausdrucks (der nicht allein ein AdditiveExpr-Ausdruck ist) geschieht, indem die Objekte miteinander verglichen werden, die im Ergebnis der Auswertung der beiden Operanden entstehen. Die folgenden drei Absätze definieren den Vergleich zwischen den daraus resultierenden Objekten. Erst werden Vergleiche, die Knotenmengen betreffen, über Vergleiche definiert, die keine Knotenmengen betreffen; dies geschieht einheitlich für =, !=, <=, <, >= und >. Dann werden Vergleiche, die keine Knotenmengen betreffen, für = und != definiert. Schließlich werden Vergleiche, die keine Knotenmengen betreffen, für <=, <, >= und > definiert.

Wenn beide zu vergleichenden Objekte Knotenmengen sind, so liefert ein Vergleich genau dann den Wert *wahr*, wenn es einen Knoten aus der ersten Knotenmenge und einen Knoten aus der zweiten Knotenmenge gibt, sodass das Ergebnis des Vergleichs der Zeichenkettenwerte dieser beiden Knoten *wahr* ergibt. Wenn eines der zu vergleichenden Objekte eine Knotenmenge und das andere eine Zahl ist, dann liefert ein Vergleich genau dann den Wert *wahr*, wenn es einen Knoten in der Knotenmenge gibt, sodass der Vergleich zwischen der Zahl und dem Ergebnis der Konvertierung des Zeichenkettenwerts dieses Knotens zu einer Zahl über die Funktion **number** *wahr* ergibt. Wenn eines der zu vergleichenden Objekte eine Knotenmenge und das andere eine Zeichenkette ist, dann liefert ein Vergleich genau dann den Wert *wahr*, wenn es einen Knoten in der Knotenmenge gibt, sodass der Vergleich zwischen der Zeichenkette und dem Zeichenkettenwert dieses Knotens *wahr* ergibt. Wenn eines der zu vergleichenden Objekte eine Knotenmenge und das andere ein boolescher Wert ist, dann liefert ein Vergleich genau dann den Wert *wahr*, wenn es einen Knoten in der Knotenmenge gibt, sodass der Vergleich zwischen dem booleschen Wert und dem Ergebnis der Konvertierung des Zeichenkettenwerts dieses Knotens zu einem boolschen Wert über die Funktion **boolean** *wahr* ergibt.

Wenn keines der zu vergleichenden Objekte eine Knotenmenge ist und als Operator = oder != vorkommt, so werden die Objekte wie nachfolgend beschrieben in einen gemeinsamen Typ konvertiert und anschließend verglichen. Wenn wenigstens eines der zu vergleichenden Objekte ein boolescher Wert ist, wird jedes Objekt wie bei der Anwendung der Funktion **boolean** in einen booleschen Wert konvertiert. Wenn wenigstens eines der zu vergleichenden Objekte eine Zahl ist, wird jedes Objekt wie bei der Anwendung der Funktion **number** in eine Zahl konvertiert. Andernfalls werden beide Objekte wie bei der Anwendung der Funktion **string** in Zeichenketten

konvertiert. Der Vergleich = liefert als Ergebnis genau dann den Wert *wahr*, wenn beide Objekte gleich sind; der Vergleich `!=` liefert als Ergebnis genau dann den Wert *wahr*, wenn beide Objekte ungleich sind. Zahlen werden gemäß IEEE 754 [IEEE 754] verglichen. Zwei boolesche Werte sind gleich, wenn sie entweder beide *wahr* oder beide *falsch* sind. Zwei Zeichenketten sind genau dann gleich, wenn sie aus derselben Folge von UCS-Zeichen bestehen.

> **Anmerkung:** Wenn `$x` mit einer Knotenmenge belegt ist, dann bedeutet `$x="foo"` nicht dasselbe wie `not($x!="foo")`: Der erste Vergleich ergibt genau dann *wahr*, wenn *ein* Knoten in `$x` den Zeichenkettenwert `foo` hat; der zweite ergibt genau dann *wahr*, wenn *alle* Knoten in `$x` den Zeichenkettenwert `foo` haben.

Anmerkung des Übersetzers:

Der Vergleich `$x!="foo"` ist genau dann *wahr*, wenn es wenigstens einen Knoten aus `$x` gibt, für den die Ungleichheit gilt, d.h. *falsch*, wenn es keinen solchen Knoten gibt. Damit ist `not($x!="foo")` *wahr*, wenn es keinen Knoten aus `$x` gibt, dessen Zeichenkettenwert verschieden von `foo` ist, d.h. wenn alle Knoten den Zeichenkettenwert `foo` besitzen.

Ein häufigerer Fall dürfte der Test auf Ungleichheit sein. Man möchte z.B. feststellen, ob der Wert eines Ausdrucks verschieden von allen Kindelementen `entry` ist. Die Lösung lautet in diesem Fall nicht `entry!="foo"` (hier wird auf alle `entry`-Kindelemente zugegriffen und getestet, ob unter diesen eines existiert, das verschieden von der Zeichenkette `"foo"` ist), sondern `not(entry="foo")`.

Bemerkenswert ist noch der Fall, dass die beteiligte Knotenmenge leer ist. Ein Vergleich `@type!="warning"` ist *falsch*, wenn kein Attribut `type` existiert. Dagegen liefert `not(@type="warning")` in diesem Fall den Wert *wahr*.

Es sei noch einmal darauf hingewiesen, dass für Knotenmengen mit den Operatoren = und != nicht die Identität von Knoten getestet wird, sondern die ihrer Zeichenkettenwerte. Zwei Knoten können unter Zuhilfenahme des Operators | auf Identität getestet werden, siehe Anmerkung in Kapitel [**3.3 Knotenmengen**].

Wenn keines der zu vergleichenden Objekte eine Knotenmenge ist und als Operator <=, <, >= oder > vorkommt, so werden beide Objekte in Zahlen konvertiert und anschließend gemäß IEEE 754 verglichen. Der Vergleich < ergibt genau dann den Wert *wahr*, wenn die erste Zahl kleiner als die zweite Zahl ist. Der Vergleich <= ergibt genau dann den Wert *wahr*, wenn die erste Zahl kleiner oder gleich der zweiten Zahl ist. Der Vergleich > ergibt genau dann den Wert *wahr*, wenn die erste Zahl größer als die zweite Zahl ist. Der Vergleich >= ergibt genau dann den Wert *wahr*, wenn die erste Zahl größer oder gleich der zweiten Zahl ist.

Anmerkung des Übersetzers:

Somit sind Vergleiche, an denen Knotenmengen beteiligt sind, dann erfüllt, wenn sich wenigstens ein Knoten aus der jeweiligen Menge finden lässt, dessen Zeichenkettenwert den Vergleich erfüllt. Insbesondere liefert der Vergleich mit wenigstens einer leeren Menge in jedem Fall den Wert *falsch*. Sind nur Werte verschiedener skalarer Typen beteiligt, so wird in der Rangfolge "boolescher Wert – Zahl – Zeichenkette" ein gemeinsamer Typ gesucht und der jeweils andere Wert konvertiert. Größenvergleiche sind nur für Zahlen definiert.

Unter Ausnutzung dieser Regeln kann z.B. die kleinste Zahl in einer Knotenmenge `$set` folgendermaßen bestimmt werden:

```
$set[not(. > $set)]
```

Ein Vergleich mit dem speziellen Zahlenwert NaN (Not a Number) liefert immer den Wert *falsch*, selbst bei `number('NaN')=number('NaN')`. Bei `'NaN'` handelt es sich hier nicht um eine spezielle Zeichenkette, die in den Wert NaN konvertiert wird, sondern einfach um eine, die sich nicht konvertieren lässt. Können in der Beispielmenge `$set` auch Knoten auftreten, deren Zeichenkettenwert sich nicht in eine Zahl konvertieren lässt, könnte man den obigen Ausdruck auf folgende, etwas unkonventionelle Weise vervollständigen:

```
$set[number()=number() and not(. > $set)]
```

Vorsicht ist geboten, wenn die Werte einer Knotenmenge vor dem Vergleich durch eine Funktion mit skalarem Argumenttyp bearbeitet werden sollen. Übergibt man der Funktion die gesamte Knotenmenge als Argument, wird nur mit dem Funktionswert des ersten Knotens verglichen. Die spezielle Semantik des Vergleichs mit Knotenmengen geht verloren. In der Anmerkung zur Funktion **normalize-space** wird dies an einem Beispiel ausführlicher erläutert.

XPath stellt keine Möglichkeit zur Verfügung, mit der man Zeichenketten lexikographisch der Größe nach vergleichen könnte. Abhängig vom Anwendungsfall lassen sich in XSLT solche Vergleiche durch die Programmierung rekursiver Templates oder die Benutzung des `xsl:sort`-Elements realisieren.

Anmerkung: Wenn ein XPath-Ausdruck in einem XML-Dokument vorkommt, müssen alle Operatoren < und <= gemäß den XML-1.0-Regeln geschützt werden, zum Beispiel als `<` und `<=`. Im folgenden Beispiel ist der Wert des Attributes `test` ein XPath-Ausdruck:

```
<xsl:if test="@value &lt; 10">...</xsl:if>
```

[21]	OrExpr	::=	AndExpr
			| OrExpr 'or' AndExpr
[22]	AndExpr	::=	EqualityExpr
			| AndExpr 'and' EqualityExpr
[23]	EqualityExpr	::=	RelationalExpr
			| EqualityExpr '=' RelationalExpr
			| EqualityExpr '!=' RelationalExpr
[24]	RelationalExpr	::=	AdditiveExpr

| RelationalExpr '<' AdditiveExpr
| RelationalExpr '>' AdditiveExpr
| RelationalExpr '<=' AdditiveExpr
| RelationalExpr '>=' AdditiveExpr

Anmerkung: Mit der obigen Grammatik ergibt sich folgende Vorrangfolge (kleinster Vorrang zuerst):

- `or`
- `and`
- `=, !=`
- `<=, <, >=, >`

Alle Operatoren sind links-assoziativ. Beispielsweise ist `3 > 2 > 1` äquivalent zu `(3 > 2) > 1`, was den Wert *falsch* ergibt.

Anmerkung des Übersetzers:

Der Vergleich `3 > 2` ergibt zunächst den Wert *wahr*. Dieses Ergebnis wird aufgrund des folgenden Vergleichsoperators > in die Zahl 1 konvertiert, sodass nun `1 > 1` berechnet wird, was den Wert *falsch* liefert. Auf analoge Weise kann man sich überlegen, dass der Ausdruck `2 = 1 = 0` *wahr*, der Ausdruck `0 = 0 = 0` hingegen *falsch* ergibt.

Hier ist Vorsicht geboten, da solche Ausdrücke gemäß der XPath-Grammatik erlaubt sind, aber nicht die Semantik besitzen, die man auf den ersten Blick erwarten würde. Sie verhalten sich allerdings genauso wie beispielsweise in den Programmiersprachen C und C++.

3.5 Zahlen

Ein Wert vom Typ *number* repräsentiert eine Gleitkommazahl. Eine Zahl kann jeden beliebigen, doppelt-genauen 64-Bit-Wert des Formats IEEE 754 [IEEE 754] annehmen. Dies beinhaltet den speziellen Wert "Not-a-Number" (NaN), positiv und negativ unendlich, sowie positiv und negativ Null. Für eine Zusammenfassung der wichtigsten Regeln des IEEE-754-Standards siehe Abschnitt 4.2.3 in [JLS].

Anmerkung des Übersetzers:

Die genannten speziellen Werte entstehen dann, wenn eine Rechenoperation einen Überlauf produzieren würde bzw. das Ergebnis nicht definiert ist. Beim Rechnen mit Zahlen in XPath können keine Fehler oder Ausnahmen auftreten.

An dieser Stelle sei bereits kurz auf die Produktion für Number in [**3.7 Lexikalische Struktur**] hingewiesen. Zahlen in XPath sind Gleitkommazahlen ohne Exponentendarstellung. Eine Schreibweise 2.99792E+08 ist nicht zulässig. Sie können ein negatives, aber kein explizites positives Vorzeichen besitzen. Soll einer der speziellen Werte wie z.B. positiv unendlich verwendet werden, muss dieser ermittelt werden, etwa durch `1 div 0`.

Es gibt in XPath weder einen speziellen Typ für ganzzahlige Werte noch gesonderte Zahlendarstellungen, die eine Zahl als Oktal- oder Hexadezimalzahl interpretieren, wie dies in vielen Programmiersprachen möglich ist.

Die numerischen Operatoren konvertieren ihre Operanden in Zahlen, so wie bei einem Aufruf der Funktion **number**.

Der Operator + addiert.

Der Operator - subtrahiert.

Anmerkung des Übersetzers:

Gemäß Errata-Dokument [XPath Errata] ist die Semantik des einstelligen Operators - nicht spezifiziert. Stattdessen muss dieser letzte Absatz lauten:

> Der zweistellige Operator - subtrahiert. Der einstellige Operator - berechnet die Negation. Beachten Sie, dass `-0` negativ Null ergibt.

Anmerkung: Da XML innerhalb von Namen das Zeichen - erlaubt, muss der Operator - typischerweise von einem Leerraumzeichen angeführt werden. Zum Beispiel ergibt `foo-bar` eine Knotenmenge, die die Kindelemente namens `foo-bar` enthält; `foo - bar` ergibt die Differenz aus den Werten, die durch Konvertierung des Zeichenkettenwertes des ersten `foo`-Kindelements in eine Zahl und durch Konvertierung des Zeichenkettenwertes des ersten `bar`-Kindelements in eine Zahl entstehen.

Anmerkung des Übersetzers:

Gemäß Errata-Dokument [XPath Errata] ist die Semantik des Operators * nicht spezifiziert. An dieser Stelle muss folgender Absatz eingefügt werden:

> Der Operator * berechnet eine Gleitkomma-Multiplikation gemäß IEEE 754. Beachten Sie: Falls das Ergebnis nicht NaN ist, ist das Ergebnis genau dann positiv, wenn beide Operanden das gleiche Vorzeichen besitzen.

Das Zeichen * dient zugleich als Knotentest zur Auswahl beliebiger Elemente. Welche Semantik ein * innerhalb eines XPath-Ausdrucks hat, hängt damit von den umgebenden Tokens in diesem Ausdruck ab (siehe [**3.7 Lexikalische Struktur**]).

Der Operator `div` berechnet eine Gleitkomma-Division gemäß IEEE 754.

Anmerkung des Übersetzers:

Gemäß Errata-Dokument [XPath Errata] muss an dieser Stelle folgender Satz eingefügt werden:

> Beachten Sie: Falls das Ergebnis nicht NaN ist, ist das Ergebnis genau dann positiv, wenn beide Operanden das gleiche Vorzeichen besitzen.

Der Schrägstrich / ist nicht der Divisionsoperator, da dieser bereits als Pfadoperator zum Verbinden von Lokalisierungsschritten sowie als Symbol für den Wurzelknoten benutzt wird. Im Gegensatz zum Zeichen * gibt es hier nicht mehrere Interpretationsmöglichkeiten.

Der Operator `mod` liefert den Rest einer ganzzahligen Division. Beispiele:

- `5 mod 2` ergibt `1`
- `5 mod -2` ergibt `1`
- `-5 mod 2` ergibt `-1`
- `-5 mod -2` ergibt `-1`

Anmerkung: `mod` berechnet dasselbe wie der Operator `%` in Java und ECMAScript.

Anmerkung: Er berechnet nicht dasselbe wie die IEEE-754-Rest-Operation, welche den Rest einer gerundeten Division liefert.

Anmerkung des Übersetzers:

Der `mod`-Operator berechnet den genauen Rest, der sich bei der ganzzahligen Division zweier Gleitkommazahlen ergibt, ohne die Operanden zuvor auf ganze Zahlen zu runden.

Numerische Ausdrücke

[25]	AdditiveExpr	::=	MultiplicativeExpr
			| AdditiveExpr '+' MultiplicativeExpr
			| AdditiveExpr '-' MultiplicativeExpr
[26]	MultiplicativeExpr	::=	UnaryExpr
			| MultiplicativeExpr MultiplyOperator UnaryExpr
			| MultiplicativeExpr 'div' UnaryExpr
			| MultiplicativeExpr 'mod' UnaryExpr
[27]	UnaryExpr	::=	UnionExpr
			| '-' UnaryExpr

3.6 Zeichenketten

Zeichenketten bestehen aus einer Folge von null oder mehr Zeichen, wobei Zeichen wie in der XML-Empfehlung [XML] definiert sind. Ein einzelnes XPath-Zeichen entspricht damit einem einzelnen abstrakten Unicode-Zeichen mit einem einzelnen korrespondierenden skalaren Wert (siehe [Unicode]); dies unterscheidet sich allerdings von einem 16-Bit-kodierten Unicode-Zeichen: Die durch Unicode definierte kodierte Zeichenrepräsentation eines abstrakten Zeichens mit einem skalaren Wert größer als U+FFFF ist ein Paar von 16-Bit Unicode-Codes (ein Surrogat-Paar). In vielen Programmiersprachen wird eine Zeichenkette als Folge von 16-Bit-kodierten Unicode-Zeichen repräsentiert; XPath-Implementationen in solchen Sprachen müssen sicherstellen, dass ein Surrogat-Paar korrekt als einzelnes XPath-Zeichen behandelt wird.

> **Anmerkung:** In Unicode ist es möglich, dass zwei Zeichenketten als identisch anzusehen sind, obwohl sie aus unterschiedlichen Folgen abstrakter Unicode-Zeichen bestehen. Zum Beispiel können einige Akzentzeichen entweder in einer vordefinierten (precomposed) oder einer zerlegten (decomposed) Form repräsentiert werden. Damit können XPath-Ausdrücke unerwartete Resultate liefern, es sei denn, sowohl die Zeichen im XPath-Ausdruck als auch die im XML-Dokument wurden zu einer kanonischen Form normalisiert (siehe [Character Model]).

Anmerkung des Übersetzers:

Ein zusammengesetztes Zeichen, das sich auch als vordefiniertes Zeichen kodieren lässt, wird durch einen einzigen Unicode-Code repräsentiert. Beispielsweise lässt sich der Umlaut »ü« als U+00FC darstellen. Zugleich kann dieser Buchstabe auch wie jedes zusammengesetzte Zeichen in der zerlegten Form durch die Folge der beiden Codes U +0075 (»u«) und U+0308 (combining diaeresis) kodiert werden.

3.7 Lexikalische Struktur

Beim Zerlegen in einzelne Tokens wird immer das längstmögliche Token zurückgeliefert.

Zur besseren Lesbarkeit können Leerraumzeichen innerhalb von Ausdrücken verwendet werden, auch wenn es nicht explizit durch die Grammatik erlaubt wurde: ExprWhitespace kann

innerhalb von Ausdrücken frei vor oder nach beliebigen ExprTokens eingefügt werden.

Anmerkung des Übersetzers:

An dieser Stelle sei auf die Anmerkung zum Operator für die Subtraktion, das zweistellige Minus, in [**3.5 Zahlen**] hingewiesen. Der erste Absatz legt fest, dass »`foo-bar`« nur als einzelnes Token interpretiert werden darf und nicht als Folge der Tokens »`foo`«, »`-`« und »`bar`«. Der zweite Absatz erlaubt nun explizit, beispielsweise den Minus-Operator mittels Leerraumzeichen als einzeln zu interpretierendes Token zu kennzeichnen.

Üblicherweise unterscheidet man bei der Definition einer Sprache zwischen lexikalischen Produktionen, die den Aufbau der lexikalischen Einheiten, so genannter Tokens festlegen, und syntaktischen Produktionen, die die mögliche Kombination dieser Tokens zu komplexeren Konstrukten beschreiben. In der XPath-Spezifikation sind diese beiden Arten von Produktionen allerdings nicht streng voneinander abgegrenzt. Der Hauptunterschied zwischen lexikalischen und syntaktischen Produktionen besteht darin, dass zwischen einzelnen Tokens Leerraumzeichen auftreten dürfen, nicht jedoch innerhalb eines Tokens. Die Produktion für das Nichtterminal ExprToken stellt damit die oberste lexikalische Produktion dar. Daraus ergibt sich, dass ein Leerzeichen zwischen `@` und einem QName zur Abkürzung der `attribute`-Achse erlaubt ist, nicht aber zwischen `$` und einem QName bei Variablenreferenzen.

Die folgenden speziellen Regeln für die Zerlegung in Tokens müssen in der angegebenen Reihenfolge angewendet werden, um die Grammatik ExprToken eindeutig zu machen:

- Wenn es ein vorhergehendes Token gibt und dieses Token kein `@`, `::`, `(`, `[`, `,` oder ein Operator ist, dann muss ein `*` als MultiplyOperator und ein NCName als OperatorName erkannt werden.
- Falls das einem NCName folgende Zeichen (möglicherweise nach dazwischenliegendem ExprWhitespace) das Zeichen `(` ist, dann muss das Token als NodeType oder als FunctionName erkannt werden.
- Falls die einem NCName folgenden beiden Zeichen (möglicherweise nach dazwischenliegendem ExprWhitespace) die Zeichen `::` sind, dann muss das Token als AxisName erkannt werden.
- Andernfalls darf das Token nicht als MultiplyOperator, als OperatorName, als NodeType, als FunctionName oder als AxisName erkannt werden.

Anmerkung des Übersetzers:

Im zweiten Aufzählungspunkt des Originaldokuments hat sich ein Fehler eingeschlichen. Da Funktionsnamen mit einem Präfix ausgestattet sein können, muss laut Errata-Dokument [XPath Errata] auf QName statt auf NCName verwiesen werden. Alle Standardfunktionen aus XPath, XSLT und XPointer besitzen zwar nur Namen ohne Präfix, der Erweiterungsmechanismus in XSLT erlaubt jedoch XSLT-Implementationen, zusätzliche Funktionen aus einem proprietären Namensraum zur Verfügung zu stellen.

Das hier diskutierte Problem der Mehrdeutigkeit umgeht man in vielen anderen Programmiersprachen durch die Definition von Schlüsselwörtern, die dann für frei wählbare Bezeichner nicht mehr zur Verfügung stehen. Ein XML-Autor ist jedoch frei in seiner Wahl der Element- und Attributnamen, also muss auch die Sprache XPath damit umgehen können. Die gefundene Regelung ermöglicht eine kompakte Schreibweise für XPath-Ausdrücke und bürdet die Last der eindeutigen Interpretation der jeweiligen XPath-Implementation auf.

Die folgenden Beispiele zeigen, wie Bezeichner (und der *-Operator) abhängig vom Kontext unterschiedlich interpretiert werden müssen:

- `***`

bestimmt jeweils den in eine Zahl konvertierten Zeichenkettenwert der ersten Knoten in den durch * repräsentierten Knotenmengen aller Kindelemente und multipliziert diese miteinander, d.h. das erste und dritte * werden als Lokalisierungspfad, das mittlere * als Multiplikationsoperator interpretiert.

- `and or mod`

ergibt den Wert *wahr*, wenn der Kontextknoten wenigstens ein Kindelement namens `and` oder ein Kindelement namens `mod` besitzt (Operatorname versus Elementname).

- `text and text()`

ergibt den Wert *wahr*, wenn der Kontextknoten sowohl wenigstens ein Element namens `text` als auch wenigstens einen Textknoten als Kinder besitzt (Knotentyp versus Elementname).

- `position() = position`

ergibt den Wert *wahr*, wenn die aktuelle Kontextposition mit dem in eine Zahl konvertierten Zeichenkettenwert eines Kindelements namens `position` übereinstimmt (Funktionsname versus Elementname).

- `parent or parent::child`

ergibt den Wert *wahr*, wenn der Kontextknoten ein Kindelement namens `parent` oder ein Elternelement namens `child` besitzt (Achsenname versus Elementname).

Lexikalische Struktur von Ausdrücken

[28] ExprToken ::= '(' | ')' | '[' | ']' | '.' | '..' | '@' | ',' | '::'

			| NameTest
			| NodeType
			| Operator
			| FunctionName
			| AxisName
			| Literal
			| Number
			| VariableReference
[29]	Literal	::=	'"' [^"]* '"'
			| "'" [^']* "'"
[30]	Number	::=	Digits ('.' Digits?)?
			| '.' Digits
[31]	Digits	::=	[0-9]+
[32]	Operator	::=	OperatorName
			| MultiplyOperator
			| '/' | '//' | '|' | '+' | '-' | '=' | '!=' | '<' | '<=' | '>' | '>='
[33]	OperatorName	::=	'and' | 'or' | 'mod' | 'div'
[34]	MultiplyOperator	::=	'*'
[35]	FunctionName	::=	QName - NodeType
[36]	VariableReference	::=	'$' QName
[37]	NameTest	::=	'*'
			| NCName ':' '*'
			| QName
[38]	NodeType	::=	'comment'
			| 'text'
			| 'processing-instruction'
			| 'node'
[39]	ExprWhitespace	::=	S

4 Bibliothek der Grundfunktionen

Dieser Abschnitt beschreibt die Funktionen, die bei einer XPath-Implementation immer in der bei der Auswertung von Ausdrücken benutzten Funktionsbibliothek enthalten sein müssen.

Jede Funktion in der Funktionsbibliothek wird über einen Funktionsprototypen spezifiziert, der den Rückgabetyp, den Namen der Funktion und die Typen der Argumente angibt. Falls ein Argument von einem Fragezeichen gefolgt wird, so ist es optional; andernfalls ist es obligatorisch.

Anmerkung des Übersetzers:

Ein Funktionsargument, dem ein Stern * folgt, darf keinmal, einmal oder mehrmals auftreten. Der Argumenttyp *object* steht für einen beliebigen Typ.

4.1 Funktionen auf Knotenmengen

Funktion: *number* **last**()

Die Funktion **last** liefert eine Zahl, die gleich der Kontextgröße des Kontexts des ausgewerteten Ausdrucks ist.

Funktion: *number* **position**()

Die Funktion **position** liefert eine Zahl, die gleich der Kontextposition im Kontext des ausgewerteten Ausdrucks ist.

Funktion: *number* **count**(*node-set*)

Die Funktion **count** liefert die Anzahl der Knoten der übergebenen Knotenmenge.

Anmerkung des Übersetzers:

Für den letzten Knoten einer Knotenmenge gilt damit die Gleichheit `position()=last()`. Beachten Sie, dass die Nummerierung der Knoten bei 1 beginnt.

Während **last** also die Kontextgröße zurückgibt, d.h. die Anzahl der Knoten der Knotenmenge, in der sich der Kontextknoten befindet, berechnet **count** die Anzahl der Knoten einer beliebigen als Argument zu übergebenden Knotenmenge. Es gibt keinen XPath-Ausdruck, der für einen beliebigen Knoten dessen Position in einer vorgegebenen Knotenmenge berechnet.

In bestimmten Situationen lässt sich jedoch das Wissen über die Art der Beziehungen der Knoten in der Knotenmenge ausnutzen. Möchte man beispielsweise herausfinden, welche Position das erste `para`-Kind mit dem Attribut `type="warning"` unter allen `para`-Kindern besitzt, kann man einfach die diesem Element vorangehenden `para`-Geschwister zählen und 1 addieren:

```
count(para[@type="warning"]/preceding-sibling::para)+1
```

Diese Methode funktioniert hier deshalb, weil klar ist, dass alle `para`-Elemente Geschwister sind und auf die vorherigen Knoten der Menge daher über die Achse `preceding-sibling` zugegriffen werden kann. Wenn nicht bekannt ist, wie die Knotenmenge gebildet wurde, können die anderen Knoten auch nicht bestimmt und gezählt werden.

In Kapitel [**3.3 Knotenmengen**] wird im Zusammenhang mit dem Mengenvereinigungsoperator | gezeigt, wie die Funktion **count** zur Bestimmung von Durchschnitt und Differenz zweier Mengen genutzt werden kann.

Funktion: *node-set* **id**(*object*)

Die Funktion **id** wählt Elemente anhand ihrer eindeutigen ID aus (siehe [**5.2.1 Eindeutige IDs**]). Wenn als Argument eine Knotenmenge übergeben wird, so ergibt sich das Ergebnis aus der Vereinigung der Knotenmengen, die durch den Aufruf von **id** mit dem Zeichenkettenwert jedes Knotens aus der übergebenen Knotenmenge berechnet werden. Ist das Argument der Funktion **id** von einem beliebigen anderen Typ, so wird es wie bei einem Aufruf der Funktion **string** in

eine Zeichenkette konvertiert; die Zeichenkette wird in eine durch Leerraumzeichen getrennte Liste von Tokens aufgeteilt (Leerraumzeichen sind beliebige Folgen von Zeichen, die sich aus der Produktion S ableiten lassen). Das Ergebnis ist eine Knotenmenge, die die Elemente aus dem Dokument des Kontextknotens enthält, die eine eindeutige ID mit dem gleichen Wert wie eines der Tokens der Liste besitzen.

- `id("foo")` wählt die Elemente mit der eindeutigen ID `foo` aus.
- `id("foo")/child::para[position()=5]` wählt das fünfte `para`-Kindelement des Elements mit der eindeutigen ID `foo` aus.

Anmerkung des Übersetzers:

Diese Funktion ermöglicht die Auswertung von Attributen des Typs `IDREF` bzw. `IDREFS`. Da ein `IDREF`-Wert als ID an ein Element im gleichen XML-Dokument vergeben worden sein muss, lässt sich dieses Element mit Hilfe der Funktion **id** bestimmen. Handelt es sich um mehrere IDs als Wert eines `IDREFS`-Attributs, liefert **id** alle zugehörigen Elemente als Knotenmenge.

Entsprechend der obigen Definition gilt also:

```
id("foo bar baz") = id("foo") | id("bar") | id("baz")
```

Die Knoten in der Ergebnisknotenmenge werden durch ein folgendes Prädikat in Dokumentordnung gefiltert und nicht in der Reihenfolge der angegebenen IDs.

Dagegen wird bei der Übergabe einer Knotenmenge, wie im Beispiel `id(foo)`, von allen Knoten der Menge deren Zeichenkettenwert als Parameter verarbeitet und dieser wie ein `IDREFS`-Attribut interpretiert. Das Ergebnis ist die Vereinigung aller auf diese Weise bestimmten Knoten.

Die Funktion **id** kann aber nur korrekt arbeiten, wenn `ID`-wertige Attribute als solche erkannt werden. Dazu muss die DTD bekannt und vom Parser ausgewertet worden sein. Dies ist einer der wenigen Fälle, in denen das Ergebnis eines XPath-Ausdrucks von der Kenntnis der DTD des Dokuments abhängt. Fehlt diese Information, liefert **id** immer eine leere Knotenmenge.

Es gibt keine komplementäre Funktion, die für ein Element dessen ID zurückliefert. Unter Zuhilfenahme von **id** lässt sich diese Information allerdings herausfinden. Gesucht ist nämlich genau das Attribut, für das die Funktion **id** den Elternknoten dieses Attributs liefert:

```
@*[id(.) and count(id(.)|..)=1]
```

Funktion: *string* **local-name**(*node-set*?)

Die Funktion **local-name** liefert den lokalen Teil des erweiterten Namens des ersten Knotens in der Argumentknotenmenge bezüglich der Dokumentordnung. Falls die übergebene Knotenmenge leer ist oder der erste Knoten keinen erweiterten Namen besitzt, wird eine leere Zeichenkette zurückgegeben. Wird kein Argument übergeben, wird stattdessen eine Knotenmenge mit dem Kontextknoten als einzigem Element benutzt.

Anmerkung des Übersetzers:

Für Element- und Attributknoten gilt damit, dass die Funktion **local-name** den dem Doppelpunkt folgenden Teil des QName zurückgibt bzw. den vollständigen Namen, wenn dieser kein Präfix enthält. Die folgenden Beispiele demonstrieren dies unter der Voraussetzung, dass der als jeweiliges Argument übergebene Lokalisierungspfad nicht die leere Knotenmenge ergibt:

```
local-name(xhtml:body) = "body"
local-name(@xlink:href) = "href"
local-name(para) = "para"
```

Für Namensraumknoten liefert **local-name** das zugewiesene Präfix, also beispielsweise für die Deklaration `xmlns:xlink="http://www.w3.org/1999/xlink"` die Zeichenkette »`xlink`«. Für Namensraumknoten, die den voreingestellten Namensraum repräsentieren, wird die leere Zeichenkette zurückgegeben.

Für Processing Instructions liefert **local-name** das jeweilige Ziel, also beispielsweise für <? `xml-stylesheet href='style.xsl'?>` die Zeichenkette »`xml-stylesheet`«.

Für Textknoten, Kommentare und den Wurzelknoten liefert **local-name** die leere Zeichenkette.

Funktion: *string* **namespace-uri**(*node-set*?)

Die Funktion **namespace-uri** liefert den Namensraum-URI des erweiterten Namens des ersten Knotens in der Argumentknotenmenge bezüglich der Dokumentordnung. Falls die übergebene Knotenmenge leer ist, der erste Knoten keinen erweiterten Namen besitzt oder der Namensraum-URI des erweiterten Namens leer ist, wird eine leere Zeichenkette zurückgegeben. Wird kein Argument übergeben, wird stattdessen eine Knotenmenge mit dem Kontextknoten als einzigem Element benutzt.

> **Anmerkung:** Die von der Funktion **namespace-uri** zurückgegebene Zeichenkette ist außer für Element- oder Attributknoten immer leer.

Anmerkung des Übersetzers:

Beispiele: `namespace-uri(xhtml:body)` liefert den zum Präfix `xhtml` gehörenden Namensraum-URI (z.B. »`http://www.w3.org/1999/xhtml`«), entsprechend ergibt `namespace-uri(@xlink:href)` die Zeichenkette »`http://www.w3.org/1999/xlink`«, falls das Präfix `xlink` an diesen URI gebunden wurde. Für Elementknoten aus dem voreingestellten Namensraum liefert **namespace-uri** den dazugehörigen URI. Für Element- und Attributknoten, die keinem Namensraum angehören, wird die leere Zeichenkette zurückgegeben.

Funktion: *string* **name**(*node-set*?)

Die Funktion **name** liefert eine Zeichenkette mit einem QName, die den erweiterten Namen des ersten Knotens in der Argumentknotenmenge bezüglich der Dokumentordnung repräsentiert.

Der QName muss den erweiterten Namen unter Berücksichtigung der Namensraumdeklarationen repräsentieren, die für den Knoten gültig sind, dessen erweiterter Name repräsentiert wird. Typischerweise ist das der QName, der in der XML-Quelle vorkommt. Das muss nicht der Fall sein, wenn es für den Knoten Namensraumdeklarationen gibt, die dem gleichen Namensraum mehrere Präfixe zuordnen. Allerdings kann eine Implementation Informationen über das Originalpräfix speichern; in diesem Fall kann die Implementation sicherstellen, dass der zurückgegebene String immer der in der XML-Quelle benutzte QName ist. Falls die übergebene Knotenmenge leer ist oder der erste Knoten keinen erweiterten Namen besitzt, wird eine leere Zeichenkette zurückgegeben. Wird kein Argument übergeben, wird stattdessen eine Knotenmenge mit dem Kontextknoten als einzigem Element benutzt.

> **Anmerkung:** Die von der Funktion **name** gelieferte Zeichenkette ist die gleiche wie die von der Funktion **local-name** gelieferte, außer für Element- und Attributknoten.

Anmerkung des Übersetzers:

Für einen folgendermaßen definierten Elementknoten

```
<x:foo xmlns:x="urn:bar" xmlns:y="urn:bar" />
```

darf die Funktion **name** damit die Zeichenkette »`y:foo`« liefern, da durch das Präfix y der gleiche Namensraum repräsentiert wird wie durch `x`.

Die drei Funktionen **local-name**, **namespace-uri** und **name** werden in der folgenden Tabelle kurz zusammengefasst. Das Zeichen »`-`« steht dabei für die leere Zeichenkette:

	local-name	**namespace-uri**	**name**
[**5.1 Wurzelknoten**]	-	-	-
[**5.2 Elementknoten**]	Name ohne Präfix	URI des Namensraums, in dem sich das Element befindet	voller Name
[**5.3 Attributknoten**]	Name ohne Präfix	URI des Namensraums, in dem sich das Attribut befindet	voller Name
[**5.4 Namensraumknoten**]	Namensraum-Präfix	-	Namensraum-Präfix
[**5.5 Processing-Instruction-Knoten**]	Ziel	-	Ziel
[**5.6 Kommentarknoten**]	-	-	-
[**5.7 Textknoten**]	-	-	-

Diese Funktionen erwarten als Argument zwar eine Knotenmenge, werten jedoch immer nur den ersten Knoten bezüglich der Dokumentordnung aus. Der zugrunde liegende Begriff erweiterter Name wird in Kapitel [**5 Datenmodell**] erläutert.

Über den Umweg, den Namen eines Knotens auszuwerten, lassen sich Knotentests variabel beschreiben. Möchte man beispielsweise den Typ eines Nachkommenelements parametrisieren, kann man nicht `descendant::$name` verwenden. Mittels der Funktionen **local-name** und **namespace-uri** lässt sich das Problem lösen, indem zunächst alle Nachkommenelemente in die Ausgangsknotenmenge aufgenommen und anschließend innerhalb eines Prädikats diejenigen mit dem richtigen Namen und dem richtigen

Namensraum herausgefiltert werden:

```
descendant::*[local-name()=$lname and namespace-uri()=$uri]
```

Für XML-Dokumente, die keinen Gebrauch von Namensräumen machen, reicht bereits ein Vergleich mit der von der Funktion **name** gelieferten Zeichenkette.

4.2 Zeichenkettenfunktionen

Funktion: *string* **string**(*object*?)

Die Funktion **string** konvertiert ein Objekt wie folgt in eine Zeichenkette:

- Eine Knotenmenge wird in eine Zeichenkette konvertiert, indem der Zeichenkettenwert des ersten Knotens in Dokumentordnung zurückgegeben wird. Falls die Knotenmenge leer ist, wird eine leere Zeichenkette zurückgegeben.

Anmerkung des Übersetzers:

Der Zeichenkettenwert für jeden Knotentyp wird in Kapitel [**5 Datenmodell**] definiert. An dieser Stelle wird bereits zusammengefasst, welchen Wert die Funktion **string** abhängig vom jeweiligen Knotentyp zurückgibt.

	string
[**5.1 Wurzelknoten**]	Zeichenkettenwert des Dokumentelements (des einzigen Kindes des Wurzelknotens)
[**5.2 Elementknoten**]	Verkettung der Zeichenkettenwerte aller Element- und Text-Kindknoten
[**5.3 Attributknoten**]	normalisierter Attributwert
[**5.4 Namensraumknoten**]	URI, d.h. Name des Namensraumes
[**5.5 Processing-Instruction-Knoten**]	Inhalt, der dem Ziel der Processing Instruction folgt
[**5.6 Kommentarknoten**]	Kommentarinhalt
[**5.7 Textknoten**]	enthaltene Zeichendaten

- Eine Zahl wird wie folgt in eine Zeichenkette konvertiert:
  - NaN (Not a Number – keine gültige Zahl) wird in die Zeichenkette `NaN` konvertiert.
  - Positiv Null wird in die Zeichenkette `0` konvertiert.
  - Negativ Null wird in die Zeichenkette `0` konvertiert.
  - Positiv unendlich wird in die Zeichenkette `Infinity` konvertiert.
  - Negativ unendlich wird in die Zeichenkette `-Infinity` konvertiert.
  - Falls die Zahl ein Integer ist, wird sie in dezimaler Form als Number dargestellt, ohne Dezimalpunkt und führende Nullen, mit negativem Vorzeichen (`-`), falls die

Zahl negativ ist.

- Ansonsten wird die Zahl in Dezimalform als Number dargestellt, einschließlich Dezimalpunkt, wenigstens einer Ziffer vor und nach dem Dezimalpunkt sowie einem negativen Vorzeichen (`-`), falls die Zahl negativ ist. Es dürfen keine führenden Nullen vor dem Dezimalpunkt auftreten mit Ausnahme der eventuell erforderlichen Ziffer direkt vor dem Dezimalpunkt. Abgesehen von der einen erforderlichen Ziffer nach dem Dezimalpunkt müssen dort so viele, aber nicht mehr, Ziffern auftreten, wie zur eindeutigen Unterscheidung von allen anderen IEEE 754 numerischen Werten notwendig sind.

- Der boolesche Wert *falsch* wird in die Zeichenkette `false` konvertiert. Der boolesche Wert *wahr* wird in die Zeichenkette `true` konvertiert.

- Die Konvertierung eines Objekts von einem anderen Typ als den vier Grundtypen hängt von diesem Typ ab.

Wird kein Argument übergeben, wird stattdessen eine Knotenmenge mit dem Kontextknoten als einzigem Element benutzt.

> **Anmerkung:** Die Funktion `string` ist nicht dafür gedacht, Zahlen in Zeichenketten zu konvertieren, die an Nutzer ausgegeben werden. Die in [XSLT] definierte Funktion `format-number` und das ebenfalls dort definierte Element `xsl:number` stellen diese Funktionalität bereit.

Funktion: *string* **concat**(*string*, *string*, *string**)

Die Funktion **concat** liefert die Verkettung ihrer Argumente.

Funktion: *boolean* **starts-with**(*string*, *string*)

Die Funktion **starts-with** liefert den logischen Wert *wahr*, falls die im ersten Argument übergebene Zeichenkette mit der im zweiten Argument übergebenen Zeichenkette beginnt, und andernfalls *falsch*.

Anmerkung des Übersetzers:

Gemäß Errata-Dokument [XPath Errata] muss an dieser Stelle folgender Satz ergänzt werden:

> Wenn das zweite Argument die leere Zeichenkette ist, wird der Wert *wahr* zurückgegeben.

Funktion: *boolean* **contains**(*string*, *string*)

Die Funktion **contains** liefert den logischen Wert *wahr*, falls die im ersten Argument übergebene Zeichenkette die im zweiten Argument übergebene Zeichenkette enthält, und andernfalls *falsch*.

Anmerkung des Übersetzers:

Gemäß Errata-Dokument [XPath Errata] muss an dieser Stelle folgender Satz ergänzt werden:

> Wenn das zweite Argument die leere Zeichenkette ist, wird der Wert *wahr* zurückgegeben.

Funktion: *string* **substring-before**(*string*, *string*)

Die Funktion **substring-before** liefert aus der im ersten Argument übergebenen Zeichenkette die Teilzeichenkette, die vor dem ersten Auftreten der im zweiten Argument übergebenen Zeichenkette steht, bzw. die leere Zeichenkette, falls die erste Zeichenkette nicht die zweite enthält. Zum Beispiel liefert `substring-before("1999/04/01","/")` das Ergebnis `1999`.

Anmerkung des Übersetzers:

Gemäß Errata-Dokument [XPath Errata] muss an dieser Stelle folgender Satz ergänzt werden:

> Wenn das zweite Argument die leere Zeichenkette ist, wird die leere Zeichenkette zurückgegeben.

Funktion: *string* **substring-after**(*string*, *string*)

Die Funktion **substring-after** liefert aus der im ersten Argument übergebenen Zeichenkette die Teilzeichenkette, die nach dem ersten Auftreten der im zweiten Argument übergebenen Zeichenkette steht, bzw. die leere Zeichenkette, falls die erste Zeichenkette nicht die zweite enthält. Zum Beispiel liefert `substring-after("1999/04/01","/")` das Ergebnis `04/01` und `substring-after("1999/04/01","19")` liefert `99/04/01`.

Anmerkung des Übersetzers:

Gemäß Errata-Dokument [XPath Errata] muss an dieser Stelle folgender Satz ergänzt werden:

> Wenn das zweite Argument die leere Zeichenkette ist, wird die im ersten Argument übergebene Zeichenkette zurückgegeben.

Funktion: *string* **substring**(*string*, *number*, *number*?)

Die Funktion **substring** liefert aus der im ersten Argument übergebenen Zeichenkette die Teilzeichenkette, die an der im zweiten Argument angegebenen Position beginnt und die im dritten Argument angegebene Länge besitzt. Zum Beispiel liefert `substring("12345",2,3)` das Ergebnis `"234"`. Falls kein drittes Argument angegeben wird, liefert die Funktion die Teilzeichenkette, die an der im zweiten Argument angegebenen Position beginnt und bis zum

Ende der Zeichenkette reicht. Zum Beispiel liefert `substring("12345",2)` das Ergebnis `"2345"`.

Genauer gesagt wird für jedes Zeichen der Zeichenkette (siehe [3.6 Zeichenketten]) eine numerische Position angenommen: die Position des ersten Zeichens ist 1, die Position des zweiten Zeichens ist 2 usw.

> **Anmerkung:** Dies unterscheidet sich von Java und ECMAScript, in denen die Methode `String.substring` die Position des ersten Zeichens mit 0 definiert.

Die zurückgegebene Teilzeichenkette enthält die Zeichen, deren Position größer oder gleich dem gerundeten Wert des zweiten Arguments ist und, falls ein drittes Argument übergeben wurde, kleiner als die Summe des gerundeten Wertes des zweiten und des gerundeten Wertes des dritten Arguments. Vergleich und Addition folgen den Standardregeln von IEEE 754; die Rundung erfolgt wie durch die Funktion **round**. Die folgenden Beispiele illustrieren einige unübliche Fälle:

- `substring("12345", 1.5, 2.6)` liefert `"234"`
- `substring("12345", 0, 3)` liefert `"12"`
- `substring("12345", 0 div 0, 3)` liefert `""`
- `substring("12345", 1, 0 div 0)` liefert `""`
- `substring("12345", -42, 1 div 0)` liefert `"12345"`
- `substring("12345", -1 div 0, 1 div 0)` liefert `""`

Anmerkung des Übersetzers:

Die letzten vier Beispiele werden plausibel, wenn man die Definition für **substring** wortgetreu anwendet und beachtet, dass ein Vergleich mit dem Wert NaN immer *falsch* sowie jede Gleitkommazahl kleiner als positiv unendlich ist.

Mit Hilfe eines ähnlich gearteten Aufrufs dieser Funktion können bedingte Ausdrücke für Zeichenketten und damit auch Zahlen simuliert werden. Ein bedingter Ausdruck liefert abhängig von der Auswertung eines logischen Ausdrucks einen von zwei möglichen vorgegebenen Werten. Für Knotenmengen wurde die analoge Funktionalität im Zusammenhang mit dem Vereinigungsoperator | bereits in Kapitel [3.3 Knotenmengen] vorgestellt.

Unter Ausnutzung der Tatsache, dass ein logischer Wert *wahr* nach 1 und ein logischer Wert *falsch* nach 0 konvertiert wird (siehe Regeln bei der Funktion **number**), ergibt der Ausdruck

```
substring(string, 1 div boolean-test)
```

den Wert *`string`*, falls *`boolean-test`* *wahr* ist, und ansonsten die leere Zeichenkette. Die vollständige Formulierung eines bedingten Ausdrucks sieht damit folgendermaßen aus:

```
concat(substring(true-string, 1 div boolean-test),
       substring(false-string, 1 div not(boolean-test)))
```

Solche Ausdrücke können beim Sortieren in XSLT eingesetzt werden, wenn der zu benutzende Sortierschlüssel von einer aktuell auszuwertenden Bedingung abhängt. Allerdings werden sie bei komplexeren Bedingungen schnell unübersichtlich.

Funktion: *number* **string-length**(*string*?)

Die Funktion **string-length** liefert die Anzahl der Zeichen der Zeichenkette (siehe [**3.6 Zeichenketten**]). Falls kein Argument übergeben wurde, wird der in eine Zeichenkette konvertierte Kontextknoten angenommen, mit anderen Worten der Zeichenkettenwert des Kontextknotens.

Anmerkung des Übersetzers:

Es gibt in XPath keine Funktion `ends-with`, die analog zu **starts-with** bestimmt, ob eine Zeichenkette mit einer anderen endet. Mit Hilfe der Funktionen **substring** und **string-length** kann diese Funktionalität jedoch nachgebildet werden. `ends-with($str1, $str2)` würde den Wert *wahr* liefern, falls gilt:

```
$str2 = substring($str1, string-length($str1) - string-length($str2) + 1)
```

Für das Auffüllen einer Zeichenkette von links mit Leerzeichen (z.B. für eine rechtsbündige Textausgabe) erweisen sich die Funktionen **concat**, **substring** und **string-length** sowie eine ausreichend lange, ausschließlich aus Leerzeichen bestehende Zeichenkette als gute Helfer:

```
concat(substring("                                                  ",
                 1, $width - string-length($eingabe)), $eingabe)
```

Funktion: *string* **normalize-space**(*string*?)

Die Funktion **normalize-space** liefert als Ergebnis die übergebene Zeichenkette mit normalisiertem Leerraum zurück, d.h. führender und abschließender Leerraum werden entfernt, Folgen von mehreren Leerraumzeichen werden durch ein einzelnes Leerzeichen ersetzt. Leerraumzeichen sind jene, die durch die Produktion S in XML definiert sind. Falls kein Argument übergeben wurde, wird der in eine Zeichenkette konvertierte Kontextknoten angenommen, mit anderen Worten der Zeichenkettenwert des Kontextknotens.

Anmerkung des Übersetzers:

Diese Funktion verarbeitet eine übergebene Zeichenkette in der gleichen Weise, wie ein XML-Prozessor nach Ersetzung aller Referenzen Nicht-CDATA-Attribute behandelt, siehe Kapitel 3.3.3 in [XML, 2nd Edition].

Zur Erläuterung sei das folgende Beispiel angegeben. Häufig finden sich in XML-Dokumenten Textdaten, die folgendermaßen ausgezeichnet sind:

```
<name>
   Otto Normal
</name>
```

Der Zeichenkettenwert des Elements `name` enthält neben der wichtigen Information über `Otto Normal` auch alle Leerraumzeichen, also den Zeilenumbruch nach dem Start-Tag `<name>`, die Leerzeichen vor `Otto` und den Zeilenumbruch inklusive eventueller Leerzeichen vor dem schließenden End-Tag. Ein Vergleich für den Kontextknoten `name`

```
.="Otto Normal"
```

liefert also nicht das erhoffte Ergebnis, sondern schlägt fehl. Korrekt ist in diesem Fall stattdessen ein Vergleich mit dem normalisierten Wert des Knotens:

```
normalize-space()="Otto Normal"
```

Am Beispiel **normalize-space** soll an dieser Stelle vor einer Falle gewarnt werden, in die man bei der Übergabe von Knotenmengen an Funktionen mit skalaren Parametertypen leicht geraten kann. Angenommen, der Name `Otto Normal` kommt mehrfach vor und man möchte nur den ersten dieser `name`-Knoten bearbeiten. In diesem Fall kann man z.B. verlangen, dass keiner der Vorgänger den gleichen Namen hat. Ist eine Normalisierung nicht erforderlich, lautet der Test einfach:

```
.="Otto Normal" and not(preceding-sibling::name="Otto Normal")
```

Muss man aber normalisieren, erweist sich der folgende Test als ungeeignet:

```
normalize-space()="Otto Normal" and
not(normalize-space(preceding-sibling::name)="Otto Normal")
```

Da die Funktion **normalize-space** als Argument eine Zeichenkette erwartet, wird die übergebene Knotenmenge mit der Funktion **string** konvertiert. Es wird also letztlich nur der erste Knoten (bezüglich der Dokumentordnung) der übergebenen Knotenmenge durch **normalize-space** ausgewertet und damit nur getestet, ob der erste `name`-Knoten einen von »`Otto Normal`« verschiedenen normalisierten Zeichenkettenwert besitzt. Die Lösung besteht in solchen Fällen darin, den Test in ein Prädikat zu verschieben:

```
normalize-space()="Otto Normal" and
not(preceding-sibling::name[normalize-space()="Otto Normal"])
```

Funktion: *string* **translate**(*string*, *string*, *string*)

Die Funktion **translate** liefert als Ergebnis die im ersten Argument übergebene Zeichenkette, wobei jedes Vorkommen eines Zeichens aus der im zweiten Argument übergebenen Zeichenkette ersetzt wird durch das Zeichen an der korrespondierenden Position aus der im dritten Argument übergebenen Zeichenkette. Zum Beispiel liefert `translate("bar","abc","ABC")` die Zeichenkette `BAr`. Wenn es im zweiten Argument ein Zeichen gibt, für das kein korrespondierendes Zeichen im dritten Argument existiert (weil das zweite Argument länger ist als das dritte), so werden alle Vorkommen dieses Zeichens im ersten Argument entfernt. Zum Beispiel liefert `translate("--aaa--","abc-","ABC")` das Ergebnis `"AAA"`. Falls ein Zeichen mehrmals im zweiten Argument vorkommt, bestimmt das erste Auftreten das Ersetzungszeichen. Falls die im dritten Argument übergebene Zeichenkette länger ist als die zweite, werden überzählige Zeichen ignoriert.

> **Anmerkung:** Die Funktion **translate** ist keine ausreichende Lösung für die Umwandlung zwischen Groß- und Kleinschreibung in allen Sprachen. Eine zukünftige Version von XPath kann zusätzliche Funktionen für diese Umwandlung

zur Verfügung stellen.

Anmerkung des Übersetzers:

Für deutsche Umlaute reicht sie im Allgemeinen aus. Möchte man in einem Stylesheet an mehreren Stellen Klein- in Großschreibung umwandeln, bietet es sich an, geeignete Variablen zu definieren und fortan diese zu benutzen.

In XSLT sähe das folgendermaßen aus:

```
<xsl:variable name="upper-case"
              select="'ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ'" />
<xsl:variable name="lower-case"
              select="'abcdefghijklmnopqrstuvwxyzäöü'" />

...
<xsl:if test="translate(firma, $lower-case, $upper-case) = 'XYZ GMBH'">
   ...
</xsl:if>
```

Allerdings stößt man schon beim Buchstaben »ß« an die Grenzen der Funktion **translate**. Die Ersetzung von »ß« durch die beiden Buchstaben »SS« ist auf diese Weise nicht möglich, weil der Ersetzungstext länger als ein Zeichen ist.

Allgemein gilt, dass die Funktion **translate** kein Mittel zum Ersetzen von beliebigen Teilzeichenketten ist. Ein einmaliges Ersetzen im Sinne von `replace($string, $from, $to)` kann durch

```
concat(substring-before($string, $from), $to,
       substring-after($string, $from))
```

ausgedrückt werden. XPath 1.0 stellt keine Mittel bereit, alle Vorkommen von `$from` in `$string` durch `$to` zu ersetzen. Dies wird sich in einer zukünftigen XPath-Version ändern [XPath Operators 2.0].

Mit der Funktion **translate** lassen sich darüber hinaus sehr einfach ausgewählte Zeichen aus einer Zeichenkette entfernen, indem als drittes Argument eine leere Zeichenkette übergeben wird. Dies kann beispielsweise für den Test genutzt werden, ob eine Zeichenkette `$string` nur die in `$allowed-char` aufgezählten Zeichen enthält:

```
translate($string, $allowed-char, "") = ""
```

4.3 Boolesche Funktionen

Funktion: *boolean* **boolean**(*object*)

Die Funktion **boolean** konvertiert ihr Argument wie folgt in einen Boolean-Wert:

- Eine Zahl ergibt den Wert *wahr* genau dann, wenn sie weder positiv oder negativ Null noch NaN ist.
- Eine Knotenmenge ergibt den Wert *wahr* genau dann, wenn sie nicht leer ist.

> **Anmerkung des Übersetzers:**
>
> Aus diesem Grund lässt sich ein Test auf das Vorhandensein von Knoten sehr kompakt aufschreiben: `img[@alt]` bedeutet dasselbe wie `img[boolean(@alt)]`. Gesucht ist damit ein `img`-Element, das ein `alt`-Attribut besitzt.

- Eine Zeichenkette ergibt genau dann den Wert *wahr*, wenn ihre Länge ungleich Null ist.

> **Anmerkung des Übersetzers:**
>
> Zum Vergleich mit dem letzten Beispiel: `img[string(@alt)]` steht abkürzend für `img[boolean(string(@alt))]` und bedeutet jetzt, dass ein `img`-Element gesucht wird, dessen `alt`-Attribut einen nichtleeren Zeichenkettenwert besitzt. Für `<img alt="" ... />` wäre der erste Test erfüllt, der zweite jedoch nicht.

- Die Konvertierung eines Objekts von einem anderen Typ als den vier Grundtypen hängt von diesem Typ ab.

> **Anmerkung des Übersetzers:**
>
> Während das Argument der Funktionen **string** und **number** optional ist, beim Fehlen damit der Kontextknoten angenommen wird, ist das Argument für die Funktion **boolean** obligatorisch. Eine analoge Definition, die gegebenenfalls auf den Kontextknoten zurückgreift, hätte auch nicht viel Sinn, da in diesem Fall die Funktion **boolean** den Wert *wahr* liefern müsste – der Kontextknoten ist schließlich immer vorhanden.

Funktion: *boolean* **not**(*boolean*)

Die Funktion **not** liefert den Wert *wahr*, wenn ihr Argument *falsch* ist, und ansonsten *falsch*.

> **Anmerkung des Übersetzers:**
>
> Im Unterschied zu den Operatoren `and` und `or`, die in den Produktionen OrExpr bzw. AndExpr definiert werden, handelt es sich bei **not** um eine Funktion.

Funktion: *boolean* **true**()

Die Funktion **true** liefert den Wert *wahr*.

Funktion: *boolean* **false**()

Die Funktion **false** liefert den Wert *falsch*.

Anmerkung des Übersetzers:

XPath definiert keine Literale für *wahr* und *falsch*. Soll ein boolescher Wert als Parameter an eine Funktion oder ein benanntes XSLT-Template übergeben werden, muss dafür eine der Funktionen **true** oder **false** benutzt werden.

Funktion: *boolean* **lang**(*string*)

Die Funktion **lang** liefert einen Wert *wahr* oder *falsch* in Abhängigkeit davon, ob die durch `xml:lang`-Attribute angegebene Sprache des Kontextknotens die gleiche oder eine Untersprache der im Argument übergebenen Zeichenkette ist. Die Sprache des Kontextknotens wird durch den Wert des Attributes `xml:lang` des Kontextknotens bestimmt oder, wenn der Kontextknoten kein Attribut `xml:lang` besitzt, durch den Wert des Attributes `xml:lang` beim nächsten Vorfahren des Kontextknotens, der ein Attribut `xml:lang` besitzt. Wenn es kein solches Attribut gibt, liefert **lang** den Wert *falsch*. Wenn es ein solches Attribut gibt, liefert **lang** den Wert *wahr*, wenn der Attributwert gleich dem Argument ist, unabhängig von der Groß- oder Kleinschreibung, oder wenn es ein mit `-` beginnendes Suffix derart gibt, dass der Attributwert gleich dem Argument ohne dieses Suffix ist, unabhängig von der Groß- oder Kleinschreibung. Beispielsweise würde `lang("en")` den Wert *wahr* liefern, wenn es sich beim Kontextknoten um eines dieser fünf Elemente handelt:

```
<para xml:lang="en"/>
<div xml:lang="en"><para/></div>
<para xml:lang="EN"/>
<para xml:lang="en-us"/>
```

Anmerkung des Übersetzers:

Handelt es sich beim Kontextknoten dagegen um das `para`-Element im Beispiel

```
<div xml:lang="en"><sect xml:lang="de"><para/></sect></div>
```

liefert `lang("en")` den Wert *falsch*. Der nächste Vorfahre mit einem `xml:lang`-Attribut ist in diesem Fall das Element `sect`, in welchem die Sprache auf `de` gesetzt wird.

Das Verhalten der Funktion **lang** darf nicht mit einem impliziten Vorhandensein des Attributs `xml:lang` verwechselt werden. Für das obige Beispiel liefert `//para[@xml:lang='de']` eine leere Knotenmenge, während `//para[lang('de')]` das `para`-Element auswählt.

4.4 Zahlenfunktionen

Funktion: *number* **number**(*object*?)

Die Funktion **number** konvertiert ihr Argument wie folgt in eine Zahl:

- Eine Zeichenkette, die aus optionalem Leerraum besteht, gefolgt von einem optionalen Minuszeichen, gefolgt von einer Number, gefolgt von Leerraum, wird in eine IEEE-754-

Zahl konvertiert, die (entsprechend den Rundungsregeln in IEEE 754) dem mathematischen Wert am nächsten ist, der durch die Zeichenkette repräsentiert wird. Jede andere Zeichenkette wird in NaN konvertiert.

- Der boolesche Wert *wahr* wird in 1 konvertiert; der boolesche Wert *falsch* wird in 0 konvertiert.
- Eine Knotenmenge wird zunächst wie beim Aufruf der Funktion **string** in eine Zeichenkette konvertiert und anschließend in der gleichen Weise wie eine Zeichenkette konvertiert.
- Die Konvertierung eines Objekts von einem anderen Typ als den vier Grundtypen hängt von diesem Typ ab.

Wird kein Argument übergeben, wird stattdessen eine Knotenmenge mit dem Kontextknoten als einzigem Element benutzt.

Anmerkung: Die Funktion **number** sollte nicht für die Konvertierung numerischer Daten in einem Element eines XML-Dokuments benutzt werden, es sei denn, das Element ist von einem Typ, der numerische Daten in einem sprachunabhängigen Format repräsentiert (das typischerweise für die Präsentation für einen Anwender in ein sprachspezifisches Format umgewandelt würde). Darüber hinaus kann die Funktion **number** nur genutzt werden, wenn das von dem Element genutzte sprachunabhängige Format mit der XPath-Syntax für Number konsistent ist.

Anmerkung des Übersetzers:

Insbesondere eignet sich eine Zahl im hiesigen Format (der Punkt zur Kennzeichnung der Tausenderstellen, das Komma zur Abgrenzung von den Dezimalstellen) nicht als Argument für die Funktion **number**. Der Aufruf von `number('12.000,50')` liefert beispielsweise den Wert NaN. Eine Umwandlung in das Format für Number kann in diesem Fall mit Hilfe der Funktion **translate** vorgenommen werden: `number(translate('12.000,50', ',.', '.'))`.

Funktion: *number* **sum**(*node-set*)

Die Funktion **sum** liefert die Summe aller in eine Zahl konvertierten Zeichenkettenwerte der Knoten aus der Argumentknotenmenge.

Anmerkung des Übersetzers:

Sie eignet sich damit nur für Fälle, in denen die zu summierenden Werte direkt in der XML-Quelle vorliegen. Sollen die Summanden komplexer sein, d.h. jeweils durch einen eigenen Ausdruck berechnet werden, kann die Funktion **sum** nicht mehr benutzt werden. XPath nutzende Implementationen können aber eigene Sprachmittel bereitstellen, mit denen solche Berechnungen durchgeführt werden können.

Die Funktion **sum** hilft noch in einem anderen Anwendungsfall: Aus der Definition des Verhaltens von **number** geht hervor, dass letztere eine leere Knotenmenge in den Wert NaN konvertiert. Ein Ausdruck `foo - bar` liefert damit NaN, falls wenigstens eines der Elemente `foo` oder `bar` nicht existiert. Möchte man stattdessen, dass in diesem Fall die

Zahl 0 für nicht existierende Elemente angenommen wird, kann man dies über den Ausdruck `sum(foo[1]) - sum(bar[1])` erreichen.

Funktion: *number* **floor**(*number*)

Die Funktion **floor** liefert die größte Zahl (die am nächsten an positiv unendlich liegt), die nicht größer als das Argument und ganzzahlig ist.

Anmerkung des Übersetzers:

Gemäß Errata-Dokument [XPath Errata] muss an dieser Stelle folgender Absatz ergänzt werden:

> Wenn das Argument NaN ist, wird NaN zurückgegeben. Wenn das Argument positiv unendlich ist, wird positiv unendlich zurückgegeben. Wenn das Argument negativ unendlich ist, wird negativ unendlich zurückgegeben. Wenn das Argument positiv Null ist, wird positiv Null zurückgegeben. Wenn das Argument negativ Null ist, wird negativ Null zurückgegeben. Wenn das Argument größer als Null, aber kleiner als 1 ist, wird positiv Null zurückgegeben.

Funktion: *number* **ceiling**(*number*)

Die Funktion **ceiling** liefert die kleinste Zahl (die am nächsten an negativ unendlich liegt), die nicht kleiner als das Argument und ganzzahlig ist.

Anmerkung des Übersetzers:

Gemäß Errata-Dokument [XPath Errata] muss an dieser Stelle folgender Absatz ergänzt werden:

> Wenn das Argument NaN ist, wird NaN zurückgegeben. Wenn das Argument positiv unendlich ist, wird positiv unendlich zurückgegeben. Wenn das Argument negativ unendlich ist, wird negativ unendlich zurückgegeben. Wenn das Argument positiv Null ist, wird positiv Null zurückgegeben. Wenn das Argument negativ Null ist, wird negativ Null zurückgegeben. Wenn das Argument kleiner als Null, aber größer als -1 ist, wird positiv Null zurückgegeben.

Funktion: *number* **round**(*number*)

Die Funktion **round** liefert die Zahl, die am nächsten am Argument liegt und die ganzzahlig ist. Wenn es zwei solche Zahlen gibt, wird die Zahl geliefert, die näher an positiv unendlich liegt. Ist das Argument NaN, wird NaN zurückgegeben. Ist das Argument positiv unendlich, wird positiv unendlich zurückgegeben. Ist das Argument negativ unendlich, wird negativ unendlich zurückgegeben. Ist das Argument positiv Null, wird positiv Null zurückgegeben. Ist das Argument negativ Null, wird negativ Null zurückgegeben. Ist das Argument kleiner als Null, aber größer oder gleich -0.5, wird negativ Null zurückgegeben.

Anmerkung: In den letzten beiden Fällen ist das Ergebnis des Aufrufs der Funktion **round** verschieden von der Addition von 0.5 und dem Aufruf der Funktion **floor**.

Anmerkung des Übersetzers:

Davon kann man sich leicht selbst überzeugen:

Für -0 ergibt sich: `floor(-0 + 0.5) = floor(0.5) = 0` (positiv Null)

Für -0.5 ergibt sich: `floor(-0.5 + 0.5) = floor(0) = 0` (positiv Null)

5 Datenmodell

XPath operiert auf der Baumrepräsentation eines XML-Dokuments. Das folgende Kapitel beschreibt, wie XPath ein XML-Dokument als Baum modelliert. Dieses Modell ist rein konzeptionell und schreibt keinerlei spezielle Implementation vor. Die Beziehung zwischen diesem Modell und der XML-Informationsmenge [XML Infoset] wird in [**B Abbildung auf die XML-Informationsmenge**] beschrieben.

Die durch XPath verarbeiteten XML-Dokumente müssen sich nach der XML-Namensraum-Empfehlung [XML Names] richten.

Anmerkung des Übersetzers:

Insbesondere dürfen Namen von Elementen und Attributen das Zeichen »`:`« nur als Separator zwischen Namensraum-Präfix und lokalem Namen enthalten. Außerdem muss jedes benutzte Präfix deklariert worden sein. XML-Dokumente, die den Doppelpunkt nicht Namensraum-konform verwenden, können durch XPath-Implementationen nicht verarbeitet werden.

Der Baum enthält Knoten. Es gibt sieben Typen von Knoten:

- Wurzelknoten
- Elementknoten
- Textknoten
- Attributknoten
- Namensraumknoten
- Processing-Instruction-Knoten
- Kommentarknoten

Für jeden Knotentyp gibt es einen Weg, den **Zeichenkettenwert** eines Knotens dieses Typs zu

bestimmen. Bei einigen Knotentypen ist der Zeichenkettenwert Teil des Knotens, bei anderen Knotentypen wird der Zeichenkettenwert aus den Zeichenkettenwerten der Nachkommen berechnet.

> **Anmerkung:** Für Element- und Wurzelknoten ist der Zeichenkettenwert eines Knotens verschieden von der Zeichenkette, die von der DOM-Methode `nodeValue` zurückgegeben wird (siehe [DOM]).

Einige Knotentypen besitzen außerdem einen **erweiterten Namen** – ein Paar bestehend aus einem lokalen Teil und einem Namensraum-URI. Der lokale Teil ist eine Zeichenkette. Der Namensraum-URI ist entweder leer oder eine Zeichenkette. Der im XML-Dokument spezifizierte Namensraum-URI kann eine URI-Referenz sein, wie sie in [RFC2396] definiert ist; das bedeutet, sie kann einen Fragment-Bezeichner besitzen und sie kann relativ sein. Ein relativer URI sollte als absoluter URI während der Namensraum-Verarbeitung aufgelöst werden: Die Namensraum-URIs der erweiterten Namen von Knoten im Datenmodell sollten absolut sein. Zwei erweiterte Namen sind gleich, wenn sie den gleichen lokalen Teil haben und wenn beide einen leeren Namensraum-URI oder beide die gleichen, nichtleeren Namensraum-URIs besitzen.

Anmerkung des Übersetzers:

Nach einer Entscheidung des W3C-XML-Plenums ist die Behandlung relativer Namensraum-URIs implementationsabhängig. Gemäß Errata-Dokument [XPath Errata] müssen deshalb aus dem obigen Abschnitt die Sätze *"Der im XML-Dokument spezifizierte Namensraum-URI kann eine URI-Referenz sein, wie sie in [RFC2396] definiert ist; das bedeutet, sie kann einen Fragment-Bezeichner besitzen und sie kann relativ sein. Ein relativer URI sollte als absoluter URI während der Namensraum-Verarbeitung aufgelöst werden: Die Namensraum-URIs der erweiterten Namen von Knoten im Datenmodell sollten absolut sein."* ersetzt werden durch:

> Ein in einer Namensraumdeklaration spezifizierter Namensraum-Name in einem XML-Dokument ist eine URI-Referenz, wie sie in [RFC2396] definiert ist; das bedeutet, sie kann einen Fragment-Bezeichner besitzen und sie kann relativ sein. Die Namensraum-URI-Komponente eines erweiterten Namens ist implementationsabhängig, wenn der erweiterte Name aus einem QName expandiert wird, dessen Präfix durch eine Namensraumdeklaration mit einem relativen URI (mit oder ohne Fragment-Bezeichner) als Namensraum-Namen deklariert wurde. Ein XPath-Ausdruck, der vom Wert der Namensraum-URI-Komponente eines solchen erweiterten Namens abhängt, ist nicht interoperabel.

Relative URIs der Form »`host.example.org/namespace`« (fehlendes Schema), »`home/schema`« (fehlende Host-Angabe), »`http://www.w3.org/1999/../2001/XMLSchema`« (mit speziellen Pfadkomponenten) oder »`http://www.schema.org/address#one`« (mit Fragment-Bezeichner) sollten damit vermieden werden.

Es existiert eine Ordnung, die **Dokumentordnung**, die für alle Knoten im Dokument definiert ist und die zu der Ordnung korrespondiert, in der die ersten Zeichen der XML-Repräsentation aller Knoten in der XML-Repräsentation des Dokuments nach der Expandierung allgemeiner Entities stehen. Der Wurzelknoten ist demzufolge der erste Knoten. Elementknoten stehen vor ihren Kindern. Das bedeutet, die Dokumentordnung ordnet Elementknoten in der Reihenfolge ihrer Start-Tags im XML-Dokument (nach der Expandierung von Entities). Attribut- und Namensraumknoten eines Elements kommen vor den Kindern des Elements.

Namensraumknoten erscheinen per Definition vor den Attributknoten. Die relative Ordnung innerhalb der Namensraumknoten ist implementationsabhängig. Die relative Ordnung innerhalb der Attributknoten ist implementationsabhängig. Die **umgekehrte Dokumentordnung** ist die Umkehrung der Dokumentordnung.

Anmerkung des Übersetzers:

Damit ist für Attribut- und Namensraumknoten die Reihenfolge der Repräsentation im XML-Dokument unerheblich.

Man darf an dieser Stelle nicht vergessen, dass die Elemente einer Knotenmenge trotzdem immer ungeordnet sind. Die Eigenschaft, die Knoten einer Menge in einer bestimmten Reihenfolge zu betrachten, gehört immer zu einer Operation oder Funktion, die auf dieser Menge ausgeführt wird. So konvertieren die Funktionen **string**, **boolean** und **number** bei einer gegebenen Knotenmenge immer den ersten Knoten bezüglich der Dokumentordnung. Entsprechend legen auf Ausdrücke angewandte Prädikate immer die Dokumentordnung zugrunde. Demgegenüber legt innerhalb eines Lokalisierungsschrittes immer die jeweilige Achse die für die dazugehörenden Prädikate relevante Ordnung fest. Nachdem eine Knotenmenge konstruiert wurde, ist sie (wieder) ungeordnet. Siehe dazu auch die entsprechende Anmerkung in Kapitel [**3.3 Knotenmengen**].

Wurzel- und Elementknoten besitzen eine geordnete Liste von Kindknoten. Mehrere Knoten haben niemals gemeinsame Kinder: Wenn ein Knoten von einem anderen Knoten verschieden ist, dann ist kein Kindknoten des einen Knotens mit einem der Kindknoten des anderen Knotens identisch. Jeder Knoten mit Ausnahme des Wurzelknotens besitzt genau einen **Elternknoten**, wobei dieser entweder ein Elementknoten oder der Wurzelknoten ist. Ein Wurzel- oder ein Elementknoten ist der Elternknoten jedes seiner Kindknoten. Die **Nachkommen** eines Knotens sind die Kinder des Knotens sowie die Nachkommen der Kinder des Knotens.

Anmerkung des Übersetzers:

Die Bezeichnungen *Eltern*- und *Kind*knoten sind vielleicht nicht ganz glücklich. Jeder Knoten besitzt nämlich maximal *einen* Elternknoten. Daneben sind Attribut- und Namensraumknoten vergleichbar mit rebellierenden Teenagern, die im XPath-Datenmodell nicht als Kinder ihrer Element-Elternknoten betrachtet werden.

5.1 Wurzelknoten

Der Wurzelknoten ist die Wurzel des Baumes. Ein Wurzelknoten kommt nur als Wurzel des Baumes vor. Der Elementknoten für das Dokumentelement ist ein Kind des Wurzelknotens. Der Wurzelknoten hat als Kinder ebenfalls Processing-Instruction-Knoten und Kommentarknoten für Processing Instructions und Kommentare, die im Prolog oder hinter dem Ende des Dokumentelements auftreten.

Anmerkung des Übersetzers:

Die Dokumenttyp-Deklaration ist kein Kind des Wurzelknotens. Sie kommt als solche an keiner Stelle im XPath-Datenmodell vor. So kann ein XSLT-Stylesheet keine Kopie der Dokumenttyp-Deklaration erzeugen. Ebenso werden innerhalb der Dokumenttyp-Deklaration auftretende Kommentare oder Processing Instructions nicht durch XPath-Knoten repräsentiert.

Der Zeichenkettenwert des Wurzelknotens ergibt sich aus der Verkettung der Zeichenkettenwerte aller Textknoten in Dokumentordnung, die Nachkommen des Wurzelknotens sind.

Anmerkung des Übersetzers:

Da außerhalb des Dokumentelements keine Textknoten auftreten dürfen, gilt also

```
string(/) = string(/*)
```

oder mit anderen Worten: Der Wurzelknoten und sein (einziger) Element-Kindknoten besitzen den gleichen Zeichenkettenwert. Diesen erhält man auch, wenn aus dem Eingabedokument jegliches Markup entfernt und nur die Zeichendaten behalten werden.

Zum Vergleich: Im Document Object Model [DOM] ist der Wert des vergleichbaren Attributs `nodeValue` für ein `Document`-Objekt dagegen die leere Zeichenkette.

Der Wurzelknoten hat keinen erweiterten Namen.

Anmerkung des Übersetzers:

Das bedeutet, dass die Funktionen **name**, **local-name** und **namespace-uri** als Ergebnis die leere Zeichenkette liefern. Das in DOM definierte Attribut `nodeName` besitzt dagegen für das `Document`-Objekt als Wert die Zeichenkette »`#document`«.

5.2 Elementknoten

Für jedes Element im Dokument gibt es einen Elementknoten. Ein Elementknoten besitzt einen erweiterten Namen, der sich durch Expandierung des im Tag des Elements spezifizierten QName in Übereinstimmung mit der XML-Namensraum-Empfehlung [XML Names] ergibt. Der Namensraum-URI des erweiterten Namens des Elements ist leer, wenn der QName kein Präfix enthält und es keinen anwendbaren voreingestellten Namensraum gibt.

> **Anmerkung:** In der im Anhang A.3 von [XML Names] verwendeten Notation entspricht der lokale Teil des erweiterten Namens dem Attribut `type` des Elements `ExpEType`; der Namensraum-URI des erweiterten Namens entspricht dem Attribut `ns` des Elements `ExpEType` und ist leer, wenn das Attribut `ns` des Elements `ExpEType` weggelassen wurde.

Die Kinder eines Elementknotens sind die enthaltenen Elementknoten, Kommentarknoten, Processing-Instruction-Knoten und Textknoten. Entity-Referenzen zu internen und externen Entities werden expandiert. Zeichenreferenzen werden aufgelöst.

Der Zeichenkettenwert eines Elementknotens ergibt sich aus der Verkettung der Zeichenkettenwerte aller Textknoten, die Nachkommen des Elementknotens in Dokumentordnung sind.

Anmerkung des Übersetzers:

Damit gilt, dass der Zeichenkettenwert eines Elementknotens genauso als Verkettung der Zeichenkettenwerte seiner Element- und Textkinder in Dokumentordnung aufgefasst werden kann.

Zum Vergleich: Im Document Object Model [DOM] ist dagegen der Wert des vergleichbaren Attributs `nodeValue` für ein `Element`-Objekt die leere Zeichenkette.

5.2.1 Eindeutige IDs

Ein Elementknoten kann einen eindeutigen Bezeichner (ID) besitzen. Dies ist der Wert des Attributes, das in der DTD mit dem Typ `ID` deklariert wurde. Keine zwei Elemente in einem Dokument dürfen den gleichen eindeutigen Bezeichner besitzen. Falls ein XML-Prozessor zwei Elemente in einem Dokument mit dem gleichen eindeutigen Bezeichner meldet (was nur möglich ist, wenn das Dokument ungültig ist), dann muss das zweite Element in Dokumentordnung so behandelt werden, als habe es keinen eindeutigen Bezeichner.

> **Anmerkung:** Wenn ein Dokument keine DTD besitzt, dann hat kein Element des Dokuments einen eindeutigen Bezeichner.

Anmerkung des Übersetzers:

Für zwei oder mehr Elemente mit dem gleichen eindeutigen Bezeichner liefert die Funktion **id** nur den ersten Elementknoten in Dokumentordnung zurück. Da die Eigenschaft eines Attributes, eindeutiger Bezeichner zu sein, in der DTD deklariert wird, kann ohne DTD dieses Attribut nicht mehr erkannt werden. Das bedeutet, dass die Funktion **id** nach dem Entfernen der DTD aus dem Eingabedokument für jedes Argument nur noch die leere Knotenmenge liefert. Dies ist einer der wenigen Fälle, in denen die Existenz einer DTD sich auf die Auswertung eines XPath-Ausdrucks auswirkt.

5.3 Attributknoten

Jedes Element besitzt eine mit ihm verbundene Menge von Attributknoten; das Element ist der Elternknoten jedes dieser Attributknoten. Allerdings ist ein Attributknoten kein Kind seines Elternknotens.

> **Anmerkung:** Dies unterscheidet sich vom DOM, welches ein Element nicht als Elternknoten seiner Attribute behandelt (siehe [DOM]).

Mehrere Elemente haben niemals gemeinsame Attributknoten: Wenn ein Elementknoten verschieden von einem anderen Elementknoten ist, dann ist kein Attributknoten des einen Elementknotens mit einem der Attributknoten eines anderen Elementknotens identisch.

> **Anmerkung:** Der Operator = testet, ob zwei Knoten den gleichen Wert haben, *nicht* ob es dieselben Knoten sind. Vergleicht man die Attribute von zwei verschiedenen Elementen mittels =, so kann sich Gleichheit ergeben, obwohl diese nicht dieselben Knoten sind.

Anmerkung des Übersetzers:

Wert bedeutet hier wieder Zeichenkettenwert. Tatsächlich gilt die obige Anmerkung für alle Knoten, nicht nur für Attribute. Um die Identität zweier Knoten zu überprüfen, kann man sich beispielsweise der in Kapitel [**3.3 Knotenmengen**] im Zusammenhang mit dem Operator | vorgestellten Technik bedienen.

Ein vorgegebenes Attribut wird genauso behandelt wie ein spezifiziertes Attribut. Falls für einen Elementtyp ein Attribut in der DTD deklariert wurde, der Vorgabewert jedoch als `#IMPLIED` deklariert und das Attribut für das Element nicht angegeben wurde, so enthält die Attributmenge des Elements keinen Knoten für dieses Attribut.

Anmerkung des Übersetzers:

Beispiel:

```
<!DOCTYPE see [
<!ELEMENT see EMPTY>
<!ATTLIST see access (public|restricted) "public"
              ref    CDATA               #IMPLIED >
]>
<see />
```

In diesem Fall besitzt der einzige Elementknoten `see` im XPath-Datenmodell genau einen Attributknoten namens `access` mit dem Wert `public`. Im Gegensatz zu `ref` handelt es sich bei `access` um ein vorgegebenes Attribut.

Das XPath-Datenmodell liefert keinerlei Informationen darüber, ob ein solches Attribut im Start-Tag eines Elements spezifiziert wurde oder nicht. Im zweiten Fall ändert sich im Falle des Entfernens der DTD die Baumrepräsentation des XML-Dokuments, da alle vorgegebenen, aber nicht spezifizierten Attribute wegfallen.

Einige Attribute, wie `xml:lang` und `xml:space`, besitzen die Semantik, dass sie für alle Elemente gelten, die Nachkommen des Elements sind, das das Attribut trägt, es sei denn, sie wurden in einer Instanz eines Nachkommenelements durch das gleiche Attribut überschrieben. Das wirkt sich allerdings nicht darauf aus, wo Attributknoten im Baum vorkommen: Ein Element besitzt nur Attributknoten für die Attribute, die explizit im Start-Tag oder Leeres-Element-Tag dieses Elements angegeben wurden oder die in der DTD explizit mit einem Vorgabewert deklariert wurden.

Anmerkung des Übersetzers:

Im Zusammenhang mit der Funktion **lang** wurde auf diese Eigenschaft bereits hingewiesen. Ein Attribut `xml:lang` wirkt sich zwar semantisch auf die Nachkommen aus, allerdings erscheint es nicht implizit als Attributknoten bei diesen Nachkommen.

Damit liefert z.B. `//*[lang('de')]` in der Regel eine andere Knotenmenge als `//*[@xml:lang='de']`. Im ersten Ausdruck werden alle die Knoten ausgewählt, für die selbst oder für deren nächsten Vorfahren die Sprache Deutsch (de) festgelegt wurde. Der zweite Ausdruck wählt dagegen nur die Knoten aus, die ein Attribut `xml:lang` explizit oder als Vorgabewert mit dem Wert `de` besitzen. Abgesehen davon würde die zweite Variante auch Attribute wie `xml:lang="DE-CH"` unberücksichtigt lassen.

Ein Attributknoten hat einen erweiterten Namen und einen Zeichenkettenwert. Der erweiterte Name wird durch Expandierung des im Tag im XML-Dokument angegebenen QName in Übereinstimmung mit der XML-Namensraum-Empfehlung [XML Names] berechnet. Der Namensraum-URI des Attributnamens ist leer, falls der QName des Attributs kein Präfix enthält.

Anmerkung des Übersetzers:

Entsprechend der XML-Namensraum-Empfehlung [XML Names] wirkt sich somit ein voreingestellter Namensraum im Gegensatz zu Elementknoten nicht auf Attribute ohne Präfix aus.

> **Anmerkung:** In der im Anhang A.3 von [XML Names] verwendeten Notation entspricht der lokale Teil des erweiterten Namens dem Attribut `name` des Elements `ExpAName`; der Namensraum-URI des erweiterten Namens entspricht dem Attribut `ns` des Elements `ExpAName` und ist leer, wenn das Attribut `ns` des Elements `ExpAName` weggelassen wurde.

Ein Attributknoten besitzt einen Zeichenkettenwert. Dieser Zeichenkettenwert ist der durch die XML-Empfehlung [XML] spezifizierte normalisierte Wert. Ein Attribut, dessen normalisierter Wert eine Zeichenkette der Länge null ist, wird nicht gesondert behandelt: Es resultiert in einem Attributknoten, dessen Zeichenkettenwert eine Zeichenkette der Länge null ist.

Anmerkung des Übersetzers:

Normalisierung bedeutet, dass alle Entity- und Zeichenreferenzen aufgelöst sowie alle Leerraumzeichen durch die gleiche Anzahl Leerzeichen ersetzt werden. Ist der Attributtyp nicht `CDATA`, wird der Attributwert darüber hinaus wie bei der Anwendung der Funktion **normalize-space** umgewandelt.

> **Anmerkung:** Es ist möglich, dass Attribute mit Vorgabewerten in einer externen DTD oder einem externen Parameter-Entity deklariert werden. Die XML-Empfehlung verlangt nicht, dass ein XML-Prozessor eine externe DTD oder ein externes Parameter-Entity einliest, es sei denn, dieser ist validierend. Ein

Stylesheet oder ein anderes Werkzeug, das annimmt, der XPath-Baum enthalte in einer externen DTD oder einem externen Parameter-Entity deklarierte Vorgabewerte, kann möglicherweise mit nicht-validierenden XML-Prozessoren nicht funktionieren.

Es gibt keine Attributknoten zu Attributen, die Namensräume deklarieren (siehe [XML Names]).

Anmerkung des Übersetzers:

Attribute mit dem Namen »`xmlns`« oder dem Präfix »`xmlns`« werden nicht als Attributknoten, sondern als Namensraumknoten im Datenmodell repräsentiert.

5.4 Namensraumknoten

Jedem Element ist eine Menge von Namensraumknoten zugeordnet, einer für jedes einzelne Namensraum-Präfix, das für das Element gültig ist (einschließlich des Präfixes `xml`, das implizit durch die XML-Namensraum-Empfehlung deklariert wird) und einer für den voreingestellten Namensraum, falls einer für das Element gültig ist. Das Element ist der Elternknoten jedes dieser Namensraumknoten; ein Namensraumknoten ist allerdings kein Kind seines Elternelements. Mehrere Elemente haben niemals gemeinsame Namensraumknoten: Wenn ein Elementknoten verschieden von einem anderen Elementknoten ist, dann ist kein Namensraumknoten des einen Elementknotens mit einem der Namensraumknoten eines anderen Elementknotens identisch. Das bedeutet, ein Element besitzt einen Namensraumknoten:

- für jedes Attribut des Elements, dessen Name mit `xmlns:` beginnt;
- für jedes Attribut eines Vorfahrenelements, dessen Name mit `xmlns:` beginnt, es sei denn, das Element selbst oder ein näherer Vorfahre deklariert das Präfix um;
- für ein `xmlns`-Attribut, falls das Element oder ein Vorfahre ein `xmlns`-Attribut besitzt und der Wert des `xmlns`-Attributs beim nächsten dieser Elemente nicht leer ist.

  Anmerkung: Ein Attribut `xmlns=""` "undeklariert" den voreingestellten Namensraum (siehe [XML Names]).

Anmerkung des Übersetzers:

Namensraumdeklarationen wirken sich damit in der Regel auf die Nachkommen des Elements aus, in dem diese Deklaration erscheint. Im Gegensatz zu den Attributen `xml:lang` und `xml:space` werden Namensraumknoten jedoch tatsächlich an die Nachkommen vererbt. Jeder Elementknoten besitzt ein eigenes Knoten-Exemplar für jeden gültigen Namensraum.

Wie bereits in Kapitel [**1 Einleitung**] an einem Beispiel verdeutlicht wurde, folgt aus der Namensraum-Empfehlung, dass jedes Element wenigstens einen Namensraumknoten mit dem Präfix `xml` und dem Namensraum-URI `http://www.w3.org/XML/1998/namespace` besitzt.

Da das XPath-Datenmodell keine Informationen darüber enthält, an welchen Stellen Namensraumdeklarationen im Dokument auftreten, besitzen die Beispiele

```
<foo:e1 xmlns:foo="bar"><foo:e2 /></foo:e1>
```

und

```
<foo:e1 xmlns:foo="bar"><foo:e2 xmlns:foo="bar" /></foo:e1>
```

die gleiche Baumrepräsentation.

Ein Namensraumknoten besitzt einen erweiterten Namen: Der lokale Teil ist das Namensraum-Präfix (dieses ist leer, falls der Namensraumknoten den voreingestellten Namensraum repräsentiert); der Namensraum-URI ist immer leer.

Der Zeichenkettenwert eines Namensraumknotens ist der Namensraum-URI, der an das Namensraum-Präfix gebunden ist. Wenn dieser relativ ist, muss er wie ein Namensraum-URI in einem erweiterten Namen aufgelöst werden.

Anmerkung des Übersetzers:

Nach einer Entscheidung des W3C-XML-Plenums ist die Behandlung relativer Namensraum-URIs implementationsabhängig. Gemäß Errata-Dokument [XPath Errata] muss dieser letzte Absatz durch den folgenden ersetzt werden:

> Der Zeichenkettenwert eines Namensraumknotens ist der Namensraum-URI, der an das Namensraum-Präfix gebunden ist; wenn der in der Namensraumdeklaration auftretende Namensraum-Name ein relativer URI (mit oder ohne Fragment-Bezeichner) ist, so ist der Zeichenkettenwert implementationsabhängig. Ein XPath-Ausdruck, der vom Zeichenkettenwert eines solchen Namensraumknotens abhängt, ist nicht interoperabel.

Die Definition des erweiterten Namens für Namensraumknoten mag auf den ersten Blick etwas ungewöhnlich erscheinen. Tatsächlich wurde hier das Namensraum-Präfix als eigentlich relevanter Namensbestandteil dem Schema für erweiterte Namen angepasst, den jeder Knoten besitzt. Der Namensraum-URI des Namens ist leer, da der Name eines Namensraumknotens nicht von anderen gültigen Namensraumdeklarationen abhängt.

Zur Illustration zwei Beispiele: Die Deklaration `xmlns:prefix="urn:eindeutiger-bezeichner"` erzeugt einen Namensraumknoten, dessen erweiterter Name gleich [»`prefix`«, »«] ist. Eine Deklaration für den voreingestellten Namensraum `xmlns="urn:eindeutiger-bezeichner"` führt zu einem Knoten mit dem erweiterten Namen [»«, »«]. Der Zeichenkettenwert ist in beiden Fällen »`urn:eindeutiger-bezeichner`«. Ein Namensraumknoten hat niemals einen leeren Zeichenkettenwert.

Namensraumknoten werden benötigt, wenn in den Daten enthaltene qualifizierte Namen ausgewertet werden sollen. Die Typangabe in einem XML-Schema ist so ein Beispiel: `type="ob:adresse"`. Zur Ermittlung des dazugehörigen Namensraum-URIs kann folgender Ausdruck verwendet werden:

```
string(namespace::*[name()=substring-before(../@type,':')])
```

Da der Namensraumknoten für den voreingestellten Namensraum einen leeren Namen besitzt und daher nicht direkt als Knotentest angegeben werden kann, lässt sich ein solcher Knoten nur mit Hilfe eines geeigneten Prädikats auswählen:

```
namespace::*[name()='']
```

5.5 Processing-Instruction-Knoten

Für jede Processing Instruction gibt es einen Processing-Instruction-Knoten, mit Ausnahme der Processing Instructions, die innerhalb der Dokumenttyp-Deklaration erscheinen.

Eine Processing Instruction besitzt einen erweiterten Namen: Der lokale Teil ist das Ziel der Processing Instruction; der Namensraum-URI ist leer. Der Zeichenkettenwert eines Processing-Instruction-Knotens ist der Teil der Processing Instruction, der dem Ziel und allem Leerraum folgt. Dieser beinhaltet nicht das abschließende ?>.

Anmerkung des Übersetzers:

Die folgende Processing Instruction

```
<?xml-stylesheet href='style.xsl' title='Hauptstylesheet'?>
```

wird z.B. durch einen Knoten repräsentiert, dessen erweiterter Name [»`xml-stylesheet`«, »«] und dessen Zeichenkettenwert »`href='style.xsl' title='Hauptstylesheet'`« ist.

Die Abschnitte, die hier wie Attribute einer Processing Instruction aussehen (`href` und `title`), sind in Wirklichkeit nur Teile des Inhalts, also des Zeichenkettenwerts. Durch XPath werden an dieser Stelle keine Attributknoten bereitgestellt. Möchte man auf diese Pseudo-Attribute zugreifen, muss man sich mit den Funktionen für Zeichenketten behelfen.

Namensraumdeklarationen wirken sich nicht auf Processing Instructions aus.

Anmerkung: Die XML-Deklaration ist keine Processing Instruction. Daher gibt es auch keinen Processing-Instruction-Knoten für die XML-Deklaration.

Anmerkung des Übersetzers:

Es ist nicht möglich, auf die Angaben innerhalb der XML-Deklaration zuzugreifen. Die Versionsnummer und die Kodierungsangabe gehören nicht zum XPath-Datenmodell.

5.6 Kommentarknoten

Für jeden Kommentar gibt es einen Kommentarknoten, mit Ausnahme der Kommentare, die innerhalb der Dokumenttyp-Deklaration erscheinen.

Der Zeichenkettenwert eines Kommentarknotens ist der Inhalt des Kommentars ohne die öffnenden Zeichen `<!--` und die schließenden Zeichen `-->`.

Ein Kommentarknoten besitzt keinen erweiterten Namen.

Anmerkung des Übersetzers:

Das bedeutet, dass die Funktionen **name**, **local-name** und **namespace-uri** als Ergebnis die leere Zeichenkette liefern. Das in DOM definierte Attribut `nodeName` besitzt dagegen für ein `Comment`-Objekt als Wert die Zeichenkette »`#comment`«.

5.7 Textknoten

Zeichendaten werden in Textknoten zusammengefasst. Dabei werden so viele Zeichen wie möglich in jedem Textknoten zusammengefasst: Ein Textknoten hat niemals als direkten Vorgänger oder Nachfolger einen anderen Textknoten. Der Zeichenkettenwert eines Textknotens besteht aus den Zeichendaten. Ein Textknoten enthält immer wenigstens ein Zeichen.

Jedes Zeichen innerhalb eines CDATA-Abschnittes wird wie Zeichendaten behandelt. So wird `<![CDATA[<]]>` im Quelldokument genauso behandelt wie `<`. Beides ergibt das einzelne Zeichen < in einem Textknoten innerhalb des Baumes. Ein CDATA-Abschnitt wird damit so behandelt, als würden `<![CDATA[` und `]]>` entfernt und jedes Vorkommen von < und & durch `<` bzw. `&` ersetzt werden.

Anmerkung des Übersetzers:

Das XPath-Datenmodell enthält also keinerlei Informationen darüber, in welcher Form ein Zeichen innerhalb des XML-Dokuments repräsentiert war. Die folgenden Textinhalte des Elements `x` werden alle auf den gleichen Textknoten mit dem Inhalt »A« abgebildet:

```
<x>A</x>
<x><![CDATA[A]]></x>
<x>&#65;</x>
<x>&#x0041;</x>
```

Entsprechend würde beispielsweise der Inhalt des folgenden Elements `para` in einem einzigen Textknoten zusammengefasst werden. Der Beginn des CDATA-Abschnittes kann in XPath nicht bestimmt werden. Das Document Object Model [DOM] sieht hier stattdessen spezielle Objekte `Text` und `CDATASection` vor.

```
<para>
   Hier folgen Beispiel &amp; Erklärung:
   <![CDATA[a > 10 and a < 20]]> bedeutet ...
</para>
```

Einzelne Zeichendaten werden innerhalb des Datenmodells nicht separat repräsentiert. Zur Verarbeitung muss deshalb auf die bereitgestellten Zeichenkettenfunktionen zurückgegriffen werden.

Anmerkung: Wird ein Textknoten, der das Zeichen < enthält, als XML ausgegeben, so muss das Zeichen < geschützt werden, beispielsweise durch `<` oder durch Einschluss in einen CDATA-Abschnitt.

Zeichen innerhalb von Kommentaren, Processing Instructions und Attributwerten erzeugen keine Textknoten. Zeilenenden in externen Entities werden zu #xA normalisiert, so wie in der XML-Empfehlung [XML] spezifiziert.

Anmerkung des Übersetzers:

Gemäß Errata-Dokument [XPath Errata] muss an dieser Stelle folgender Satz eingefügt werden:

> Leerraum außerhalb des Dokumentelements erzeugt keine Textknoten.

Genau genommen lässt sich durch XPath nicht beeinflussen, ob Leerraum generell Textknoten erzeugen soll. Abhängig von der Applikation, die XPath verwendet, werden für ausschließlich aus Leerraum bestehende Bereiche Textknoten angelegt oder nicht. Die Illustration zum Beispiel in Kapitel [**1 Einleitung**] beruht auf der Annahme, dass solche Leerraum-Textknoten vorhanden sind. Das ist auch das Standardverhalten in XSLT [XSLT] für zu verarbeitende XML-Dokumente. Durch Verwendung der XSLT-Elemente `xsl:strip-space` und `xsl:preserve-space` lässt sich dieses Verhalten allerdings beeinflussen. Darüber hinaus werden Leerraum-Textknoten, die als Kind eines Elements mit `xml:space="preserve"` (siehe [XML, 2nd Edition]) auftreten, niemals entfernt.

Die Existenz solcher Leerraum-Textknoten kann sich auf Kontextgröße und -position und damit auf das Ergebnis der Funktionen **last**, **position** und **count** auswirken. Die Berechnung des Ausdrucks `name/text()[1]` für das folgende Beispiel hängt entscheidend davon ab, ob Leerraum-Textknoten mitgezählt werden oder nicht:

```
<name>
   <anrede>Herr</anrede> Müller-Lüdenscheidt
</name>
```

Ein Textknoten besitzt keinen erweiterten Namen.

Anmerkung des Übersetzers:

Das bedeutet, dass die Funktionen **name**, **local-name** und **namespace-uri** als Ergebnis die leere Zeichenkette liefern. Das in DOM definierte Attribut `nodeName` besitzt dagegen für `Text`-Objekte als Wert die Zeichenkette »`#text`« und für `CDATASection`-Objekte den Wert »`#cdata-section`«.

6 Konformität

XPath ist in erster Linie als Komponente gedacht, die von anderen Spezifikationen genutzt werden kann. Demzufolge überlässt es XPath den Spezifikationen, die XPath nutzen (etwa [XPointer] und [XSLT]), Kriterien für die Konformität von XPath zu spezifizieren und definiert selbst keinerlei Konformitätskriterien für unabhängige XPath-Implementationen.

Anmerkung des Übersetzers:

Die Organisation Oasis hat eine Kommission für die Konformität von XSLT- und XPath-Implementationen [XSLT-Konformität] gebildet, die geeignete Szenarien für eine Test-Suite zusammenstellt.

A Referenzen

A.1 Normative Referenzen

IEEE 754
Institute of Electrical and Electronics Engineers: *IEEE Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std 754-1985.

RFC2396
T. Berners-Lee, R. Fielding und L. Masinter: *Uniform Resource Identifiers (URI): Generic Syntax*. IETF RFC 2396. Siehe http://www.ietf.org/rfc/rfc2396.txt.

XML
World Wide Web Consortium: *Extensible Markup Language (XML) 1.0.* W3C Recommendation. Siehe http://www.edition-w3c.de/TR/1998/REC-xml-19980210.

XML Names
World Wide Web Consortium: *Namespaces in XML*. W3C Recommendation. Siehe http://www.edition-w3c.de/TR/REC-xml-names.

A.2 Andere Referenzen

Character Model
World Wide Web Consortium: *Character Model for the World Wide Web.* W3C Working Draft. Siehe http://www.edition-w3c.de/TR/WD-charmod.

DOM
World Wide Web Consortium: *Document Object Model (DOM) Level 1 Specification.* W3C Recommendation. Siehe http://www.edition-w3c.de/TR/REC-DOM-Level-1.

JLS
J. Gosling, B. Joy und G. Steele: *The Java Language Specification*. Siehe http://java.sun.com/docs/books/jls/index.html.

ISO/IEC 10646
ISO (International Organization for Standardization): *ISO/IEC 10646-1:1993, Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*. Internationaler Standard. Siehe http://www.iso.ch/cate/d18741.html.

TEI
C.M. Sperberg-McQueen und L. Burnard: *Guidelines for Electronic Text Encoding and Interchange*. Siehe http://etext.virginia.edu/TEI.html.

Unicode
Unicode Consortium: *The Unicode Standard*. Siehe http://www.unicode.org/unicode/standard/standard.html.

XML Infoset
World Wide Web Consortium: *XML Information Set.* W3C Working Draft. Siehe http://www.edition-w3c.de/TR/xml-infoset.

XPointer

World Wide Web Consortium: *XML Pointer Language (XPointer).* W3C Working Draft. Siehe http://www.edition-w3c.de/TR/WD-xptr.

XQL

J. Robie, J. Lapp und D. Schach: *XML Query Language (XQL)*. Siehe http://www.w3.org/TandS/QL/QL98/pp/xql.html.

XSLT

World Wide Web Consortium: *XSL Transformations (XSLT).* W3C Recommendation. Siehe http://www.edition-w3c.de/TR/xslt.

B Abbildung auf die XML-Informationsmenge (nicht normativ)

Die Knoten im XPath-Datenmodell lassen sich aus den durch die XML-Informationsmenge [XML Infoset] bereitgestellten Informationseinheiten wie folgt ableiten.

> **Anmerkung:** Eine neue Version des Arbeitsentwurfs der XML-Informationsmenge, die die Version vom 17. Mai (1999, der Übersetzer) ersetzen wird, war zu der Zeit, als die Vorbereitung dieser Version der XPath-Spezifikation vollendet wurde, kurz vor der Fertigstellung. Ihre Veröffentlichung wurde zur gleichen Zeit oder kurz nach Veröffentlichung dieser Version von XPath erwartet. Die Abbildung wird für diese neue Version des Arbeitsentwurfs der XML-Informationsmenge angegeben. Falls die neue Version des Arbeitsentwurfs der XML-Informationsmenge noch nicht veröffentlicht worden sein sollte, können W3C-Mitglieder die interne Arbeitsgruppenversion http://www.w3.org/XML/Group/1999/09/WD-xml-infoset-19990915.html (nur für Mitglieder) einsehen.

Anmerkung des Übersetzers:

Die XML-Informationsmenge beschreibt die aus einem XML-Dokument ableitbaren Informationen auf einer abstrakten Ebene. Zu diesem Zweck wurden elf Typen, so genannte *Informationseinheiten* (information items) definiert. Jede dieser Informationseinheiten besitzt eine Reihe von *Eigenschaften* (properties). Es wurde in der XML-Informationsmenge bewusst nicht der Term "Knoten" verwendet, um Verwechslungen mit DOM- und XPath-Knoten zu vermeiden. Die im XPath-Datenmodell definierten Knotentypen beschreiben eine Teilmenge der in der XML-Informationsmenge verfügbaren Informationen und lassen sich daher aus diesen Informationseinheiten ableiten.

Die XML-Informationsmenge ist am 24. Oktober 2001 als W3C-Empfehlung verabschiedet worden. Im Vergleich zu der hier angesprochenen Version vom Dezember 1999 haben sich einige Details geändert. Die sich daraus ergebenden Änderungen bei der Abbildung des XPath-Datenmodells werden im Folgenden in den jeweiligen Anmerkungen angegeben.

- Der Wurzelknoten folgt aus der Dokument-Informationseinheit. Die Kinder des Wurzelknotens folgen aus den Eigenschaften *children* und *children - comments*.

Anmerkung des Übersetzers:

Die Eigenschaft *children - comments* gibt es nicht mehr. Stattdessen folgen mögliche Kommentar-Kindknoten des Wurzelknotens ebenfalls unmittelbar aus der Eigenschaft *children*.

- Ein Elementknoten folgt aus einer Element-Informationseinheit. Die Kinder eines Elementknotens folgen aus den Eigenschaften *children* und *children - comments*. Die Attribute eines Elementknotens folgen aus der Eigenschaft *attributes*. Die Namensräume eines Elementknotens folgen aus der Eigenschaft *in-scope namespaces*. Der lokale Teil des erweiterten Namens eines Elementknotens folgt aus der Eigenschaft *local name*. Der Namensraum-URI des erweiterten Namens eines Elementknotens folgt aus der Eigenschaft *namespace URI*. Der eindeutige Bezeichner (ID) eines Elementknotens folgt aus der Eigenschaft *children* der Attribut-Informationseinheit in der Eigenschaft *attributes*, deren Eigenschaft *attribute type* gleich `ID` ist.

  Anmerkung des Übersetzers:

  Die Eigenschaft *children - comments* gibt es nicht mehr. Stattdessen folgen mögliche Kommentar-Kindknoten eines Elementknotens ebenfalls unmittelbar aus der Eigenschaft *children*. Der Namensraum-URI des erweiterten Namens eines Elementknotens folgt aus der Eigenschaft *namespace name*. Der eindeutige Bezeichner (ID) eines Elementknotens folgt aus der Eigenschaft *normalized value* der Attribut-Informationseinheit in der Eigenschaft *attributes*, deren Eigenschaft *attribute type* gleich `ID` ist.

- Ein Attributknoten folgt aus einer Attribut-Informationseinheit. Der lokale Teil des erweiterten Namens des Attributknotens folgt aus der Eigenschaft *local name*. Der Namensraum-URI des erweiterten Namens des Attributknotens folgt aus der Eigenschaft *namespace URI*. Der Zeichenkettenwert des Knotens folgt aus der Verkettung aller Eigenschaften *character code* jedes Bestandteils der Eigenschaft *children*.

  Anmerkung des Übersetzers:

  Der Namensraum-URI des erweiterten Namens eines Attributknotens folgt aus der Eigenschaft *namespace name*. Der Zeichenkettenwert des Knotens folgt aus der Eigenschaft *normalized value*. Attributwerte werden gemäß der aktuellen Version der XML-Informationsmenge nicht weiter in einzelne Zeichen-Informationseinheiten zerlegt.

- Ein Textknoten folgt aus einer Folge einer oder mehrerer aufeinander folgender Zeichen-Informationseinheiten. Der Zeichenkettenwert des Knotens folgt aus der Verkettung der Eigenschaften *character code* aller Zeichen-Informationseinheiten.

- Ein Processing-Instruction-Knoten folgt aus einer Processing-Instruction-Informationseinheit. Der lokale Teil des erweiterten Namens des Knotens folgt aus der Eigenschaft *target*. (Der Teil Namensraum-URI des erweiterten Namens des Knotens ist leer.) Der Zeichenkettenwert des Knotens folgt aus der Eigenschaft *content*. Es gibt keine Processing-Instruction-Knoten für Processing-Instruction-Informationseinheiten, die Kinder der Dokumenttyp-Deklarations-Informationseinheit sind.

- Ein Kommentarknoten folgt aus einer Kommentar-Informationseinheit. Der Zeichenkettenwert des Knotens folgt aus der Eigenschaft *content*. Es gibt keine Kommentarknoten für Kommentar-Informationseinheiten, die Kinder der Dokumenttyp-Deklarations-Informationseinheit sind.

- Ein Namensraumknoten folgt aus einer Namensraumdeklarations-Informationseinheit. Der

lokale Teil des erweiterten Namens des Knotens folgt aus der Eigenschaft *prefix*. (Der Teil Namensraum-URI des erweiterten Namens des Knotens ist leer.) Der Zeichenkettenwert des Knotens folgt aus der Eigenschaft *namespace URI*.

Anmerkung des Übersetzers:

Ein Namensraumknoten folgt aus einer Namensraum-Informationseinheit. Der Zeichenkettenwert des Knotens folgt aus der Eigenschaft *namespace name*.

Zusätzliche Referenzen der deutschen Übersetzung

XML, 2nd Edition
World Wide Web Consortium: *Extensible Markup Language (XML) 1.0 (Second Edition).* W3C Recommendation. Siehe http://www.edition-w3c.de/TR/2000/REC-xml-20001006.

ISO/IEC 10646, 2nd Edition
ISO (International Organization for Standardization): *ISO/IEC 10646-1:2000, Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane, Edition: 2 (Monolingual)*. Internationaler Standard. Siehe http://www.iso.ch/cate/d29819.html.

XSLT-Konformität
Oasis (Organization for the Advancement of Structured Information Standards): *Oasis XSLT/XPath Conformance Subcommittee*. Siehe http://www.oasis-open.org/committees/xslt/.

XPath Errata
World Wide Web Consortium: *XML Path Language (XPath) Version 1.0 Specification Errata.* 29.09.2000. Siehe http://www.w3.org/1999/11/REC-xpath-19991116-errata.

XPath Requirements 2.0
World Wide Web Consortium: *XPath Requirements Version 2.0.* W3C Working Draft. Siehe http://www.edition-w3c.de/TR/xpath20req.

XPath 2.0
World Wide Web Consortium: *XML Path Language (XPath) 2.0.* W3C Working Draft. Siehe http://www.edition-w3c.de/TR/xpath20/.

XPath Operators 2.0
World Wide Web Consortium: *XQuery 1.0 and XPath 2.0 Functions and Operators.* W3C Working Draft. Siehe http://www.edition-w3c.de/TR/xquery-operators/.

XQuery
World Wide Web Consortium: *XQuery: A Query Language for XML*. W3C Working Draft. Siehe http://www.edition-w3c.de/TR/xquery.

XSLT 2.0
World Wide Web Consortium: *XSL Transformations (XSLT) Version 2.0.* W3C Working Draft. Siehe http://www.edition-w3c.de/TR/xslt20/.

Nauman Leghari's Blog

My Links
- Home
- Contact
- Syndication
- Login

Blog Stats
- Posts - 69
- Stories - 6
- Comments - 59
- Trackbacks - 9

Archives
- May, 2004 (5)
- April, 2004 (3)
- March, 2004 (3)
- February, 2004 (8)
- January, 2004 (2)
- December, 2003 (9)
- September, 2003 (1)
- August, 2003 (4)
- July, 2003 (13)
- June, 2003 (6)
- May, 2003 (6)
- April, 2003 (9)

Post Categories
- .NET (rss)
- Mobile Computing (rss)
- News (rss)
- Non-Microsoft Stuff (rss)
- Off-Topic (rss)
- Personal (rss)
- Pet Projects (rss)
- Technology (rss)

About
- Me
- My Library
- My Resume

Articles
- >> "Subscribe to ..." from IE Context Menu
- >> Building Application Frameworks with C#
- >> Creating XML-Based Message Broker in .NET
- >> Using log4net

Papers
- >> Tools and Platforms: Choices for a Mobile Application Developer

Projects
- Visual XPath

Visual XPath

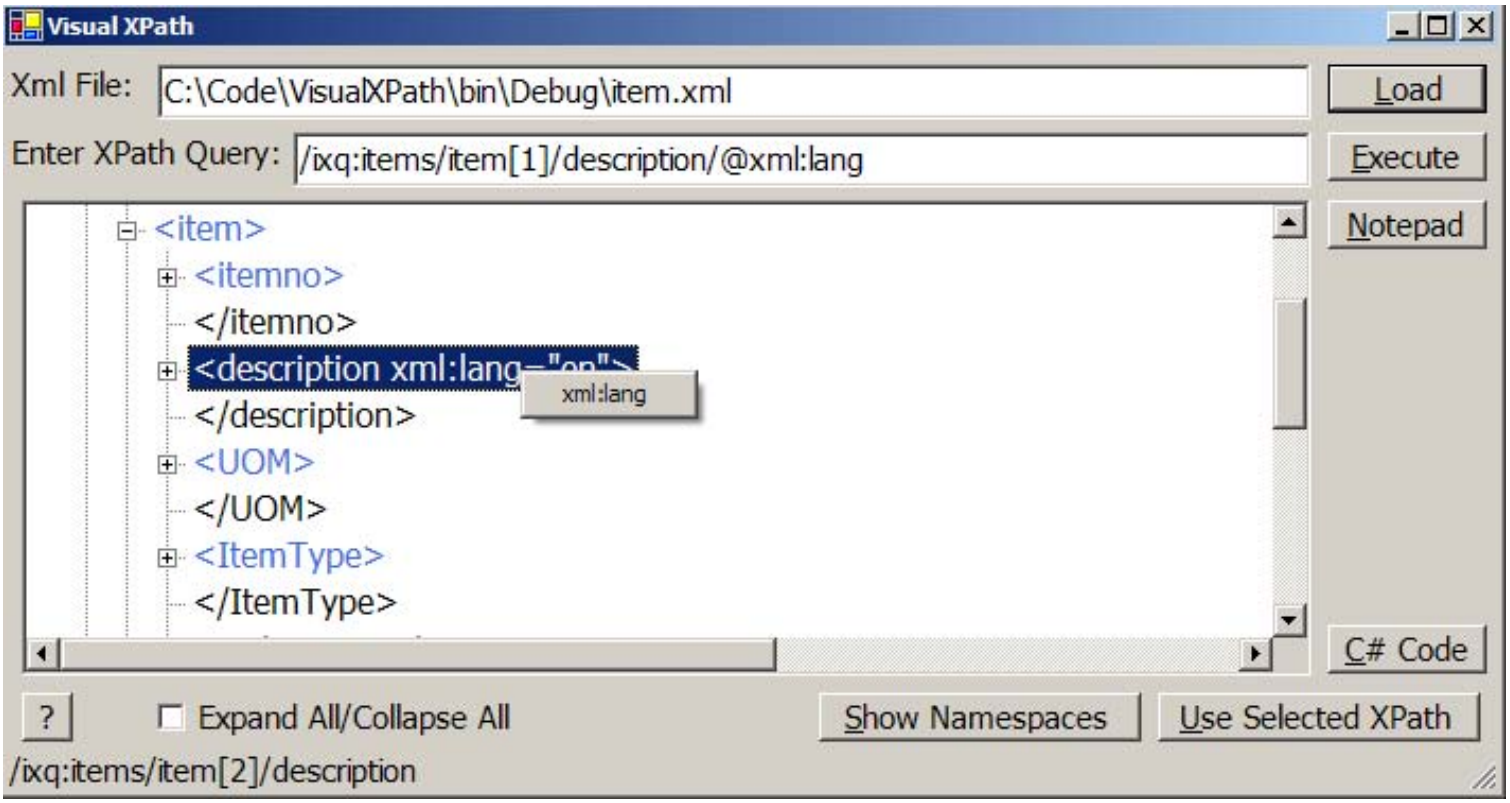
Download here

What is Visual XPath?

Visual XPath is a graphical way of generating XPath query results. It can also be used to generate XPath queries dynamically by select XML nodes shown in the form of Tree. You can also generate queries for individual attributes.

What not is Visual XPath?

Visual XPath is not an XPath Parser. It utilizes Microsoft .NET XPath parser and other XML related APIs. It does not validate XML document against DTD or Schema. Although it will display error, if the XML document is not Well Formed. It is not an XML Editor therefore I removed Editing XML and Save As XML functions from the latest version. Although the XML file, if present on your hard disk, can be opened in the Notepad from the Utility.



How to integrate with VS.NET?

Step 1: Add to the External Tools

Goto Tools->External Tools
Add Visual XPath: Put $(ItemPath) in the Arguments text box

Step 2: Map a shortcut key to this external tool

Goto Tools->Customize
Press the Keyboard button and then write "Tools.ExternalCommand(n)" in the Show command containing text box, where n is the sequence of the Visual XPath tool. You can get this number by selecting the Tools window and counting the external tools upto Visual XPath.
The assign some shortcut key for this tools. I preferred Ctrl+Shirt+X, which is also available to use.

Known Bugs: There is a bug with the tool when dealing with nested “default namespaces“ notified by Omar. Currently, the tool searches for only root level default namespaces like in the following example.

```
<?xml version="1.0"?>
<Guitars xmlns="http://www.winphone.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.winphone.com/Guitars.xsd">
....
```

It works fine with the above file but it fails to recognize any default namespaces used inside the documents as shown in the following example.

```
<item>
....
    <slash:comments>0</slash:comments>
    <body xmlns="http://www.w3.org/1999/xhtml">
      <p>
    This is a sweet phone. One of these days I'll talk a bit more about it, but
.....
```

In this rss file, the <body> tag also contains the default namespace which is treated as a document level namespace by the tool, hence putting the "def" (the word to represent default level namespace) word before any tag.

This bug only effects those files which contains the nested default namespaces.
I'll try to solve this bug as soon as possible.

Note: Email me if you have any type of comments, feature-requests and suggestions, either constructive or destructive (in case they are in the destructive category, please provide your opinion). I also need to know if there are any bugs, errors in the software. The source code is also included with the download. It is not very well documented therefore you can also email me if you want to know any part of source code or you want to know "how I code this function?" etc.

Contact Me

posted on Wednesday, September 17, 2003 4:58 PM

Comments

re: Visual XPath

Shane Courtrille

Very nice :)

Posted @ 12/4/2003 5:37 PM

Visual XPath

Eric.Canty Blog (1024x768 recommended)

Posted @ 12/8/2003 4:13 AM

re: Visual XPath

Sean Gephardt

Very cool...

Posted @ 12/12/2003 2:45 AM

re: Visual XPath

Jan Tielens

Thx! Really usefull!

Posted @ 12/14/2003 9:52 AM

re: Visual XPath

Dz

could you turn off message about CData?
because in my xml document there are like couple hundret's of them, so, i'm sure, that there is enough once to tell that it's not suported yet. ;]

Posted @ 1/28/2004 8:41 AM

re: Visual XPath

Nauman Leghari

I turned off all the dialog boxes showing "TODO.." statements. The updated version can be downloaded from the same url.

NOTE: Please leave your contact information with your comments so that I can contact you directly about the problem and the solution. Thanks.

Posted @ 1/28/2004 4:27 PM

re: Visual XPath

daku

Thank you. This has really saved me some time!

Posted @ 1/30/2004 10:35 PM

re: Visual XPath

Ashutosh

PLS..HELP

CAN I HAVE THIS TOOL FOR .NET FRAMEWORK v1.0.3530

AS I DONT HAVE TO LATEST FRAMEWORK...

Posted @ 2/13/2004 3:26 PM

re: Visual XPath SourceCode

Jens Peter Kempkes

Hello!

Can't find the SourceCode in visualxpath.zip.
Has it been removed?

This zip only contains a debug-folder.

Please Help,
jp

Posted @ 3/28/2004 9:34 AM

Post a comment

Title

re: Visual XPath

Name

Url

Comments

Remember Me

Submit

pt. purple technology

Agile, Open-Source Java and XML.

Search

Search WWW Search www.purpletech.com

Purple Technology

- News
- About Alex Chaffee
- Contact

Projects

- Purple Code
- Purple Chat Server
- XPath Explorer
- Bootstrap / Polonius
- JiffyPoll (Instant Runoff)
- JUnit Patch
- JSPXML source code
- Download This Site
- License

Writings

- Talks & Presentations
- Articles
- Maxims
- Glossary
- XP Links and Info
- Servlet Resource List

SOURCEFORGE.net

built on netBeans

eclipse

New!

View Source panel!

DocBook and CatalogResolver support! (see below)

Right-click popup menu to expand all nodes!

JDK 1.3 compatible!

Slick-looking **Kunststoff** Look-and-feel!

Much Faster when opening large documents!

Expand/Collapse All Nodes menu item!

File: Open Location menu item!

Progress Bar when loading or expanding!

Eclipse and **NetBeans** integration! (Many thanks to Simon Ru.)

File: Open menu command!

Also check out
JSP Explorer
by Mike Slinn!

XPath Explorer

Download

XPE is now hosted on SourceForge. Please go to the Project Download Page to download, in order to bump up my activity rating `:-)`. (The links below remain active, in case SourceForge is unavailable.)

- xpe.jar - runnable JAR file
- xpe-eclipse.zip - Eclipse plugin
- xpe-netbeans.jar - NetBeans plugin
- xpe-20030402-1343.tar.gz - source code
- Browse the JavaDoc

Usage:

Run the JAR using `java -jar xpe.jar`

Unzip the Eclipse plugin it into your eclipse/plugins directory; it'll appear in your Run menu and hopefully in your toolbar. If not, see below for Eclipse Plugin Hints.

What is XPath?

XPath is a syntax for navigating inside XML documents, in order to extract specific little pieces of content. It's sort of like SQL for XML. It may be better known as "the stuff inside the quotes in XSL," but it has a life (and a spec) of its own.

Example: `/doc/chapter[5]/section[2]` selects the second `section` of the fifth `chapter` of the `doc`

What is XPath Explorer?

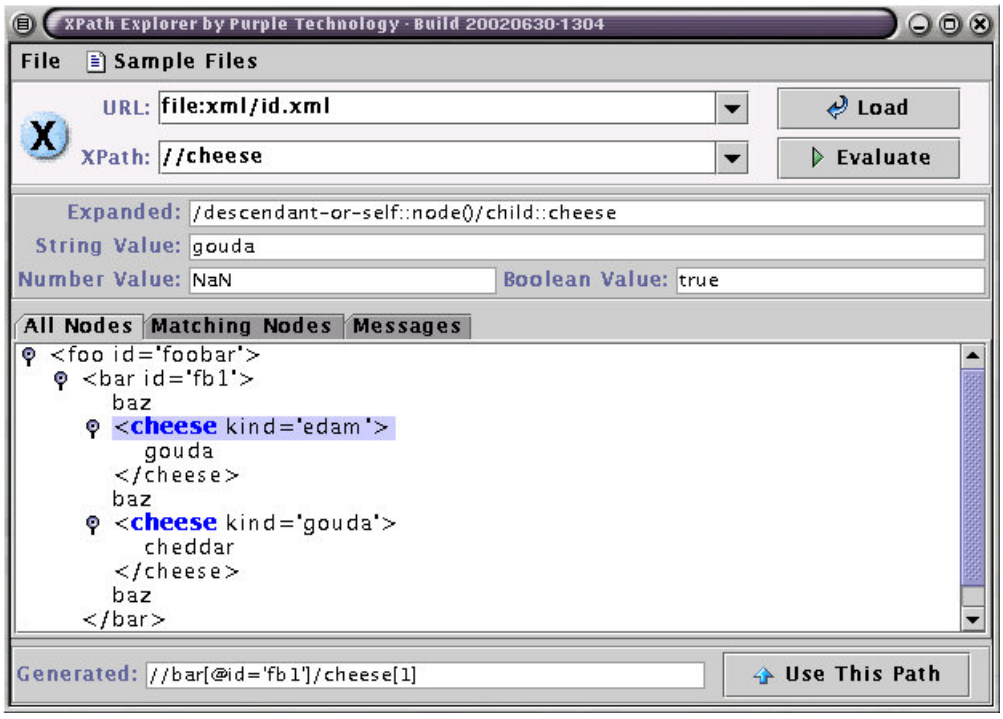
XPath Explorer (XPE) is a GUI application that lets you interactively experiment with XPath. Basically, you type in a URL (to an XML or HTML document) and an XPath expression, and it displays the elements or attributes from that document which match that expression. It also displays the value (string, number, or boolean) of the expression, and (in a stunning *coup de grace*) displays the entire XML tree of the original document, but with the matching nodes highlighted in bold. This makes it easy to play with and debug your XPath expresions.

By popular demand, we've added a reverse gear! Now when you click on a node, XPE will **generate** an XPath that uniquely identifies that node. If you double-click (or click the "Use This Path" button) it will close the circle and select all nodes that match that XPath -- should be the one you clicked on, huh?

The generation algorithm basically walks up the tree until it finds the document root, or a node with an `id` tag (since `id`s are supposed to be unique in XML), then walks back down, building the path by position.

What does it look like?

As of the beginning of July, it looks a little something like this:



Why would I want to use XPath Explorer?

Maybe you're trying to write an XSL stylesheet and you're tearing your hair out because a complicated XPath matching expression doesn't match what you think it should, and you need some debugging help.

Maybe you're using HTTPUnit to unit test your Web site, and you're sick of using the W3C DOM classes to painstakingly walk down your DOM tree. You can use XPath to jump immediately to the value you're looking for and assert that it's present. Using Jaxen, that's as easy as...

```
    assertXPathEquals("book list should contain title " + expectedTitle,
                "//td[@class='bookTitle']",
                expectedTitle,
                webresponse);

    void assertXPathEquals(String message, String xpath,
       String expected, WebResponse response) throws Exception
    {
```

```
        String value = new XPath( xpath ).valueOf( response.getDOM() );
        assertEquals(message, expected, value);
    }
```

What is Jaxen?

A great open source implementation of XPath. XPE is built on top of Jaxen. See http://www.jaxen.org.

DocBook and Catalog Support

XPath Explorer now includes the DocBook DTDs, and a nifty Catalog Resolver so it looks for them locally. This will make it easier to edit DocBook files, **if** you have the right DOCTYPE definition in your file. We recommend:

```
<!DOCTYPE section
 PUBLIC "-//OASIS//DTD DocBook XML V4.2//EN"
 "http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd" >
```

If you want to use your own locally-defined DTDs for other document types, you will have to do some work. You can try extracting the file CatalogManager.properties from xpe.jar, editing it to point to your own catalogs, then adding it back to the jar using `jar uf xpe.jar CatalogManager.properties`.

Eclipse Plugin Hints

Unzip the Eclipse plugin into your eclipse/plugins directory; then the "XPath Explorer" action and icon will appear in your Run menu and hopefully in your toolbar. What you want is for the directory "plugins" to contain a directory called "com.xpathexplorer.xpe.20030304.1234", which in turn contains the jars and other files.

If this doesn't work for you, here are some other things to try:

- Restart Eclipse. It won't notice the new plugin until you do.
- Try the "Customize Perspective" option in the Window menu. It will bring up a list of all possible plugins with buttons; if you look at the bottom you should see "XPath Explorer" or "XPath Explorer Actions". Check that checkbox, then look for the little blue X icon button in the toolbar. That will open the XPE window. (Some Eclipse versions have this option in an "Other" sublist.)
- Select "Close All Perspectives" from the Window menu. Then reopen them one at a time. When they reopen, they should notice the plugin.

To Do

```
Features:
----------
"Sample XPaths" menu (different samples for each sample document?)

output HTTPUnit test code for current XPath

"Create Test" button -- generates test for the given XPath + value

Make XML comment nodes work (display, generate)

Display rendered HTML

Servlet UI

JNLP (Web Start) packaging

make a cuter logo

display version (About Box)

check for new version

clickable link to Web site

XPath generator: allow user to specify terminal attributes (right now,
      it terminates on 'id' only; it'd be nice to allow class, name, etc.)

"Record" button: Make an HTML pane, and trap mouse clicks to record a
user sequence (click path).  Then make a HTTPUnit test that replicates
it, asserting that the values are the same along the way.

Need some way to generate paths for an attribute (right now you can't
```

```
select an attribute node) -- maybe expand the hierarchical list, like
+ <a
    id="link12"
    href="http://www.stinky.com">
    The Stinky Artists Collective
  </a>
or maybe just add a context menu (right-click) listing all the attributes

Add/remove namespaces UI (right now, it applies all namespaces in the document to
all queries)

Preserve cookies (can probably finesse HTTPUnit into doing this)

Allow HTTP password entry

Allow cookie entry

Source tab (containing original document text)

Editable source

Time evaluation (figure out how efficient your path is)

Select XML parser

Select L&F

Improve logging / status reporting; integrate with progress bar

DocBook/XML: allow user to select/add his own catalog, to point to other local DTDs

Bugs:
------

Doesn't generate correct paths if you select the document node

Doesn't generate correct paths if you select a comment node

Generates paths using "//" and the XSL gurus say "never use //"

"Collapse selected" doesn't

Regression: doesn't correctly follow HTTP redirects
```

XPath Links

- XPath Chapter in Elliotte Rusty Harold's *Processing XML with Java*
- The Jaxen-specific section of the above
- Understanding XPath article on developer.com
- XSL and XPath Quick Reference [PDF] from Mulberry Technologies
- XPath tutorial from zvon
- XPath tutorial from VBXML
- XPath spec
- Jaxen.org
- HTTPUnit

Alex Chaffee
Last modified: Fri Mar 14 08:23:25 Pacific Standard Time 2003

Page generated Mon May 24 20:58:04 PDT 2004

- Certification
- Certified Products
- Testing
- Consortia Services
- Research
- Software
- Events
- Forums
- Regional Chapters
- Membership

Online Book Store

Order books, technical documentation and memorabilia

Boundaryless Information Flow
Vision | Context

Interoperability
Pledge | Importance

Our sponsors

Election of Member-Directors is now under way

The Open Group's membership is electing four members to serve two-year terms on our Governing Board, starting from July 2004. The nominations closed on April 30th [List of candidates]. The actual voting runs from May 15 to June 11 [Members ... submit your vote]. Member-Directors make up one third of the Governing Board, so they exert a significant influence over The Open Group's direction, and responsiveness to the views of members. Full details of the election

The Open Group's Graham Bird on setting up consortia and alliance networks

Graham Bird, The Open Group's Vice President of Marketing, will discuss Setting up Consortia and Alliance Networks: Best Practices, Tricks and Traps at A.S.A.P. Marketing and Channel Alliance Conference in Dallas, Texas, June 21-22, 2004. Bird will address some of the best approaches and practices, learned over eighteen years of providing services to a number of major consortia.

The Open Group's Online Book Store now open

The Open Group's Online Book Store is now open for business, using Paypal for secure electronic payment. The major credit cards accepted include MasterCard, VISA, Discover and American Express. A wide breadth of technical publications, guides, and specifications as well as a range of memorabilia are available for online purchase. Please feel free to check it out

The Open Group Spam Survey

Unsolicited email, spam, has become a major impediment to the effectiveness of electronic mail. The Messaging Forum is focused on the impact of spam on the enterprise. Help define the problem by participating in The Open Group Spam Survey. This web-based survey should only take a few minutes of your time and will help define the business problem. The results will be presented at The Open Group's July conference in Boston.

Spotlight on recent publications: Single UNIX Specification - the 2004 Edition of the Base Specifications Issue 6

The Open Group is pleased to announce the publication of the 2004 Edition of the Base Specifications Issue 6, which are technically identical to IEEE Std 1003.1, 2004 Edition, and comprise the core volumes of the Single UNIX Specification Version 3. This latest edition incorporates the recently published Technical Corrigendum 2. Look for documents T041,C046, C047, C048 and C049 in The Open Group publications catalog

Executives Agree that Interoperability, Deperimeterization of Data and Horizontal Integration Are Essential

At The Open Group's Spring 2004 conference, *Boundaryless Information Flow™: Managing the Flow*, industry and government executives discussed the challenges of information management, and emphasized the need for pervasive interoperability and a holistic approach. With the continuous pressure on producing results, forward looking organizations want to address the whole life cycle of information management. Read more

Next Conference

- Boundaryless Information Flow™: Enterprise Information Management
  Boston, MA - July 19-23, 2004
  Join us to learn about what enterprises are doing to get information to flow systematically and what organizations are doing to tackle security issues ... and much more!

Certification News

- SIF Compliance Program: New Registration for CompassLearning
- TOGAF 7 Certification
- More TOGAF 8 Certifications
- TOGAF 8 Certifications
- LSB Certification: New Registrations for SUSE Linux AG
- [More]

- Press Releases
- Recent Articles
- Business Scenarios
- Information Sheets
- Newsletter
- White Papers
- Archived Headlines
- Gallery

Contacts · Copyright · Members · News

 Updated on Friday, 21 May 2004

ISO Focus

ISO Catalogue

ICS fields

35
Information technology. Office machines

35.060
Languages used in information technology

View Shopping Basket

The URL you've requested doesn't link to a valid catalogue entry.

Search options

- (•) Text
- ( ) ISO Number

Type in search string

Start Search

Help on using search
Extended Search

- How to use the catalogue
- Maintenance agencies and Registration Authorities
- List of withdrawn standards

JDBC
Downloads

Downloads
- Early Access
- Tools

Reference
- API Specifications
- Documentation
- FAQs
- Code Samples & Apps
- Technical Articles & Tips
- Industry Support

Community
- Books & Authors
- Code Certification
- Forums
- Bug Database

Learning
- Tutorials & Code Camps
- Online Sessions & Courses
- Instructor-Led Courses
- Quizzes

Japanese
日本語版

This page provides the following downloads and specifications:

- JDBC Rowset Implementation 1.0
- JDBC 3.0 API
- JDBC Optional Package for CDC/Foundation Profile API
- JDBC 2.1 API
- JDBC 2.0 Optional Package API
- JDBC API Test Suite 1.3.1
- JDBC API Test Suite 1.2.1

JDBC RowSet Implementations

The JDBC RowSet Implementations specification standardizes key RowSet implementations defined in the JDBC 3.0 specification. This specification will become part of the Java 2 Standard Edition 1.5 (Tiger) when it is released.

JDBC 3.0 API

All of the JDBC 3.0 API, including both the `java.sql` and `javax.sql` packages, is bundled with the Java 2 Platform, Standard Edition, version 1.4 (J2SE). If you have downloaded the J2SE platform, you already have all of the JDBC API and do not need to download anything from this page.

JDBC Optional Package for CDC/Foundation Profile API

If you write applications for the Java 2 Platform, Micro Edition, J2ME that access a relational database, you need to download this Optional Package. It is a subset of the `java.sql` package with a smaller footprint that is tailored to writing applications running CDC (Connected Device Configuration).

JDBC 2.1 Core API and JDBC 2.0 Optional Package API

General Application Programmers

> If you write only client-side applications, you do not need to download anything from this page. By downloading the Java 2 SDK, Standard Edition, you already have the JDBC core API (`java.sql` package) and the associated javadoc documentation.

Related Links

Popular Downloads
- Java Web Services Developer Pack 1.3
- J2EE 1.4 SDK
- J2SE 1.4.2
- Sun Java System Application Server

Products and Technologies
- JDBC Drivers
- Java Data Objects
- JDBC-ODBC Bridge

Sun Resources
- New to Java Center
- Developer Technical Support
- JavaOne Online
- Professional Training
- Professional Certification

Server-side Programmers

If you write Enterprise JavaBeans (EJB) components, any software that supports them, or any other server-side software, you need to download the Java 2 SDK, Enterprise Edition. The Enterprise Edition includes both the JDBC core API and the JDBC Optional Package API (the `javax.sql` package). Both packages are necessary for any JDBC application that uses a `DataSource` object to make a connection, uses connection pooling, or uses distributed transactions.

Client-side Programmers Using a `DataSource` or `RowSet` Object

In general, you might want to download the `javax.sql` package in addition to the Java 2 SDK, Standard Edition, if you do not write server-side code but want to use one of the following:

- A `DataSource` object to make a connection
  Note: To use a `DataSource` object to establish a connection, which is the recommended way, your driver must include a basic `DataSource` implementation.
- A `RowSet` object
  Rowsets are typically used as a convenient way to pass data or to add scrollability to a result set.

Downloading the `javax.sql` package instead of the entire Enterprise Edition will save you disc space. If your driver vendor bundles the `javax.sql` package with its product, as many are expected to do, you do not need to download anything beyond the Java 2 SDK, Standard Edition.

NOTE: The `javax.sql` package is also called the JDBC 2.0 Optional Package API (formerly known as the JDBC 2.0 Standard Extension API).

DOWNLOADS

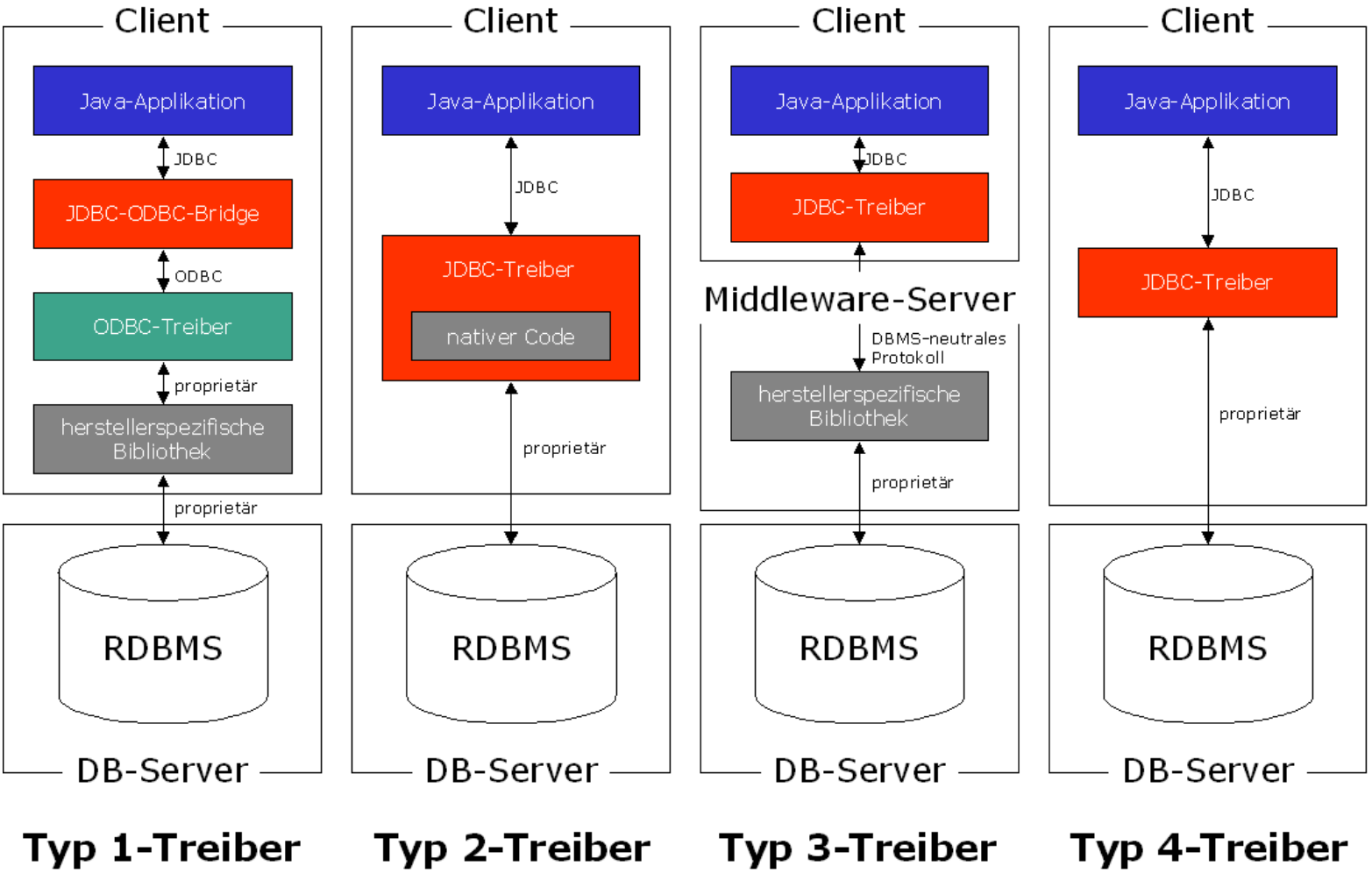
JDBC API SPECIFICATIONS	
The JDBC API Specification was created under the Java Community Process to provide full public participation in the definition and development.	
JDBC Rowset Implementations 1.0 - Final Release NEW (*Apr 7, 2004*)	
Specification	JDBC Rowset Implementations 1.0 Specification

Reference Implementation	JDBC Rowset Implementations 1.0 Reference Implementation Download
Send comments to	jsr-114-comments@jcp.org
JDBC 3.0 - Final Release (*February 13, 2002*)	
Specification	This Specification is available for viewing on-line or by downloading the document. Any use or implementation of this Specification is subject to this License agreement. Download The Adobe Acrobat Reader software is required to view the PDF format of this specification. Adobe Acrobat Reader is free and available from the Adobe Systems website.
Send comments to	jdbc@sun.com
JDBC Optional Package for CDC/Foundation Profile - Final Release (*April 8, 2004*)	
Specification	JDBC Optional Package for CDC/Foundation Profile Specification
Send comments to	jdbc-cdc@sun.com
JDBC 2.1	
JDBC 2.1 Core API	Download The Adobe Acrobat Reader software is required to view the PDF format of this specification. Adobe Acrobat Reader is free and available from the Adobe Systems website.
JDBC 2.1 Specification	View 2.1 Errata JCP 2.0 Maintenance Review APPROVED
Java 2 Platform Documentation	Java 2 Platform Documentation (contains the JDBC 2.1 Core API javadoc)

JDBC 2.1 Core Source	JDBC 2.1 Core API Source is part of Java 2
JDBC 2.1 Core Binary	JDBC 2.1 Core API is part of Java 2
JDBC 2.0 Optional Package	
JDBC 2.0 Optional package API	PostScript format (430367 bytes) PDF format (327005 bytes) The Adobe Acrobat Reader software is required to view the PDF format of this specification. Adobe Acrobat Reader is free and available from the Adobe Systems website.
JDBC 2.0 Optional Package documentation	Download the **JDBC 2.0 Optional Package API** documentation: Download Browse the JDBC 2.0 Optional Package API documentation online. Download the **JDBC 2.0 Optional Package API** documentation:
JDBC 2.0 Optional Package Source	This download contains the .java files for the javax.sql package. These consist mainly of interfaces, and a few small helper classes. The javax.sql package does not contain a JDBC driver, and it does not contain a JDBC rowset implementation. Download
JDBC 2.0 Optional Package Binary	This download contains a compiled version (.class files) of the javax.sql package. It is appropriate for inclusion in a classpath, and may be added to a Java 2 installation. Download

JDBC API Test Suite 1.3.1 (*February 21, 2002*)	
JDBC API Test Suite 1.3.1 Documentation	JDBC API Test Suite 1.3.1 Documentation for Installation and Execution
JDBC API Test Suite 1.3.1	Download
JDBC API Test Suite 1.3.1. Exclude List	Download
JDBC API Test Suite 1.2.1	
JDBC API Test Suite 1.2.1 Documentation	JDBC API Test Suite 1.2.1 Documentation for Installation and Execution
JDBC API Test Suite 1.2.1	Download
JDBC API Test Suite 1.2.1. Exclude List	Download

Looking for JDBC 1.x API Downloads? Find them here.


Client
Java-Applikation
JDBC
JDBC-ODBC-Bridge
ODBC
ODBC-Treiber
proprietär
herstellerspezifische Bibliothek
proprietär
RDBMS
DB-Server
Typ 1-Treiber
Client
Java-Applikation
JDBC
JDBC-Treiber
nativer Code
proprietär
RDBMS
DB-Server
Typ 2-Treiber
Client
Java-Applikation
JDBC
JDBC-Treiber
Middleware-Server
DBMS-neutrales Protokoll
herstellerspezifische Bibliothek
proprietär
RDBMS
DB-Server
Typ 3-Treiber
Client
Java-Applikation
JDBC
JDBC-Treiber
proprietär
RDBMS
DB-Server
Typ 4-Treiber

The world's most popular open source database

MySQL® Connector/J

MySQL Connector/J is a native Java driver that converts JDBC (Java Database Connectivity) calls into the network protocol used by the MySQL database. It lets developers working with the Java programming language easily build programs and applets that interact with MySQL and connect all corporate data, even in a heterogeneous environment. MySQL Connector/J is a Type IV JDBC driver and has a complete JDBC feature set that supports the capabilities of MySQL.

Features

The latest production version of the driver is now 50-100 percent faster in most situations than the previous version. It also creates fewer transient objects than before, leading to better performance and even more stability. The driver now also supports "streaming" result sets, which allows users to retrieve large numbers of rows without using a large memory buffer. With newly added large-packet protocol support, the driver can send rows and BLOBs up to 2 gigabytes in size.

New features from the JDBC-3.0 API in the latest production version of MySQL Connector/J include `getGeneratedKeys` which allows users to retrieve auto-increment fields in a non-database-specific way. Auto-increment fields now work with object-relational mapping tools, as well as Enterprise Java Beans (EJB) servers with Container Managed Persistence (CMP) that support JDBC-3.0.

The development version is being refactored to support new features in conjunction with version 4.1 of the MySQL database server, including server-side prepared statements and improved handling of multiple simultaneous character sets, including Unicode in the UCS2 and UTF8 encodings.

Downloads, Licensing, and Support

Three versions of MySQL Connector/J are available for download:

- MySQL Connector/J 3.0 is the production-ready version of the driver, and is available under the GPL.
- MySQL Connector/J 3.1 is the development version of the driver, and is available under the GPL.
- MySQL Connector/J 2.0 is the old production-ready version of the driver, and is available under the LGPL.

Commercial licenses for either version can also be purchased from MySQL AB, for those who don't wish to be bound by the LGPL or GPL. For more information on licensing MySQL Connector/J,

please contact us. MySQL AB also offers commercial support for MySQL Connector/J.

Interoperability

MySQL Connector/J is known to work with the following database tools and application servers:

- Apache's Jakarta Projects (Tomcat, Turbine, Velocity, etc.) — MySQL Connector/J works with all of these tools, and in most cases is the JDBC driver used in the example applications provided.
- JBoss The JBoss Open Source J2EE Application server. MySQL Connector/J works with both BMP and CMP beans in this environment .
- BEA WebLogic — MySQL Connector/J works with both BMP and CMP beans as well as servlets and JSPs on this commercial J2EE platform.
- IBM VisualAge for Java — VisualAge is a Java development environment.
- IBM WebSphere Application Server 4.0 — MySQL Connector/J works with this commercial J2EE platform. BMP, servlets, and JSPs are supported.
- Forte for Java and its open source counterpart, NetBeans — A world-class, professional IDE. The NetBeans IDE is a platform plus modules that includes components such as an editor, tools for working with source code (Java, C++, and others), version control, and a lot more.
- TableGen — TableGen automatically generates classes to represent tables within a database. It is released under the GPL.

Thinking of certifying? Try out the evaluation test.

Related pages:

- Download Binaries & Source
- Documentation
- MySQL Licensing Policy

java.sql

Interfaces

Array
Blob
CallableStatement
Clob
Connection
DatabaseMetaData
Driver
ParameterMetaData
PreparedStatement
Ref
ResultSet
ResultSetMetaData
Savepoint
SQLData
SQLInput
SQLOutput
Statement
Struct

Classes

Date
DriverManager
DriverPropertyInfo
SQLPermission
Time
Timestamp
Types

Exceptions

BatchUpdateException
DataTruncation
SQLException
SQLWarning

java.sql

Interface Connection

public interface **Connection**

A connection (session) with a specific database. SQL statements are executed and results are returned within the context of a connection.

A `Connection` object's database is able to provide information describing its tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, and so on. This information is obtained with the `getMetaData` method.

Note: By default a `Connection` object is in auto-commit mode, which means that it automatically commits changes after executing each statement. If auto-commit mode has been disabled, the method `commit` must be called explicitly in order to commit changes; otherwise, database changes will not be saved.

A new `Connection` object created using the JDBC 2.1 core API has an initially empty type map associated with it. A user may enter a custom mapping for a UDT in this type map. When a UDT is retrieved from a data source with the method `ResultSet.getObject`, the `getObject` method will check the connection's type map to see if there is an entry for that UDT. If so, the `getObject` method will map the UDT to the class indicated. If there is no entry, the UDT will be mapped using the standard mapping.

A user may create a new type map, which is a `java.util.Map` object, make an entry in it, and pass it to the `java.sql` methods that can perform custom mapping. In this case, the method will use the given type map instead of the one associated with the connection.

For example, the following code fragment specifies that the SQL type `ATHLETES` will be mapped to the class `Athletes` in the Java programming language. The code fragment retrieves the type map for the `Connection` object `con`, inserts the entry into it, and then sets the type map with the new entry as the connection's type map.

```
java.util.Map map = con.getTypeMap();
map.put("mySchemaName.ATHLETES", Class.forName("Athletes"));
con.setTypeMap(map);
```

See Also:

`DriverManager.getConnection(java.lang.String, java.util.Properties)`, `Statement`, `ResultSet`, `DatabaseMetaData`

Field Summary

`static int`	**`TRANSACTION_NONE`** A constant indicating that transactions are not supported.
`static int`	**`TRANSACTION_READ_COMMITTED`** A constant indicating that dirty reads are prevented; non-repeatable reads and phantom reads can occur.
`static int`	**`TRANSACTION_READ_UNCOMMITTED`** A constant indicating that dirty reads, non-repeatable reads and phantom reads can occur.
`static int`	**`TRANSACTION_REPEATABLE_READ`** A constant indicating that dirty reads and non-repeatable reads are prevented; phantom reads can occur.
`static int`	**`TRANSACTION_SERIALIZABLE`** A constant indicating that dirty reads, non-repeatable reads and phantom reads are prevented.

Method Summary

`void`	**`clearWarnings`**`()` Clears all warnings reported for this `Connection` object.
`void`	**`close`**`()` Releases this `Connection` object's database and JDBC resources immediately instead of waiting for them to be automatically released.
`void`	**`commit`**`()` Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by this `Connection` object.
`Statement`	**`createStatement`**`()` Creates a `Statement` object for sending SQL statements to the database.

Statement	**createStatement**(int resultSetType, int resultSetConcurrency) Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
Statement	**createStatement**(int resultSetType, int resultSetConcurrency, int resultSetHoldability) Creates a Statement object that will generate ResultSet objects with the given type, concurrency, and holdability.
boolean	**getAutoCommit**() Retrieves the current auto-commit mode for this Connection object.
String	**getCatalog**() Retrieves this Connection object's current catalog name.
int	**getHoldability**() Retrieves the current holdability of ResultSet objects created using this Connection object.
DatabaseMetaData	**getMetaData**() Retrieves a DatabaseMetaData object that contains metadata about the database to which this Connection object represents a connection.
int	**getTransactionIsolation**() Retrieves this Connection object's current transaction isolation level.
Map	**getTypeMap**() Retrieves the Map object associated with this Connection object.
SQLWarning	**getWarnings**() Retrieves the first warning reported by calls on this Connection object.
boolean	**isClosed**() Retrieves whether this Connection object has been closed.
boolean	**isReadOnly**() Retrieves whether this Connection object is in read-only mode.
String	**nativeSQL**(String sql) Converts the given SQL statement into the system's native SQL grammar.

CallableStatement	**prepareCall**(String sql) Creates a CallableStatement object for calling database stored procedures.
CallableStatement	**prepareCall**(String sql, int resultSetType, int resultSetConcurrency) Creates a CallableStatement object that will generate ResultSet objects with the given type and concurrency.
CallableStatement	**prepareCall**(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) Creates a CallableStatement object that will generate ResultSet objects with the given type and concurrency.
PreparedStatement	**prepareStatement**(String sql) Creates a PreparedStatement object for sending parameterized SQL statements to the database.
PreparedStatement	**prepareStatement**(String sql, int autoGeneratedKeys) Creates a default PreparedStatement object that has the capability to retrieve auto-generated keys.
PreparedStatement	**prepareStatement**(String sql, int[] columnIndexes) Creates a default PreparedStatement object capable of returning the auto-generated keys designated by the given array.
PreparedStatement	**prepareStatement**(String sql, int resultSetType, int resultSetConcurrency) Creates a PreparedStatement object that will generate ResultSet objects with the given type and concurrency.
PreparedStatement	**prepareStatement**(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) Creates a PreparedStatement object that will generate ResultSet objects with the given type, concurrency, and holdability.
PreparedStatement	**prepareStatement**(String sql, String [] columnNames) Creates a default PreparedStatement object capable of returning the auto-generated keys designated by the given array.
void	**releaseSavepoint**(Savepoint savepoint) Removes the given Savepoint object from the current transaction.

void	**rollback**() Undoes all changes made in the current transaction and releases any database locks currently held by this Connection object.
void	**rollback**(Savepoint savepoint) Undoes all changes made after the given Savepoint object was set.
void	**setAutoCommit**(boolean autoCommit) Sets this connection's auto-commit mode to the given state.
void	**setCatalog**(String catalog) Sets the given catalog name in order to select a subspace of this Connection object's database in which to work.
void	**setHoldability**(int holdability) Changes the holdability of ResultSet objects created using this Connection object to the given holdability.
void	**setReadOnly**(boolean readOnly) Puts this connection in read-only mode as a hint to the driver to enable database optimizations.
Savepoint	**setSavepoint**() Creates an unnamed savepoint in the current transaction and returns the new Savepoint object that represents it.
Savepoint	**setSavepoint**(String name) Creates a savepoint with the given name in the current transaction and returns the new Savepoint object that represents it.
void	**setTransactionIsolation**(int level) Attempts to change the transaction isolation level for this Connection object to the one given.
void	**setTypeMap**(Map map) Installs the given TypeMap object as the type map for this Connection object.

Field Detail

TRANSACTION_NONE

```
public static final int TRANSACTION_NONE
```

A constant indicating that transactions are not supported.

See Also:

Constant Field Values

TRANSACTION_READ_UNCOMMITTED

```
public static final int TRANSACTION_READ_UNCOMMITTED
```

A constant indicating that dirty reads, non-repeatable reads and phantom reads can occur. This level allows a row changed by one transaction to be read by another transaction before any changes in that row have been committed (a "dirty read"). If any of the changes are rolled back, the second transaction will have retrieved an invalid row.

See Also:

Constant Field Values

TRANSACTION_READ_COMMITTED

```
public static final int TRANSACTION_READ_COMMITTED
```

A constant indicating that dirty reads are prevented; non-repeatable reads and phantom reads can occur. This level only prohibits a transaction from reading a row with uncommitted changes in it.

See Also:

Constant Field Values

TRANSACTION_REPEATABLE_READ

```
public static final int TRANSACTION_REPEATABLE_READ
```

A constant indicating that dirty reads and non-repeatable reads are prevented; phantom reads can occur. This level prohibits a transaction from reading a row with uncommitted changes in it, and it also prohibits the situation where one transaction reads a row, a second transaction alters the row, and the first transaction rereads the row, getting different values the second time (a "non-repeatable read").

See Also:

Constant Field Values

TRANSACTION_SERIALIZABLE

```
public static final int TRANSACTION_SERIALIZABLE
```

A constant indicating that dirty reads, non-repeatable reads and phantom reads are prevented. This level includes the prohibitions in `TRANSACTION_REPEATABLE_READ` and further prohibits the situation where one transaction reads all rows that satisfy a `WHERE` condition, a second transaction inserts a row that satisfies that `WHERE` condition, and the first transaction rereads for the same condition, retrieving the additional "phantom" row in the second read.

See Also:

Constant Field Values

Method Detail

createStatement

```
public Statement createStatement()
                           throws SQLException
```

Creates a `Statement` object for sending SQL statements to the database. SQL statements without parameters are normally executed using `Statement` objects. If the same SQL statement is executed many times, it may be more efficient to use a `PreparedStatement` object.

Result sets created using the returned `Statement` object will by default be type `TYPE_FORWARD_ONLY` and have a concurrency level of `CONCUR_READ_ONLY`.

Returns:

a new default `Statement` object

Throws:

`SQLException` - if a database access error occurs

---

prepareStatement

```
public PreparedStatement prepareStatement(String sql)
```

```
                                    throws SQLException
```

Creates a `PreparedStatement` object for sending parameterized SQL statements to the database.

A SQL statement with or without IN parameters can be pre-compiled and stored in a `PreparedStatement` object. This object can then be used to efficiently execute this statement multiple times.

Note: This method is optimized for handling parametric SQL statements that benefit from precompilation. If the driver supports precompilation, the method `prepareStatement` will send the statement to the database for precompilation. Some drivers may not support precompilation. In this case, the statement may not be sent to the database until the `PreparedStatement` object is executed. This has no direct effect on users; however, it does affect which methods throw certain `SQLException` objects.

Result sets created using the returned `PreparedStatement` object will by default be type `TYPE_FORWARD_ONLY` and have a concurrency level of `CONCUR_READ_ONLY`.

Parameters:

`sql` - an SQL statement that may contain one or more '?' IN parameter placeholders

Returns:

a new default `PreparedStatement` object containing the pre-compiled SQL statement

Throws:

`SQLException` - if a database access error occurs

---

prepareCall

```
public CallableStatement prepareCall(String sql)
                              throws SQLException
```

Creates a `CallableStatement` object for calling database stored procedures. The `CallableStatement` object provides methods for setting up its IN and OUT parameters, and methods for executing the call to a stored procedure.

Note: This method is optimized for handling stored procedure call statements. Some drivers may send the call statement to the database when the method `prepareCall` is done; others may wait until the `CallableStatement` object is executed. This has no direct effect on users; however, it does affect which method throws certain SQLExceptions.

Result sets created using the returned `CallableStatement` object will by default be type

TYPE_FORWARD_ONLY and have a concurrency level of CONCUR_READ_ONLY.

Parameters:
sql - an SQL statement that may contain one or more '?' parameter placeholders. Typically this statement is a JDBC function call escape string.
Returns:
a new default CallableStatement object containing the pre-compiled SQL statement
Throws:
SQLException - if a database access error occurs

---

nativeSQL

```
public String nativeSQL(String sql)
                 throws SQLException
```

Converts the given SQL statement into the system's native SQL grammar. A driver may convert the JDBC SQL grammar into its system's native SQL grammar prior to sending it. This method returns the native form of the statement that the driver would have sent.

Parameters:
sql - an SQL statement that may contain one or more '?' parameter placeholders
Returns:
the native form of this statement
Throws:
SQLException - if a database access error occurs

---

setAutoCommit

```
public void setAutoCommit(boolean autoCommit)
                   throws SQLException
```

Sets this connection's auto-commit mode to the given state. If a connection is in auto-commit mode, then all its SQL statements will be executed and committed as individual transactions. Otherwise, its SQL statements are grouped into transactions that are terminated by a call to either the method commit or the method rollback. By default, new connections are in auto-commit mode.

The commit occurs when the statement completes or the next execute occurs, whichever comes first. In the case of statements returning a ResultSet object, the statement completes

when the last row of the `ResultSet` object has been retrieved or the `ResultSet` object has been closed. In advanced cases, a single statement may return multiple results as well as output parameter values. In these cases, the commit occurs when all results and output parameter values have been retrieved.

NOTE: If this method is called during a transaction, the transaction is committed.

Parameters:
`autoCommit` - `true` to enable auto-commit mode; `false` to disable it
Throws:
`SQLException` - if a database access error occurs
See Also:
`getAutoCommit()`

---

getAutoCommit

```
public boolean getAutoCommit()
                      throws SQLException
```

Retrieves the current auto-commit mode for this `Connection` object.

Returns:
the current state of this `Connection` object's auto-commit mode
Throws:
`SQLException` - if a database access error occurs
See Also:
`setAutoCommit(boolean)`

---

commit

```
public void commit()
            throws SQLException
```

Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by this `Connection` object. This method should be used only when auto-commit mode has been disabled.

Throws:
`SQLException` - if a database access error occurs or this `Connection` object is in

auto-commit mode

See Also:

`setAutoCommit(boolean)`

---

rollback

```
public void rollback()
            throws SQLException
```

Undoes all changes made in the current transaction and releases any database locks currently held by this `Connection` object. This method should be used only when auto-commit mode has been disabled.

Throws:

`SQLException` - if a database access error occurs or this `Connection` object is in auto-commit mode

See Also:

`setAutoCommit(boolean)`

---

close

```
public void close()
          throws SQLException
```

Releases this `Connection` object's database and JDBC resources immediately instead of waiting for them to be automatically released.

Calling the method `close` on a `Connection` object that is already closed is a no-op.

Note: A `Connection` object is automatically closed when it is garbage collected. Certain fatal errors also close a `Connection` object.

Throws:

`SQLException` - if a database access error occurs

---

isClosed

```
public boolean isClosed()
                throws SQLException
```

Retrieves whether this `Connection` object has been closed. A connection is closed if the method `close` has been called on it or if certain fatal errors have occurred. This method is guaranteed to return `true` only when it is called after the method `Connection.close` has been called.

This method generally cannot be called to determine whether a connection to a database is valid or invalid. A typical client can determine that a connection is invalid by catching any exceptions that might be thrown when an operation is attempted.

Returns:

`true` if this `Connection` object is closed; `false` if it is still open

Throws:

`SQLException` - if a database access error occurs

---

getMetaData

```
public DatabaseMetaData getMetaData()
                            throws SQLException
```

Retrieves a `DatabaseMetaData` object that contains metadata about the database to which this `Connection` object represents a connection. The metadata includes information about the database's tables, its supported SQL grammar, its stored procedures, the capabilities of this connection, and so on.

Returns:

a `DatabaseMetaData` object for this `Connection` object

Throws:

`SQLException` - if a database access error occurs

---

setReadOnly

```
public void setReadOnly(boolean readOnly)
                throws SQLException
```

Puts this connection in read-only mode as a hint to the driver to enable database optimizations.

Note: This method cannot be called during a transaction.

Parameters:
`readOnly` - `true` enables read-only mode; `false` disables it
Throws:
`SQLException` - if a database access error occurs or this method is called during a transaction

---

isReadOnly

```
public boolean isReadOnly()
                  throws SQLException
```

Retrieves whether this `Connection` object is in read-only mode.

Returns:
`true` if this `Connection` object is read-only; `false` otherwise
Throws:
`SQLException` - if a database access error occurs

---

setCatalog

```
public void setCatalog(String catalog)
                throws SQLException
```

Sets the given catalog name in order to select a subspace of this `Connection` object's database in which to work.

If the driver does not support catalogs, it will silently ignore this request.

Parameters:
`catalog` - the name of a catalog (subspace in this `Connection` object's database) in which to work
Throws:
`SQLException` - if a database access error occurs
See Also:
`getCatalog()`

---

getCatalog

```
public String getCatalog()
                  throws SQLException
```

Retrieves this `Connection` object's current catalog name.

Returns:
the current catalog name or `null` if there is none

Throws:
`SQLException` - if a database access error occurs

See Also:
`setCatalog(java.lang.String)`

---

setTransactionIsolation

```
public void setTransactionIsolation(int level)
                            throws SQLException
```

Attempts to change the transaction isolation level for this `Connection` object to the one given. The constants defined in the interface `Connection` are the possible transaction isolation levels.

Note: If this method is called during a transaction, the result is implementation-defined.

Parameters:
`level` - one of the following `Connection` constants: `Connection.TRANSACTION_READ_UNCOMMITTED`, `Connection.TRANSACTION_READ_COMMITTED`, `Connection.TRANSACTION_REPEATABLE_READ`, or `Connection.TRANSACTION_SERIALIZABLE`. (Note that `Connection.TRANSACTION_NONE` cannot be used because it specifies that transactions are not supported.)

Throws:
`SQLException` - if a database access error occurs or the given parameter is not one of the `Connection` constants

See Also:
`DatabaseMetaData.supportsTransactionIsolationLevel(int)`, `getTransactionIsolation()`

---

getTransactionIsolation

```
public int getTransactionIsolation()
                            throws SQLException
```

Retrieves this `Connection` object's current transaction isolation level.

Returns:
the current transaction isolation level, which will be one of the following constants: `Connection.TRANSACTION_READ_UNCOMMITTED`, `Connection.TRANSACTION_READ_COMMITTED`, `Connection.TRANSACTION_REPEATABLE_READ`, `Connection.TRANSACTION_SERIALIZABLE`, or `Connection.TRANSACTION_NONE`.

Throws:
`SQLException` - if a database access error occurs

See Also:
`setTransactionIsolation(int)`

---

getWarnings

```
public SQLWarning getWarnings()
                         throws SQLException
```

Retrieves the first warning reported by calls on this `Connection` object. If there is more than one warning, subsequent warnings will be chained to the first one and can be retrieved by calling the method `SQLWarning.getNextWarning` on the warning that was retrieved previously.

This method may not be called on a closed connection; doing so will cause an `SQLException` to be thrown.

Note: Subsequent warnings will be chained to this SQLWarning.

Returns:
the first `SQLWarning` object or `null` if there are none

Throws:
`SQLException` - if a database access error occurs or this method is called on a closed connection

See Also:
`SQLWarning`

clearWarnings

```
public void clearWarnings()
                    throws SQLException
```

Clears all warnings reported for this `Connection` object. After a call to this method, the method `getWarnings` returns `null` until a new warning is reported for this `Connection` object.

Throws:

`SQLException` - if a database access error occurs

createStatement

```
public Statement createStatement(int resultSetType,
                                 int resultSetConcurrency)
                          throws SQLException
```

Creates a `Statement` object that will generate `ResultSet` objects with the given type and concurrency. This method is the same as the `createStatement` method above, but it allows the default result set type and concurrency to be overridden.

Parameters:

`resultSetType` - a result set type; one of `ResultSet.TYPE_FORWARD_ONLY`, `ResultSet.TYPE_SCROLL_INSENSITIVE`, or `ResultSet.TYPE_SCROLL_SENSITIVE`

`resultSetConcurrency` - a concurrency type; one of `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`

Returns:

a new `Statement` object that will generate `ResultSet` objects with the given type and concurrency

Throws:

`SQLException` - if a database access error occurs or the given parameters are not `ResultSet` constants indicating type and concurrency

Since:

1.2

prepareStatement

```
public PreparedStatement prepareStatement(String sql,
                                          int resultSetType,
                                          int resultSetConcurrency)
                                   throws SQLException
```

Creates a `PreparedStatement` object that will generate `ResultSet` objects with the given type and concurrency. This method is the same as the `prepareStatement` method above, but it allows the default result set type and concurrency to be overridden.

Parameters:

`sql` - a `String` object that is the SQL statement to be sent to the database; may contain one or more ? IN parameters

`resultSetType` - a result set type; one of `ResultSet.TYPE_FORWARD_ONLY`, `ResultSet.TYPE_SCROLL_INSENSITIVE`, or `ResultSet.TYPE_SCROLL_SENSITIVE`

`resultSetConcurrency` - a concurrency type; one of `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`

Returns:

a new PreparedStatement object containing the pre-compiled SQL statement that will produce `ResultSet` objects with the given type and concurrency

Throws:

`SQLException` - if a database access error occurs or the given parameters are not `ResultSet` constants indicating type and concurrency

Since:

1.2

---

prepareCall

```
public CallableStatement prepareCall(String sql,
                                     int resultSetType,
                                     int resultSetConcurrency)
                              throws SQLException
```

Creates a `CallableStatement` object that will generate `ResultSet` objects with the given type and concurrency. This method is the same as the `prepareCall` method above, but it allows the default result set type and concurrency to be overridden.

Parameters:

`sql` - a `String` object that is the SQL statement to be sent to the database; may contain on or more ? parameters

`resultSetType` - a result set type; one of `ResultSet.TYPE_FORWARD_ONLY`, `ResultSet.TYPE_SCROLL_INSENSITIVE`, or `ResultSet.`

`TYPE_SCROLL_SENSITIVE`
`resultSetConcurrency` - a concurrency type; one of `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`

Returns:

a new `CallableStatement` object containing the pre-compiled SQL statement that will produce `ResultSet` objects with the given type and concurrency

Throws:

`SQLException` - if a database access error occurs or the given parameters are not `ResultSet` constants indicating type and concurrency

Since:

1.2

---

getTypeMap

```
public Map getTypeMap()
                throws SQLException
```

Retrieves the `Map` object associated with this `Connection` object. Unless the application has added an entry, the type map returned will be empty.

Returns:

the `java.util.Map` object associated with this `Connection` object

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

See Also:

`setTypeMap(java.util.Map)`

---

setTypeMap

```
public void setTypeMap(Map map)
                 throws SQLException
```

Installs the given `TypeMap` object as the type map for this `Connection` object. The type map will be used for the custom mapping of SQL structured types and distinct types.

Parameters:

`map` - the `java.util.Map` object to install as the replacement for this `Connection` object's default type map

Throws:

SQLException - if a database access error occurs or the given parameter is not a `java.util.Map` object

Since:

1.2

See Also:

getTypeMap()

---

setHoldability

```
public void setHoldability(int holdability)
                    throws SQLException
```

Changes the holdability of `ResultSet` objects created using this `Connection` object to the given holdability.

Parameters:

`holdability` - a `ResultSet` holdability constant; one of `ResultSet.HOLD_CURSORS_OVER_COMMIT` or `ResultSet.CLOSE_CURSORS_AT_COMMIT`

Throws:

SQLException - if a database access occurs, the given parameter is not a `ResultSet` constant indicating holdability, or the given holdability is not supported

Since:

1.4

See Also:

getHoldability(), ResultSet

---

getHoldability

```
public int getHoldability()
                  throws SQLException
```

Retrieves the current holdability of `ResultSet` objects created using this `Connection` object.

Returns:

the holdability, one of `ResultSet.HOLD_CURSORS_OVER_COMMIT` or `ResultSet.CLOSE_CURSORS_AT_COMMIT`

Throws:

SQLException - if a database access occurs

Since:

1.4

See Also:

setHoldability(int), ResultSet

---

setSavepoint

```
public Savepoint setSavepoint()
                        throws SQLException
```

Creates an unnamed savepoint in the current transaction and returns the new `Savepoint` object that represents it.

Returns:

the new `Savepoint` object

Throws:

SQLException - if a database access error occurs or this `Connection` object is currently in auto-commit mode

Since:

1.4

See Also:

Savepoint

---

setSavepoint

```
public Savepoint setSavepoint(String name)
                        throws SQLException
```

Creates a savepoint with the given name in the current transaction and returns the new `Savepoint` object that represents it.

Parameters:

`name` - a `String` containing the name of the savepoint

Returns:

the new `Savepoint` object

Throws:

SQLException - if a database access error occurs or this `Connection` object is currently in auto-commit mode

Since:
1.4
See Also:
Savepoint

---

rollback

```
public void rollback(Savepoint savepoint)
              throws SQLException
```

Undoes all changes made after the given `Savepoint` object was set.

This method should be used only when auto-commit has been disabled.

Parameters:
`savepoint` - the `Savepoint` object to roll back to
Throws:
`SQLException` - if a database access error occurs, the `Savepoint` object is no longer valid, or this `Connection` object is currently in auto-commit mode
Since:
1.4
See Also:
`Savepoint`, `rollback()`

---

releaseSavepoint

```
public void releaseSavepoint(Savepoint savepoint)
                      throws SQLException
```

Removes the given `Savepoint` object from the current transaction. Any reference to the savepoint after it have been removed will cause an `SQLException` to be thrown.

Parameters:
`savepoint` - the `Savepoint` object to be removed
Throws:
`SQLException` - if a database access error occurs or the given `Savepoint` object is not a valid savepoint in the current transaction
Since:
1.4

---

createStatement

```
public Statement createStatement(int resultSetType,
                                 int resultSetConcurrency,
                                 int resultSetHoldability)
                          throws SQLException
```

Creates a `Statement` object that will generate `ResultSet` objects with the given type, concurrency, and holdability. This method is the same as the `createStatement` method above, but it allows the default result set type, concurrency, and holdability to be overridden.

Parameters:

`resultSetType` - one of the following `ResultSet` constants: `ResultSet.TYPE_FORWARD_ONLY`, `ResultSet.TYPE_SCROLL_INSENSITIVE`, or `ResultSet.TYPE_SCROLL_SENSITIVE`

`resultSetConcurrency` - one of the following `ResultSet` constants: `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`

`resultSetHoldability` - one of the following `ResultSet` constants: `ResultSet.HOLD_CURSORS_OVER_COMMIT` or `ResultSet.CLOSE_CURSORS_AT_COMMIT`

Returns:

a new `Statement` object that will generate `ResultSet` objects with the given type, concurrency, and holdability

Throws:

`SQLException` - if a database access error occurs or the given parameters are not `ResultSet` constants indicating type, concurrency, and holdability

Since:

1.4

See Also:

`ResultSet`

---

prepareStatement

```
public PreparedStatement prepareStatement(String sql,
                                          int resultSetType,
                                          int resultSetConcurrency,
                                          int resultSetHoldability)
                                   throws SQLException
```

Creates a `PreparedStatement` object that will generate `ResultSet` objects with the

given type, concurrency, and holdability.

This method is the same as the `prepareStatement` method above, but it allows the default result set type, concurrency, and holdability to be overridden.

Parameters:

`sql` - a `String` object that is the SQL statement to be sent to the database; may contain one or more ? IN parameters
`resultSetType` - one of the following `ResultSet` constants: `ResultSet.TYPE_FORWARD_ONLY`, `ResultSet.TYPE_SCROLL_INSENSITIVE`, or `ResultSet.TYPE_SCROLL_SENSITIVE`
`resultSetConcurrency` - one of the following `ResultSet` constants: `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`
`resultSetHoldability` - one of the following `ResultSet` constants: `ResultSet.HOLD_CURSORS_OVER_COMMIT` or `ResultSet.CLOSE_CURSORS_AT_COMMIT`

Returns:

a new `PreparedStatement` object, containing the pre-compiled SQL statement, that will generate `ResultSet` objects with the given type, concurrency, and holdability

Throws:

`SQLException` - if a database access error occurs or the given parameters are not `ResultSet` constants indicating type, concurrency, and holdability

Since:

1.4

See Also:

`ResultSet`

---

prepareCall

```
public CallableStatement prepareCall(String sql,
                                     int resultSetType,
                                     int resultSetConcurrency,
                                     int resultSetHoldability)
                              throws SQLException
```

Creates a `CallableStatement` object that will generate `ResultSet` objects with the given type and concurrency. This method is the same as the `prepareCall` method above, but it allows the default result set type, result set concurrency type and holdability to be overridden.

Parameters:

`sql` - a `String` object that is the SQL statement to be sent to the database; may

contain on or more ? parameters
`resultSetType` - one of the following `ResultSet` constants: `ResultSet.TYPE_FORWARD_ONLY`, `ResultSet.TYPE_SCROLL_INSENSITIVE`, or `ResultSet.TYPE_SCROLL_SENSITIVE`
`resultSetConcurrency` - one of the following `ResultSet` constants: `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`
`resultSetHoldability` - one of the following `ResultSet` constants: `ResultSet.HOLD_CURSORS_OVER_COMMIT` or `ResultSet.CLOSE_CURSORS_AT_COMMIT`

Returns:

a new `CallableStatement` object, containing the pre-compiled SQL statement, that will generate `ResultSet` objects with the given type, concurrency, and holdability

Throws:

`SQLException` - if a database access error occurs or the given parameters are not `ResultSet` constants indicating type, concurrency, and holdability

Since:

1.4

See Also:

`ResultSet`

---

prepareStatement

```
public PreparedStatement prepareStatement(String sql,
                                          int autoGeneratedKeys)
                                   throws SQLException
```

Creates a default `PreparedStatement` object that has the capability to retrieve auto-generated keys. The given constant tells the driver whether it should make auto-generated keys available for retrieval. This parameter is ignored if the SQL statement is not an `INSERT` statement.

Note: This method is optimized for handling parametric SQL statements that benefit from precompilation. If the driver supports precompilation, the method `prepareStatement` will send the statement to the database for precompilation. Some drivers may not support precompilation. In this case, the statement may not be sent to the database until the `PreparedStatement` object is executed. This has no direct effect on users; however, it does affect which methods throw certain SQLExceptions.

Result sets created using the returned `PreparedStatement` object will by default be type `TYPE_FORWARD_ONLY` and have a concurrency level of `CONCUR_READ_ONLY`.

Parameters:

`sql` - an SQL statement that may contain one or more '?' IN parameter placeholders
`autoGeneratedKeys` - a flag indicating whether auto-generated keys should be returned; one of `Statement.RETURN_GENERATED_KEYS` or `Statement.NO_GENERATED_KEYS`

Returns:

a new `PreparedStatement` object, containing the pre-compiled SQL statement, that will have the capability of returning auto-generated keys

Throws:

`SQLException` - if a database access error occurs or the given parameter is not a `Statement` constant indicating whether auto-generated keys should be returned

Since:

1.4

---

prepareStatement

```
public PreparedStatement prepareStatement(String sql,
                                          int[] columnIndexes)
                                   throws SQLException
```

Creates a default `PreparedStatement` object capable of returning the auto-generated keys designated by the given array. This array contains the indexes of the columns in the target table that contain the auto-generated keys that should be made available. This array is ignored if the SQL statement is not an `INSERT` statement.

An SQL statement with or without IN parameters can be pre-compiled and stored in a `PreparedStatement` object. This object can then be used to efficiently execute this statement multiple times.

Note: This method is optimized for handling parametric SQL statements that benefit from precompilation. If the driver supports precompilation, the method `prepareStatement` will send the statement to the database for precompilation. Some drivers may not support precompilation. In this case, the statement may not be sent to the database until the `PreparedStatement` object is executed. This has no direct effect on users; however, it does affect which methods throw certain SQLExceptions.

Result sets created using the returned `PreparedStatement` object will by default be type `TYPE_FORWARD_ONLY` and have a concurrency level of `CONCUR_READ_ONLY`.

Parameters:

`sql` - an SQL statement that may contain one or more '?' IN parameter placeholders
`columnIndexes` - an array of column indexes indicating the columns that should be returned from the inserted row or rows

Returns:

a new `PreparedStatement` object, containing the pre-compiled statement, that is capable of returning the auto-generated keys designated by the given array of column indexes

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

prepareStatement

```
public PreparedStatement prepareStatement(String sql,
                                          String[] columnNames)
                                   throws SQLException
```

Creates a default `PreparedStatement` object capable of returning the auto-generated keys designated by the given array. This array contains the names of the columns in the target table that contain the auto-generated keys that should be returned. This array is ignored if the SQL statement is not an `INSERT` statement.

An SQL statement with or without IN parameters can be pre-compiled and stored in a `PreparedStatement` object. This object can then be used to efficiently execute this statement multiple times.

Note: This method is optimized for handling parametric SQL statements that benefit from precompilation. If the driver supports precompilation, the method `prepareStatement` will send the statement to the database for precompilation. Some drivers may not support precompilation. In this case, the statement may not be sent to the database until the `PreparedStatement` object is executed. This has no direct effect on users; however, it does affect which methods throw certain SQLExceptions.

Result sets created using the returned `PreparedStatement` object will by default be type `TYPE_FORWARD_ONLY` and have a concurrency level of `CONCUR_READ_ONLY`.

Parameters:

`sql` - an SQL statement that may contain one or more '?' IN parameter placeholders
`columnNames` - an array of column names indicating the columns that should be returned from the inserted row or rows

Returns:

a new `PreparedStatement` object, containing the pre-compiled statement, that is capable of returning the auto-generated keys designated by the given array of column names

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

java.sql

Interface Statement

All Known Subinterfaces:

CallableStatement, PreparedStatement

public interface **Statement**

The object used for executing a static SQL statement and returning the results it produces.

By default, only one `ResultSet` object per `Statement` object can be open at the same time. Therefore, if the reading of one `ResultSet` object is interleaved with the reading of another, each must have been generated by different `Statement` objects. All execution methods in the `Statement` interface implicitly close a statment's current `ResultSet` object if an open one exists.

See Also:

`Connection.createStatement()`, `ResultSet`

Field Summary

`static int`	**`CLOSE_ALL_RESULTS`** The constant indicating that all `ResultSet` objects that have previously been kept open should be closed when calling `getMoreResults`.
`static int`	**`CLOSE_CURRENT_RESULT`** The constant indicating that the current `ResultSet` object should be closed when calling `getMoreResults`.
`static int`	**`EXECUTE_FAILED`** The constant indicating that an error occured while executing a batch statement.
`static int`	**`KEEP_CURRENT_RESULT`** The constant indicating that the current `ResultSet` object should not be closed when calling `getMoreResults`.

static int	**NO_GENERATED_KEYS** The constant indicating that generated keys should not be made available for retrieval.
static int	**RETURN_GENERATED_KEYS** The constant indicating that generated keys should be made available for retrieval.
static int	**SUCCESS_NO_INFO** The constant indicating that a batch statement executed successfully but that no count of the number of rows it affected is available.

Method Summary

void	**addBatch**(String sql) Adds the given SQL command to the current list of commmands for this Statement object.
void	**cancel**() Cancels this Statement object if both the DBMS and driver support aborting an SQL statement.
void	**clearBatch**() Empties this Statement object's current list of SQL commands.
void	**clearWarnings**() Clears all the warnings reported on this Statement object.
void	**close**() Releases this Statement object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.
boolean	**execute**(String sql) Executes the given SQL statement, which may return multiple results.
boolean	**execute**(String sql, int autoGeneratedKeys) Executes the given SQL statement, which may return multiple results, and signals the driver that any auto-generated keys should be made available for retrieval.
boolean	**execute**(String sql, int[] columnIndexes) Executes the given SQL statement, which may return multiple results, and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval.
boolean	**execute**(String sql, String[] columnNames) Executes the given SQL statement, which may return multiple results, and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval.

int[]	**executeBatch**() Submits a batch of commands to the database for execution and if all commands execute successfully, returns an array of update counts.
ResultSet	**executeQuery**(String sql) Executes the given SQL statement, which returns a single ResultSet object.
int	**executeUpdate**(String sql) Executes the given SQL statement, which may be an INSERT, UPDATE, or DELETE statement or an SQL statement that returns nothing, such as an SQL DDL statement.
int	**executeUpdate**(String sql, int autoGeneratedKeys) Executes the given SQL statement and signals the driver with the given flag about whether the auto-generated keys produced by this Statement object should be made available for retrieval.
int	**executeUpdate**(String sql, int[] columnIndexes) Executes the given SQL statement and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval.
int	**executeUpdate**(String sql, String[] columnNames) Executes the given SQL statement and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval.
Connection	**getConnection**() Retrieves the Connection object that produced this Statement object.
int	**getFetchDirection**() Retrieves the direction for fetching rows from database tables that is the default for result sets generated from this Statement object.
int	**getFetchSize**() Retrieves the number of result set rows that is the default fetch size for ResultSet objects generated from this Statement object.
ResultSet	**getGeneratedKeys**() Retrieves any auto-generated keys created as a result of executing this Statement object.
int	**getMaxFieldSize**() Retrieves the maximum number of bytes that can be returned for character and binary column values in a ResultSet object produced by this Statement object.
int	**getMaxRows**() Retrieves the maximum number of rows that a ResultSet object produced by this Statement object can contain.

boolean	**getMoreResults**() Moves to this Statement object's next result, returns true if it is a ResultSet object, and implicitly closes any current ResultSet object(s) obtained with the method getResultSet.
boolean	**getMoreResults**(int current) Moves to this Statement object's next result, deals with any current ResultSet object(s) according to the instructions specified by the given flag, and returns true if the next result is a ResultSet object.
int	**getQueryTimeout**() Retrieves the number of seconds the driver will wait for a Statement object to execute.
ResultSet	**getResultSet**() Retrieves the current result as a ResultSet object.
int	**getResultSetConcurrency**() Retrieves the result set concurrency for ResultSet objects generated by this Statement object.
int	**getResultSetHoldability**() Retrieves the result set holdability for ResultSet objects generated by this Statement object.
int	**getResultSetType**() Retrieves the result set type for ResultSet objects generated by this Statement object.
int	**getUpdateCount**() Retrieves the current result as an update count; if the result is a ResultSet object or there are no more results, -1 is returned.
SQLWarning	**getWarnings**() Retrieves the first warning reported by calls on this Statement object.
void	**setCursorName**(String name) Sets the SQL cursor name to the given String, which will be used by subsequent Statement object execute methods.
void	**setEscapeProcessing**(boolean enable) Sets escape processing on or off.
void	**setFetchDirection**(int direction) Gives the driver a hint as to the direction in which rows will be processed in ResultSet objects created using this Statement object.
void	**setFetchSize**(int rows) Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed.

void	**setMaxFieldSize**(int max) Sets the limit for the maximum number of bytes in a ResultSet column storing character or binary values to the given number of bytes.
void	**setMaxRows**(int max) Sets the limit for the maximum number of rows that any ResultSet object can contain to the given number.
void	**setQueryTimeout**(int seconds) Sets the number of seconds the driver will wait for a Statement object to execute to the given number of seconds.

Field Detail

CLOSE_CURRENT_RESULT

```
public static final int CLOSE_CURRENT_RESULT
```

The constant indicating that the current ResultSet object should be closed when calling getMoreResults.

Since:
1.4
See Also:
Constant Field Values

KEEP_CURRENT_RESULT

```
public static final int KEEP_CURRENT_RESULT
```

The constant indicating that the current ResultSet object should not be closed when calling getMoreResults.

Since:
1.4
See Also:
Constant Field Values

CLOSE_ALL_RESULTS

`public static final int` **`CLOSE_ALL_RESULTS`**

The constant indicating that all `ResultSet` objects that have previously been kept open should be closed when calling `getMoreResults`.

Since:
1.4
See Also:
Constant Field Values

SUCCESS_NO_INFO

`public static final int` **`SUCCESS_NO_INFO`**

The constant indicating that a batch statement executed successfully but that no count of the number of rows it affected is available.

Since:
1.4
See Also:
Constant Field Values

EXECUTE_FAILED

`public static final int` **`EXECUTE_FAILED`**

The constant indicating that an error occured while executing a batch statement.

Since:
1.4
See Also:
Constant Field Values

RETURN_GENERATED_KEYS

`public static final int` **`RETURN_GENERATED_KEYS`**

The constant indicating that generated keys should be made available for retrieval.

Since:
1.4
See Also:
Constant Field Values

NO_GENERATED_KEYS

`public static final int` **`NO_GENERATED_KEYS`**

The constant indicating that generated keys should not be made available for retrieval.

Since:
1.4
See Also:
Constant Field Values

Method Detail

executeQuery

```
public ResultSet executeQuery(String sql)
                       throws SQLException
```

Executes the given SQL statement, which returns a single `ResultSet` object.

Parameters:
`sql` - an SQL statement to be sent to the database, typically a static SQL `SELECT` statement
Returns:
a `ResultSet` object that contains the data produced by the given query; never `null`
Throws:
`SQLException` - if a database access error occurs or the given SQL statement produces anything other than a single `ResultSet` object

---

executeUpdate

```
public int executeUpdate(String sql)
                  throws SQLException
```

Executes the given SQL statement, which may be an `INSERT`, `UPDATE`, or `DELETE` statement or an SQL statement that returns nothing, such as an SQL DDL statement.

Parameters:

`sql` - an SQL `INSERT`, `UPDATE` or `DELETE` statement or an SQL statement that returns nothing

Returns:

either the row count for `INSERT`, `UPDATE` or `DELETE` statements, or `0` for SQL statements that return nothing

Throws:

`SQLException` - if a database access error occurs or the given SQL statement produces a `ResultSet` object

---

close

```
public void close()
           throws SQLException
```

Releases this `Statement` object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed. It is generally good practice to release resources as soon as you are finished with them to avoid tying up database resources.

Calling the method `close` on a `Statement` object that is already closed has no effect.

Note: A `Statement` object is automatically closed when it is garbage collected. When a `Statement` object is closed, its current `ResultSet` object, if one exists, is also closed.

Throws:

`SQLException` - if a database access error occurs

---

getMaxFieldSize

```
public int getMaxFieldSize()
                   throws SQLException
```

Retrieves the maximum number of bytes that can be returned for character and binary column values in a `ResultSet` object produced by this `Statement` object. This limit applies only to `BINARY`, `VARBINARY`, `LONGVARBINARY`, `CHAR`, `VARCHAR`, and `LONGVARCHAR` columns. If the limit is exceeded, the excess data is silently discarded.

Returns:

the current column size limit for columns storing character and binary values; zero means there is no limit

Throws:

`SQLException` - if a database access error occurs

See Also:

`setMaxFieldSize(int)`

---

setMaxFieldSize

```
public void setMaxFieldSize(int max)
                    throws SQLException
```

Sets the limit for the maximum number of bytes in a `ResultSet` column storing character or binary values to the given number of bytes. This limit applies only to `BINARY`, `VARBINARY`, `LONGVARBINARY`, `CHAR`, `VARCHAR`, and `LONGVARCHAR` fields. If the limit is exceeded, the excess data is silently discarded. For maximum portability, use values greater than 256.

Parameters:

`max` - the new column size limit in bytes; zero means there is no limit

Throws:

`SQLException` - if a database access error occurs or the condition max >= 0 is not satisfied

See Also:

`getMaxFieldSize()`

---

getMaxRows

```
public int getMaxRows()
                throws SQLException
```

Retrieves the maximum number of rows that a `ResultSet` object produced by this `Statement` object can contain. If this limit is exceeded, the excess rows are silently dropped.

Returns:

the current maximum number of rows for a `ResultSet` object produced by this `Statement` object; zero means there is no limit

Throws:

`SQLException` - if a database access error occurs

See Also:

`setMaxRows(int)`

---

setMaxRows

```
public void setMaxRows(int max)
                throws SQLException
```

Sets the limit for the maximum number of rows that any `ResultSet` object can contain to the given number. If the limit is exceeded, the excess rows are silently dropped.

Parameters:

`max` - the new max rows limit; zero means there is no limit

Throws:

`SQLException` - if a database access error occurs or the condition max >= 0 is not satisfied

See Also:

`getMaxRows()`

---

setEscapeProcessing

```
public void setEscapeProcessing(boolean enable)
                         throws SQLException
```

Sets escape processing on or off. If escape scanning is on (the default), the driver will do escape substitution before sending the SQL statement to the database. Note: Since prepared statements have usually been parsed prior to making this call, disabling escape processing for `PreparedStatements` objects will have no effect.

Parameters:

`enable` - `true` to enable escape processing; `false` to disable it

Throws:

`SQLException` - if a database access error occurs

---

getQueryTimeout

```
public int getQueryTimeout()
                    throws SQLException
```

Retrieves the number of seconds the driver will wait for a `Statement` object to execute. If the limit is exceeded, a `SQLException` is thrown.

Returns:
the current query timeout limit in seconds; zero means there is no limit
Throws:
`SQLException` - if a database access error occurs
See Also:
`setQueryTimeout(int)`

---

setQueryTimeout

```
public void setQueryTimeout(int seconds)
                     throws SQLException
```

Sets the number of seconds the driver will wait for a `Statement` object to execute to the given number of seconds. If the limit is exceeded, an `SQLException` is thrown.

Parameters:
`seconds` - the new query timeout limit in seconds; zero means there is no limit
Throws:
`SQLException` - if a database access error occurs or the condition seconds >= 0 is not satisfied
See Also:
`getQueryTimeout()`

---

cancel

```
public void cancel()
            throws SQLException
```

Cancels this `Statement` object if both the DBMS and driver support aborting an SQL statement. This method can be used by one thread to cancel a statement that is being executed

by another thread.

Throws:

SQLException - if a database access error occurs

---

getWarnings

```
public SQLWarning getWarnings()
                        throws SQLException
```

Retrieves the first warning reported by calls on this `Statement` object. Subsequent `Statement` object warnings will be chained to this `SQLWarning` object.

The warning chain is automatically cleared each time a statement is (re)executed. This method may not be called on a closed `Statement` object; doing so will cause an `SQLException` to be thrown.

Note: If you are processing a `ResultSet` object, any warnings associated with reads on that `ResultSet` object will be chained on it rather than on the `Statement` object that produced it.

Returns:

the first `SQLWarning` object or `null` if there are no warnings

Throws:

SQLException - if a database access error occurs or this method is called on a closed statement

---

clearWarnings

```
public void clearWarnings()
                    throws SQLException
```

Clears all the warnings reported on this `Statement` object. After a call to this method, the method `getWarnings` will return `null` until a new warning is reported for this `Statement` object.

Throws:

SQLException - if a database access error occurs

---

setCursorName

```
public void setCursorName(String name)
                   throws SQLException
```

Sets the SQL cursor name to the given `String`, which will be used by subsequent `Statement` object `execute` methods. This name can then be used in SQL positioned update or delete statements to identify the current row in the `ResultSet` object generated by this statement. If the database does not support positioned update/delete, this method is a noop. To insure that a cursor has the proper isolation level to support updates, the cursor's `SELECT` statement should have the form `SELECT FOR UPDATE`. If `FOR UPDATE` is not present, positioned updates may fail.

Note: By definition, the execution of positioned updates and deletes must be done by a different `Statement` object than the one that generated the `ResultSet` object being used for positioning. Also, cursor names must be unique within a connection.

Parameters:

`name` - the new cursor name, which must be unique within a connection

Throws:

`SQLException` - if a database access error occurs

---

execute

```
public boolean execute(String sql)
                throws SQLException
```

Executes the given SQL statement, which may return multiple results. In some (uncommon) situations, a single SQL statement may return multiple result sets and/or update counts. Normally you can ignore this unless you are (1) executing a stored procedure that you know may return multiple results or (2) you are dynamically executing an unknown SQL string.

The `execute` method executes an SQL statement and indicates the form of the first result. You must then use the methods `getResultSet` or `getUpdateCount` to retrieve the result, and `getMoreResults` to move to any subsequent result(s).

Parameters:

`sql` - any SQL statement

Returns:

`true` if the first result is a `ResultSet` object; `false` if it is an update count or there

are no results

Throws:

SQLException - if a database access error occurs

See Also:

getResultSet(), getUpdateCount(), getMoreResults()

---

getResultSet

```
public ResultSet getResultSet()
                        throws SQLException
```

Retrieves the current result as a `ResultSet` object. This method should be called only once per result.

Returns:

the current result as a `ResultSet` object or `null` if the result is an update count or there are no more results

Throws:

SQLException - if a database access error occurs

See Also:

execute(java.lang.String)

---

getUpdateCount

```
public int getUpdateCount()
                    throws SQLException
```

Retrieves the current result as an update count; if the result is a `ResultSet` object or there are no more results, -1 is returned. This method should be called only once per result.

Returns:

the current result as an update count; -1 if the current result is a `ResultSet` object or there are no more results

Throws:

SQLException - if a database access error occurs

See Also:

execute(java.lang.String)

---

getMoreResults

```
public boolean getMoreResults()
                        throws SQLException
```

Moves to this `Statement` object's next result, returns `true` if it is a `ResultSet` object, and implicitly closes any current `ResultSet` object(s) obtained with the method `getResultSet`.

There are no more results when the following is true:

```
     // stmt is a Statement object
     ((stmt.getMoreResults() == false) && (stmt.getUpdateCount
() == -1))
```

Returns:

`true` if the next result is a `ResultSet` object; `false` if it is an update count or there are no more results

Throws:

`SQLException` - if a database access error occurs

See Also:

`execute(java.lang.String)`

---

setFetchDirection

```
public void setFetchDirection(int direction)
                        throws SQLException
```

Gives the driver a hint as to the direction in which rows will be processed in `ResultSet` objects created using this `Statement` object. The default value is `ResultSet.FETCH_FORWARD`.

Note that this method sets the default fetch direction for result sets generated by this `Statement` object. Each result set has its own methods for getting and setting its own fetch direction.

Parameters:

`direction` - the initial direction for processing rows

Throws:

`SQLException` - if a database access error occurs or the given direction is not one of `ResultSet.FETCH_FORWARD`, `ResultSet.FETCH_REVERSE`, or

`ResultSet.FETCH_UNKNOWN`

Since:

1.2

See Also:

`getFetchDirection()`

---

getFetchDirection

```
public int getFetchDirection()
                    throws SQLException
```

Retrieves the direction for fetching rows from database tables that is the default for result sets generated from this `Statement` object. If this `Statement` object has not set a fetch direction by calling the method `setFetchDirection`, the return value is implementation-specific.

Returns:

the default fetch direction for result sets generated from this `Statement` object

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

See Also:

`setFetchDirection(int)`

---

setFetchSize

```
public void setFetchSize(int rows)
                  throws SQLException
```

Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed. The number of rows specified affects only result sets created using this statement. If the value specified is zero, then the hint is ignored. The default value is zero.

Parameters:

`rows` - the number of rows to fetch

Throws:

`SQLException` - if a database access error occurs, or the condition 0 <= `rows` <= `this.getMaxRows()` is not satisfied.

Since:

1.2

See Also:

`getFetchSize()`

---

getFetchSize

```
public int getFetchSize()
                  throws SQLException
```

Retrieves the number of result set rows that is the default fetch size for `ResultSet` objects generated from this `Statement` object. If this `Statement` object has not set a fetch size by calling the method `setFetchSize`, the return value is implementation-specific.

Returns:

the default fetch size for result sets generated from this `Statement` object

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

See Also:

`setFetchSize(int)`

---

getResultSetConcurrency

```
public int getResultSetConcurrency()
                             throws SQLException
```

Retrieves the result set concurrency for `ResultSet` objects generated by this `Statement` object.

Returns:

either `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getResultSetType

```
public int getResultSetType()
                    throws SQLException
```

Retrieves the result set type for `ResultSet` objects generated by this `Statement` object.

Returns:
one of `ResultSet.TYPE_FORWARD_ONLY`, `ResultSet.TYPE_SCROLL_INSENSITIVE`, or `ResultSet.TYPE_SCROLL_SENSITIVE`

Throws:
`SQLException` - if a database access error occurs

Since:
1.2

---

addBatch

```
public void addBatch(String sql)
              throws SQLException
```

Adds the given SQL command to the current list of commmands for this `Statement` object. The commands in this list can be executed as a batch by calling the method `executeBatch`.

NOTE: This method is optional.

Parameters:
`sql` - typically this is a static SQL `INSERT` or `UPDATE` statement

Throws:
`SQLException` - if a database access error occurs, or the driver does not support batch updates

Since:
1.2

See Also:
`executeBatch()`

---

clearBatch

```
public void clearBatch()
                throws SQLException
```

Empties this `Statement` object's current list of SQL commands.

NOTE: This method is optional.

Throws:

`SQLException` - if a database access error occurs or the driver does not support batch updates

Since:

1.2

See Also:

`addBatch(java.lang.String)`

---

executeBatch

```
public int[] executeBatch()
                    throws SQLException
```

Submits a batch of commands to the database for execution and if all commands execute successfully, returns an array of update counts. The `int` elements of the array that is returned are ordered to correspond to the commands in the batch, which are ordered according to the order in which they were added to the batch. The elements in the array returned by the method `executeBatch` may be one of the following:

1. A number greater than or equal to zero -- indicates that the command was processed successfully and is an update count giving the number of rows in the database that were affected by the command's execution
2. A value of `SUCCESS_NO_INFO` -- indicates that the command was processed successfully but that the number of rows affected is unknown

   If one of the commands in a batch update fails to execute properly, this method throws a `BatchUpdateException`, and a JDBC driver may or may not continue to process the remaining commands in the batch. However, the driver's behavior must be consistent with a particular DBMS, either always continuing to process commands or never continuing to process commands. If the driver continues processing after a failure, the array returned by the method `BatchUpdateException.getUpdateCounts` will contain as many elements as there are commands in the batch, and at least one of the elements will be the following:

3. A value of `EXECUTE_FAILED` -- indicates that the command failed to execute successfully and occurs only if a driver continues to process commands after a command fails

A driver is not required to implement this method. The possible implementations and return

values have been modified in the Java 2 SDK, Standard Edition, version 1.3 to accommodate the option of continuing to proccess commands in a batch update after a `BatchUpdateException` obejct has been thrown.

Returns:

an array of update counts containing one element for each command in the batch. The elements of the array are ordered according to the order in which commands were added to the batch.

Throws:

`SQLException` - if a database access error occurs or the driver does not support batch statements. Throws `BatchUpdateException` (a subclass of `SQLException`) if one of the commands sent to the database fails to execute properly or attempts to return a result set.

Since:

1.3

---

getConnection

```
public Connection getConnection()
                           throws SQLException
```

Retrieves the `Connection` object that produced this `Statement` object.

Returns:

the connection that produced this statement

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getMoreResults

```
public boolean getMoreResults(int current)
                       throws SQLException
```

Moves to this `Statement` object's next result, deals with any current `ResultSet` object(s) according to the instructions specified by the given flag, and returns `true` if the next result is a `ResultSet` object.

There are no more results when the following is true:

```
        // stmt is a Statement object
        ((stmt.getMoreResults() == false) && (stmt.getUpdateCount
    () == -1))
```

Parameters:

`current` - one of the following `Statement` constants indicating what should happen to current `ResultSet` objects obtained using the method `getResultSet`: `Statement.CLOSE_CURRENT_RESULT`, `Statement.KEEP_CURRENT_RESULT`, or `Statement.CLOSE_ALL_RESULTS`

Returns:

`true` if the next result is a `ResultSet` object; `false` if it is an update count or there are no more results

Throws:

`SQLException` - if a database access error occurs or the argument supplied is not one of the following: `Statement.CLOSE_CURRENT_RESULT`, `Statement.KEEP_CURRENT_RESULT`, or `Statement.CLOSE_ALL_RESULTS`

Since:

1.4

See Also:

`execute(java.lang.String)`

---

getGeneratedKeys

```
public ResultSet getGeneratedKeys()
                            throws SQLException
```

Retrieves any auto-generated keys created as a result of executing this `Statement` object. If this `Statement` object did not generate any keys, an empty `ResultSet` object is returned.

Returns:

a `ResultSet` object containing the auto-generated key(s) generated by the execution of this `Statement` object

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

executeUpdate

```
public int executeUpdate(String sql,
                         int autoGeneratedKeys)
                  throws SQLException
```

Executes the given SQL statement and signals the driver with the given flag about whether the auto-generated keys produced by this `Statement` object should be made available for retrieval.

Parameters:

`sql` - must be an SQL `INSERT`, `UPDATE` or `DELETE` statement or an SQL statement that returns nothing

`autoGeneratedKeys` - a flag indicating whether auto-generated keys should be made available for retrieval; one of the following constants: `Statement.RETURN_GENERATED_KEYS` `Statement.NO_GENERATED_KEYS`

Returns:

either the row count for `INSERT`, `UPDATE` or `DELETE` statements, or `0` for SQL statements that return nothing

Throws:

`SQLException` - if a database access error occurs, the given SQL statement returns a `ResultSet` object, or the given constant is not one of those allowed

Since:

1.4

---

executeUpdate

```
public int executeUpdate(String sql,
                         int[] columnIndexes)
                  throws SQLException
```

Executes the given SQL statement and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval. The driver will ignore the array if the SQL statement is not an `INSERT` statement.

Parameters:

`sql` - an SQL `INSERT`, `UPDATE` or `DELETE` statement or an SQL statement that returns nothing, such as an SQL DDL statement

`columnIndexes` - an array of column indexes indicating the columns that should be returned from the inserted row

Returns:

either the row count for `INSERT`, `UPDATE`, or `DELETE` statements, or 0 for SQL statements that return nothing

Throws:

SQLException - if a database access error occurs, the SQL statement returns a ResultSet object, or the second argument supplied to this method is not an int array whose elements are valid column indexes

Since:

1.4

---

executeUpdate

```
public int executeUpdate(String sql,
                         String[] columnNames)
                  throws SQLException
```

Executes the given SQL statement and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval. The driver will ignore the array if the SQL statement is not an INSERT statement.

Parameters:

sql - an SQL INSERT, UPDATE or DELETE statement or an SQL statement that returns nothing

columnNames - an array of the names of the columns that should be returned from the inserted row

Returns:

either the row count for INSERT, UPDATE, or DELETE statements, or 0 for SQL statements that return nothing

Throws:

SQLException - if a database access error occurs, the SQL statement returns a ResultSet object, or the second argument supplied to this method is not a String array whose elements are valid column names

Since:

1.4

---

execute

```
public boolean execute(String sql,
                       int autoGeneratedKeys)
                throws SQLException
```

Executes the given SQL statement, which may return multiple results, and signals the driver that any auto-generated keys should be made available for retrieval. The driver will ignore this signal if the SQL statement is not an INSERT statement.

In some (uncommon) situations, a single SQL statement may return multiple result sets and/or update counts. Normally you can ignore this unless you are (1) executing a stored procedure that you know may return multiple results or (2) you are dynamically executing an unknown SQL string.

The `execute` method executes an SQL statement and indicates the form of the first result. You must then use the methods `getResultSet` or `getUpdateCount` to retrieve the result, and `getMoreResults` to move to any subsequent result(s).

Parameters:

`sql` - any SQL statement

`autoGeneratedKeys` - a constant indicating whether auto-generated keys should be made available for retrieval using the method `getGeneratedKeys`; one of the following constants: `Statement.RETURN_GENERATED_KEYS` or `Statement.NO_GENERATED_KEYS`

Returns:

`true` if the first result is a `ResultSet` object; `false` if it is an update count or there are no results

Throws:

`SQLException` - if a database access error occurs or the second parameter supplied to this method is not `Statement.RETURN_GENERATED_KEYS` or `Statement.NO_GENERATED_KEYS`.

Since:

1.4

See Also:

`getResultSet()`, `getUpdateCount()`, `getMoreResults()`, `getGeneratedKeys()`

---

execute

```
public boolean execute(String sql,
                       int[] columnIndexes)
                throws SQLException
```

Executes the given SQL statement, which may return multiple results, and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval. This array contains the indexes of the columns in the target table that contain the auto-generated keys that should be made available. The driver will ignore the array if the given SQL statement is not an `INSERT` statement.

Under some (uncommon) situations, a single SQL statement may return multiple result sets

and/or update counts. Normally you can ignore this unless you are (1) executing a stored procedure that you know may return multiple results or (2) you are dynamically executing an unknown SQL string.

The `execute` method executes an SQL statement and indicates the form of the first result. You must then use the methods `getResultSet` or `getUpdateCount` to retrieve the result, and `getMoreResults` to move to any subsequent result(s).

Parameters:

`sql` - any SQL statement

`columnIndexes` - an array of the indexes of the columns in the inserted row that should be made available for retrieval by a call to the method `getGeneratedKeys`

Returns:

`true` if the first result is a `ResultSet` object; `false` if it is an update count or there are no results

Throws:

`SQLException` - if a database access error occurs or the elements in the `int` array passed to this method are not valid column indexes

Since:

1.4

See Also:

`getResultSet()`, `getUpdateCount()`, `getMoreResults()`

---

execute

```
public boolean execute(String sql,
                       String[] columnNames)
                throws SQLException
```

Executes the given SQL statement, which may return multiple results, and signals the driver that the auto-generated keys indicated in the given array should be made available for retrieval. This array contains the names of the columns in the target table that contain the auto-generated keys that should be made available. The driver will ignore the array if the given SQL statement is not an `INSERT` statement.

In some (uncommon) situations, a single SQL statement may return multiple result sets and/or update counts. Normally you can ignore this unless you are (1) executing a stored procedure that you know may return multiple results or (2) you are dynamically executing an unknown SQL string.

The `execute` method executes an SQL statement and indicates the form of the first result. You must then use the methods `getResultSet` or `getUpdateCount` to retrieve the

result, and `getMoreResults` to move to any subsequent result(s).

Parameters:

`sql` - any SQL statement

`columnNames` - an array of the names of the columns in the inserted row that should be made available for retrieval by a call to the method `getGeneratedKeys`

Returns:

`true` if the next result is a `ResultSet` object; `false` if it is an update count or there are no more results

Throws:

`SQLException` - if a database access error occurs or the elements of the `String` array passed to this method are not valid column names

Since:

1.4

See Also:

`getResultSet()`, `getUpdateCount()`, `getMoreResults()`, `getGeneratedKeys()`

---

getResultSetHoldability

```
public int getResultSetHoldability()
                            throws SQLException
```

Retrieves the result set holdability for `ResultSet` objects generated by this `Statement` object.

Returns:

either `ResultSet.HOLD_CURSORS_OVER_COMMIT` or `ResultSet.CLOSE_CURSORS_AT_COMMIT`

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

definitions of terms, workarounds, and working code examples.

---

java.sql

Interface PreparedStatement

All Superinterfaces:

Statement

All Known Subinterfaces:

CallableStatement

---

public interface **PreparedStatement**
extends Statement

An object that represents a precompiled SQL statement.

A SQL statement is precompiled and stored in a `PreparedStatement` object. This object can then be used to efficiently execute this statement multiple times.

Note: The setter methods (`setShort`, `setString`, and so on) for setting IN parameter values must specify types that are compatible with the defined SQL type of the input parameter. For instance, if the IN parameter has SQL type `INTEGER`, then the method `setInt` should be used.

If arbitrary parameter type conversions are required, the method `setObject` should be used with a target SQL type.

In the following example of setting a parameter, `con` represents an active connection:

```
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES
                                  SET SALARY = ? WHERE ID = ?");
pstmt.setBigDecimal(1, 153833.00)
pstmt.setInt(2, 110592)
```

See Also:

`Connection.prepareStatement(java.lang.String)`, `ResultSet`

Field Summary

Fields inherited from interface java.sql.Statement

CLOSE_ALL_RESULTS, CLOSE_CURRENT_RESULT, EXECUTE_FAILED, KEEP_CURRENT_RESULT, NO_GENERATED_KEYS, RETURN_GENERATED_KEYS, SUCCESS_NO_INFO

Method Summary

void	**addBatch**() Adds a set of parameters to this PreparedStatement object's batch of commands.
void	**clearParameters**() Clears the current parameter values immediately.
boolean	**execute**() Executes the SQL statement in this PreparedStatement object, which may be any kind of SQL statement.
ResultSet	**executeQuery**() Executes the SQL query in this PreparedStatement object and returns the ResultSet object generated by the query.
int	**executeUpdate**() Executes the SQL statement in this PreparedStatement object, which must be an SQL INSERT, UPDATE or DELETE statement; or an SQL statement that returns nothing, such as a DDL statement.
ResultSetMetaData	**getMetaData**() Retrieves a ResultSetMetaData object that contains information about the columns of the ResultSet object that will be returned when this PreparedStatement object is executed.
ParameterMetaData	**getParameterMetaData**() Retrieves the number, types and properties of this PreparedStatement object's parameters.
void	**setArray**(int i, Array x) Sets the designated parameter to the given Array object.
void	**setAsciiStream**(int parameterIndex, InputStream x, int length) Sets the designated parameter to the given input stream, which will have the specified number of bytes.

void	**setBigDecimal**(int parameterIndex, BigDecimal x) Sets the designated parameter to the given java.math.BigDecimal value.
void	**setBinaryStream**(int parameterIndex, InputStream x, int length) Sets the designated parameter to the given input stream, which will have the specified number of bytes.
void	**setBlob**(int i, Blob x) Sets the designated parameter to the given Blob object.
void	**setBoolean**(int parameterIndex, boolean x) Sets the designated parameter to the given Java boolean value.
void	**setByte**(int parameterIndex, byte x) Sets the designated parameter to the given Java byte value.
void	**setBytes**(int parameterIndex, byte[] x) Sets the designated parameter to the given Java array of bytes.
void	**setCharacterStream**(int parameterIndex, Reader reader, int length) Sets the designated parameter to the given Reader object, which is the given number of characters long.
void	**setClob**(int i, Clob x) Sets the designated parameter to the given Clob object.
void	**setDate**(int parameterIndex, Date x) Sets the designated parameter to the given java.sql.Date value.
void	**setDate**(int parameterIndex, Date x, Calendar cal) Sets the designated parameter to the given java.sql.Date value, using the given Calendar object.
void	**setDouble**(int parameterIndex, double x) Sets the designated parameter to the given Java double value.
void	**setFloat**(int parameterIndex, float x) Sets the designated parameter to the given Java float value.
void	**setInt**(int parameterIndex, int x) Sets the designated parameter to the given Java int value.
void	**setLong**(int parameterIndex, long x) Sets the designated parameter to the given Java long value.
void	**setNull**(int parameterIndex, int sqlType) Sets the designated parameter to SQL NULL.

void	**setNull**(int paramIndex, int sqlType, String typeName) Sets the designated parameter to SQL NULL.
void	**setObject**(int parameterIndex, Object x) Sets the value of the designated parameter using the given object.
void	**setObject**(int parameterIndex, Object x, int targetSqlType) Sets the value of the designated parameter with the given object.
void	**setObject**(int parameterIndex, Object x, int targetSqlType, int scale) Sets the value of the designated parameter with the given object.
void	**setRef**(int i, Ref x) Sets the designated parameter to the given REF(<structured-type>) value.
void	**setShort**(int parameterIndex, short x) Sets the designated parameter to the given Java short value.
void	**setString**(int parameterIndex, String x) Sets the designated parameter to the given Java String value.
void	**setTime**(int parameterIndex, Time x) Sets the designated parameter to the given java.sql.Time value.
void	**setTime**(int parameterIndex, Time x, Calendar cal) Sets the designated parameter to the given java.sql.Time value, using the given Calendar object.
void	**setTimestamp**(int parameterIndex, Timestamp x) Sets the designated parameter to the given java.sql.Timestamp value.
void	**setTimestamp**(int parameterIndex, Timestamp x, Calendar cal) Sets the designated parameter to the given java.sql.Timestamp value, using the given Calendar object.
void	**setUnicodeStream**(int parameterIndex, InputStream x, int length) **Deprecated.**
void	**setURL**(int parameterIndex, URL x) Sets the designated parameter to the given java.net.URL value.

Methods inherited from interface java.sql.Statement
addBatch, cancel, clearBatch, clearWarnings, close, execute, execute, execute, execute, executeBatch, executeQuery, executeUpdate, executeUpdate, executeUpdate, executeUpdate, getConnection, getFetchDirection, getFetchSize, getGeneratedKeys, getMaxFieldSize, getMaxRows, getMoreResults, getMoreResults, getQueryTimeout, getResultSet, getResultSetConcurrency, getResultSetHoldability, getResultSetType, getUpdateCount, getWarnings, setCursorName, setEscapeProcessing, setFetchDirection, setFetchSize, setMaxFieldSize, setMaxRows, setQueryTimeout

Method Detail

executeQuery

```
public ResultSet executeQuery()
                        throws SQLException
```

Executes the SQL query in this `PreparedStatement` object and returns the `ResultSet` object generated by the query.

Returns:

a `ResultSet` object that contains the data produced by the query; never `null`

Throws:

`SQLException` - if a database access error occurs or the SQL statement does not return a `ResultSet` object

---

executeUpdate

```
public int executeUpdate()
                    throws SQLException
```

Executes the SQL statement in this `PreparedStatement` object, which must be an SQL `INSERT`, `UPDATE` or `DELETE` statement; or an SQL statement that returns nothing, such as a DDL statement.

Returns:

either (1) the row count for `INSERT`, `UPDATE`, or `DELETE` statements or (2) 0 for SQL statements that return nothing

Throws:

`SQLException` - if a database access error occurs or the SQL statement returns a `ResultSet` object

---

setNull

```
public void setNull(int parameterIndex,
                    int sqlType)
             throws SQLException
```

Sets the designated parameter to SQL `NULL`.

Note: You must specify the parameter's SQL type.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`sqlType` - the SQL type code defined in `java.sql.Types`

Throws:

`SQLException` - if a database access error occurs

---

setBoolean

```
public void setBoolean(int parameterIndex,
                       boolean x)
                throws SQLException
```

Sets the designated parameter to the given Java `boolean` value. The driver converts this to an SQL `BIT` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setByte

```
public void setByte(int parameterIndex,
                    byte x)
             throws SQLException
```

Sets the designated parameter to the given Java `byte` value. The driver converts this to an SQL `TINYINT` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setShort

```
public void setShort(int parameterIndex,
                     short x)
              throws SQLException
```

Sets the designated parameter to the given Java `short` value. The driver converts this to an SQL `SMALLINT` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setInt

```
public void setInt(int parameterIndex,
                   int x)
            throws SQLException
```

Sets the designated parameter to the given Java `int` value. The driver converts this to an SQL `INTEGER` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...

`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setLong

```
public void setLong(int parameterIndex,
                    long x)
             throws SQLException
```

Sets the designated parameter to the given Java `long` value. The driver converts this to an SQL `BIGINT` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...

`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setFloat

```
public void setFloat(int parameterIndex,
                     float x)
              throws SQLException
```

Sets the designated parameter to the given Java `float` value. The driver converts this to an SQL `FLOAT` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...

`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setDouble

```
public void setDouble(int parameterIndex,
                      double x)
               throws SQLException
```

Sets the designated parameter to the given Java `double` value. The driver converts this to an SQL `DOUBLE` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setBigDecimal

```
public void setBigDecimal(int parameterIndex,
                          BigDecimal x)
                   throws SQLException
```

Sets the designated parameter to the given `java.math.BigDecimal` value. The driver converts this to an SQL `NUMERIC` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setString

```
public void setString(int parameterIndex,
                      String x)
               throws SQLException
```

Sets the designated parameter to the given Java `String` value. The driver converts this to an SQL `VARCHAR` or `LONGVARCHAR` value (depending on the argument's size relative to the driver's limits on `VARCHAR` values) when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setBytes

```
public void setBytes(int parameterIndex,
                     byte[] x)
              throws SQLException
```

Sets the designated parameter to the given Java array of bytes. The driver converts this to an SQL `VARBINARY` or `LONGVARBINARY` (depending on the argument's size relative to the driver's limits on `VARBINARY` values) when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setDate

```
public void setDate(int parameterIndex,
                    Date x)
             throws SQLException
```

Sets the designated parameter to the given `java.sql.Date` value. The driver converts this to an SQL `DATE` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setTime

```
public void setTime(int parameterIndex,
                    Time x)
             throws SQLException
```

Sets the designated parameter to the given `java.sql.Time` value. The driver converts this to an SQL `TIME` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setTimestamp

```
public void setTimestamp(int parameterIndex,
                         Timestamp x)
                  throws SQLException
```

Sets the designated parameter to the given `java.sql.Timestamp` value. The driver converts this to an SQL `TIMESTAMP` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value

Throws:

`SQLException` - if a database access error occurs

---

setAsciiStream

```
public void setAsciiStream(int parameterIndex,
                           InputStream x,
                           int length)
                    throws SQLException
```

Sets the designated parameter to the given input stream, which will have the specified number of bytes. When a very large ASCII value is input to a `LONGVARCHAR` parameter, it may be more practical to send it via a `java.io.InputStream`. Data will be read from the stream as needed until end-of-file is reached. The JDBC driver will do any necessary conversion from

ASCII to the database char format.

Note: This stream object can either be a standard Java stream object or your own subclass that implements the standard interface.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the Java input stream that contains the ASCII parameter value
`length` - the number of bytes in the stream

Throws:

`SQLException` - if a database access error occurs

---

setUnicodeStream

```
public void setUnicodeStream(int parameterIndex,
                             InputStream x,
                             int length)
                      throws SQLException
```

Deprecated.

Sets the designated parameter to the given input stream, which will have the specified number of bytes. A Unicode character has two bytes, with the first byte being the high byte, and the second being the low byte. When a very large Unicode value is input to a `LONGVARCHAR` parameter, it may be more practical to send it via a `java.io.InputStream` object. The data will be read from the stream as needed until end-of-file is reached. The JDBC driver will do any necessary conversion from Unicode to the database char format.

Note: This stream object can either be a standard Java stream object or your own subclass that implements the standard interface.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - a `java.io.InputStream` object that contains the Unicode parameter value as two-byte Unicode characters
`length` - the number of bytes in the stream

Throws:

`SQLException` - if a database access error occurs

---

setBinaryStream

```
public void setBinaryStream(int parameterIndex,
                            InputStream x,
                            int length)
                     throws SQLException
```

Sets the designated parameter to the given input stream, which will have the specified number of bytes. When a very large binary value is input to a `LONGVARBINARY` parameter, it may be more practical to send it via a `java.io.InputStream` object. The data will be read from the stream as needed until end-of-file is reached.

Note: This stream object can either be a standard Java stream object or your own subclass that implements the standard interface.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the java input stream which contains the binary parameter value
`length` - the number of bytes in the stream

Throws:

`SQLException` - if a database access error occurs

---

clearParameters

```
public void clearParameters()
                     throws SQLException
```

Clears the current parameter values immediately.

In general, parameter values remain in force for repeated use of a statement. Setting a parameter value automatically clears its previous value. However, in some cases it is useful to immediately release the resources used by the current parameter values; this can be done by calling the method `clearParameters`.

Throws:

`SQLException` - if a database access error occurs

---

setObject

```
public void setObject(int parameterIndex,
                      Object x,
```

```
                int targetSqlType,
                int scale)
         throws SQLException
```

Sets the value of the designated parameter with the given object. The second argument must be an object type; for integral values, the `java.lang` equivalent objects should be used.

The given Java object will be converted to the given targetSqlType before being sent to the database. If the object has a custom mapping (is of a class implementing the interface `SQLData`), the JDBC driver should call the method `SQLData.writeSQL` to write it to the SQL data stream. If, on the other hand, the object is of a class implementing `Ref`, `Blob`, `Clob`, `Struct`, or `Array`, the driver should pass it to the database as a value of the corresponding SQL type.

Note that this method may be used to pass database-specific abstract data types.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the object containing the input parameter value
`targetSqlType` - the SQL type (as defined in java.sql.Types) to be sent to the database. The scale argument may further qualify this type.
`scale` - for java.sql.Types.DECIMAL or java.sql.Types.NUMERIC types, this is the number of digits after the decimal point. For all other types, this value will be ignored.

Throws:

`SQLException` - if a database access error occurs

See Also:

`Types`

---

setObject

```
public void setObject(int parameterIndex,
                      Object x,
                      int targetSqlType)
               throws SQLException
```

Sets the value of the designated parameter with the given object. This method is like the method `setObject` above, except that it assumes a scale of zero.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the object containing the input parameter value
`targetSqlType` - the SQL type (as defined in java.sql.Types) to be sent to the

database

Throws:

SQLException - if a database access error occurs

---

setObject

```
public void setObject(int parameterIndex,
                      Object x)
               throws SQLException
```

Sets the value of the designated parameter using the given object. The second parameter must be of type `Object`; therefore, the `java.lang` equivalent objects should be used for built-in types.

The JDBC specification specifies a standard mapping from Java `Object` types to SQL types. The given argument will be converted to the corresponding SQL type before being sent to the database.

Note that this method may be used to pass datatabase- specific abstract data types, by using a driver-specific Java type. If the object is of a class implementing the interface `SQLData`, the JDBC driver should call the method `SQLData.writeSQL` to write it to the SQL data stream. If, on the other hand, the object is of a class implementing `Ref`, `Blob`, `Clob`, `Struct`, or `Array`, the driver should pass it to the database as a value of the corresponding SQL type.

This method throws an exception if there is an ambiguity, for example, if the object is of a class implementing more than one of the interfaces named above.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...

`x` - the object containing the input parameter value

Throws:

SQLException - if a database access error occurs or the type of the given object is ambiguous

---

execute

```
public boolean execute()
                throws SQLException
```

Executes the SQL statement in this `PreparedStatement` object, which may be any kind of SQL statement. Some prepared statements return multiple results; the `execute` method handles these complex statements as well as the simpler form of statements handled by the methods `executeQuery` and `executeUpdate`.

The `execute` method returns a `boolean` to indicate the form of the first result. You must call either the method `getResultSet` or `getUpdateCount` to retrieve the result; you must call `getMoreResults` to move to any subsequent result(s).

Returns:

`true` if the first result is a `ResultSet` object; `false` if the first result is an update count or there is no result

Throws:

`SQLException` - if a database access error occurs or an argument is supplied to this method

See Also:

`Statement.execute(java.lang.String)`, `Statement.getResultSet()`, `Statement.getUpdateCount()`, `Statement.getMoreResults()`

---

addBatch

```
public void addBatch()
            throws SQLException
```

Adds a set of parameters to this `PreparedStatement` object's batch of commands.

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

See Also:

`Statement.addBatch(java.lang.String)`

---

setCharacterStream

```
public void setCharacterStream(int parameterIndex,
                               Reader reader,
                               int length)
                        throws SQLException
```

Sets the designated parameter to the given `Reader` object, which is the given number of characters long. When a very large UNICODE value is input to a `LONGVARCHAR` parameter, it may be more practical to send it via a `java.io.Reader` object. The data will be read from the stream as needed until end-of-file is reached. The JDBC driver will do any necessary conversion from UNICODE to the database char format.

Note: This stream object can either be a standard Java stream object or your own subclass that implements the standard interface.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`reader` - the `java.io.Reader` object that contains the Unicode data
`length` - the number of characters in the stream

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

setRef

```
public void setRef(int i,
                   Ref x)
            throws SQLException
```

Sets the designated parameter to the given `REF(<structured-type>)` value. The driver converts this to an SQL `REF` value when it sends it to the database.

Parameters:

`i` - the first parameter is 1, the second is 2, ...
`x` - an SQL `REF` value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

setBlob

```
public void setBlob(int i,
                    Blob x)
             throws SQLException
```

Sets the designated parameter to the given `Blob` object. The driver converts this to an SQL `BLOB` value when it sends it to the database.

Parameters:

`i` - the first parameter is 1, the second is 2, ...
`x` - a `Blob` object that maps an SQL `BLOB` value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

setClob

```
public void setClob(int i,
                    Clob x)
             throws SQLException
```

Sets the designated parameter to the given `Clob` object. The driver converts this to an SQL `CLOB` value when it sends it to the database.

Parameters:

`i` - the first parameter is 1, the second is 2, ...
`x` - a `Clob` object that maps an SQL `CLOB` value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

setArray

```
public void setArray(int i,
                     Array x)
              throws SQLException
```

Sets the designated parameter to the given `Array` object. The driver converts this to an SQL ARRAY value when it sends it to the database.

Parameters:

`i` - the first parameter is 1, the second is 2, ...
`x` - an `Array` object that maps an SQL ARRAY value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getMetaData

```
public ResultSetMetaData getMetaData()
                                throws SQLException
```

Retrieves a `ResultSetMetaData` object that contains information about the columns of the `ResultSet` object that will be returned when this `PreparedStatement` object is executed.

Because a `PreparedStatement` object is precompiled, it is possible to know about the `ResultSet` object that it will return without having to execute it. Consequently, it is possible to invoke the method `getMetaData` on a `PreparedStatement` object rather than waiting to execute it and then invoking the `ResultSet.getMetaData` method on the `ResultSet` object that is returned.

NOTE: Using this method may be expensive for some drivers due to the lack of underlying DBMS support.

Returns:

the description of a `ResultSet` object's columns or `null` if the driver cannot return a `ResultSetMetaData` object

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

setDate

```
public void setDate(int parameterIndex,
                    Date x,
                    Calendar cal)
             throws SQLException
```

Sets the designated parameter to the given `java.sql.Date` value, using the given `Calendar` object. The driver uses the `Calendar` object to construct an SQL `DATE` value, which the driver then sends to the database. With a `Calendar` object, the driver can calculate the date taking into account a custom timezone. If no `Calendar` object is specified, the driver uses the default timezone, which is that of the virtual machine running the application.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value
`cal` - the `Calendar` object the driver will use to construct the date

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

setTime

```
public void setTime(int parameterIndex,
                    Time x,
                    Calendar cal)
             throws SQLException
```

Sets the designated parameter to the given `java.sql.Time` value, using the given `Calendar` object. The driver uses the `Calendar` object to construct an SQL `TIME` value, which the driver then sends to the database. With a `Calendar` object, the driver can calculate the time taking into account a custom timezone. If no `Calendar` object is specified, the driver uses the default timezone, which is that of the virtual machine running the application.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value
`cal` - the `Calendar` object the driver will use to construct the time

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

setTimestamp

```
public void setTimestamp(int parameterIndex,
```

```
                    Timestamp x,
                    Calendar cal)
             throws SQLException
```

Sets the designated parameter to the given `java.sql.Timestamp` value, using the given `Calendar` object. The driver uses the `Calendar` object to construct an SQL `TIMESTAMP` value, which the driver then sends to the database. With a `Calendar` object, the driver can calculate the timestamp taking into account a custom timezone. If no `Calendar` object is specified, the driver uses the default timezone, which is that of the virtual machine running the application.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the parameter value
`cal` - the `Calendar` object the driver will use to construct the timestamp

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

setNull

```
public void setNull(int paramIndex,
                    int sqlType,
                    String typeName)
             throws SQLException
```

Sets the designated parameter to SQL `NULL`. This version of the method `setNull` should be used for user-defined types and REF type parameters. Examples of user-defined types include: STRUCT, DISTINCT, JAVA_OBJECT, and named array types.

Note: To be portable, applications must give the SQL type code and the fully-qualified SQL type name when specifying a NULL user-defined or REF parameter. In the case of a user-defined type the name is the type name of the parameter itself. For a REF parameter, the name is the type name of the referenced type. If a JDBC driver does not need the type code or type name information, it may ignore it. Although it is intended for user-defined and Ref parameters, this method may be used to set a null parameter of any JDBC type. If the parameter does not have a user-defined or REF type, the given typeName is ignored.

Parameters:

`paramIndex` - the first parameter is 1, the second is 2, ...
`sqlType` - a value from `java.sql.Types`

`typeName` - the fully-qualified name of an SQL user-defined type; ignored if the parameter is not a user-defined type or REF

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

setURL

```
public void setURL(int parameterIndex,
                   URL x)
            throws SQLException
```

Sets the designated parameter to the given `java.net.URL` value. The driver converts this to an SQL `DATALINK` value when it sends it to the database.

Parameters:

`parameterIndex` - the first parameter is 1, the second is 2, ...
`x` - the `java.net.URL` object to be set

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

getParameterMetaData

```
public ParameterMetaData getParameterMetaData()
                                        throws SQLException
```

Retrieves the number, types and properties of this `PreparedStatement` object's parameters.

Returns:

a `ParameterMetaData` object that contains information about the number, types and properties of this `PreparedStatement` object's parameters

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

See Also:

`ParameterMetaData`

Submit a bug or feature

For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

java.sql

Interface ResultSet

All Known Subinterfaces:

RowSet

public interface **ResultSet**

A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

A `ResultSet` object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The `next` method moves the cursor to the next row, and because it returns `false` when there are no more rows in the `ResultSet` object, it can be used in a `while` loop to iterate through the result set.

A default `ResultSet` object is not updatable and has a cursor that moves forward only. Thus, you can iterate through it only once and only from the first row to the last row. It is possible to produce `ResultSet` objects that are scrollable and/or updatable. The following code fragment, in which `con` is a valid `Connection` object, illustrates how to make a result set that is scrollable and insensitive to updates by others, and that is updatable. See `ResultSet` fields for other options.

```
       Statement stmt = con.createStatement(
                                       ResultSet.
TYPE_SCROLL_INSENSITIVE,
                                       ResultSet.CONCUR_UPDATABLE);
       ResultSet rs = stmt.executeQuery("SELECT a, b FROM TABLE2");
       // rs will be scrollable, will not show changes made by
others,
       // and will be updatable
```

The `ResultSet` interface provides *getter* methods (`getBoolean`, `getLong`, and so on) for retrieving column values from the current row. Values can be retrieved using either the index number of the column or the name of the column. In general, using the column index will be more efficient.

Columns are numbered from 1. For maximum portability, result set columns within each row should be read in left-to-right order, and each column should be read only once.

For the getter methods, a JDBC driver attempts to convert the underlying data to the Java type specified in the getter method and returns a suitable Java value. The JDBC specification has a table showing the allowable mappings from SQL types to Java types that can be used by the `ResultSet` getter methods.

Column names used as input to getter methods are case insensitive. When a getter method is called with a column name and several columns have the same name, the value of the first matching column will be returned. The column name option is designed to be used when column names are used in the SQL query that generated the result set. For columns that are NOT explicitly named in the query, it is best to use column numbers. If column names are used, there is no way for the programmer to guarantee that they actually refer to the intended columns.

A set of updater methods were added to this interface in the JDBC 2.0 API (Java™ 2 SDK, Standard Edition, version 1.2). The comments regarding parameters to the getter methods also apply to parameters to the updater methods.

The updater methods may be used in two ways:

1. to update a column value in the current row. In a scrollable `ResultSet` object, the cursor can be moved backwards and forwards, to an absolute position, or to a position relative to the current row. The following code fragment updates the `NAME` column in the fifth row of the `ResultSet` object `rs` and then uses the method `updateRow` to update the data source table from which `rs` was derived.

```
        rs.absolute(5); // moves the cursor to the fifth row of
rs
        rs.updateString("NAME", "AINSWORTH"); // updates the
           // NAME column of row 5 to be AINSWORTH
        rs.updateRow(); // updates the row in the data source
```

2. to insert column values into the insert row. An updatable `ResultSet` object has a special row associated with it that serves as a staging area for building a row to be inserted. The following code fragment moves the cursor to the insert row, builds a three-column row, and inserts it into `rs` and into the data source table using the method `insertRow`.

```
        rs.moveToInsertRow(); // moves cursor to the insert row
        rs.updateString(1, "AINSWORTH"); // updates the
           // first column of the insert row to be AINSWORTH
```

```
        rs.updateInt(2,35); // updates the second column to be
    35
        rs.updateBoolean(3, true); // updates the third column
    to true
        rs.insertRow();
        rs.moveToCurrentRow();
```

A `ResultSet` object is automatically closed when the `Statement` object that generated it is closed, re-executed, or used to retrieve the next result from a sequence of multiple results.

The number, types and properties of a `ResultSet` object's columns are provided by the `ResulSetMetaData` object returned by the `ResultSet.getMetaData` method.

See Also:

`Statement.executeQuery(java.lang.String)`, `Statement.getResultSet()`, `ResultSetMetaData`

Field Summary

`static int`	**`CLOSE_CURSORS_AT_COMMIT`** The constant indicating that `ResultSet` objects should be closed when the method `Connection.commit` is called.
`static int`	**`CONCUR_READ_ONLY`** The constant indicating the concurrency mode for a `ResultSet` object that may NOT be updated.
`static int`	**`CONCUR_UPDATABLE`** The constant indicating the concurrency mode for a `ResultSet` object that may be updated.
`static int`	**`FETCH_FORWARD`** The constant indicating that the rows in a result set will be processed in a forward direction; first-to-last.
`static int`	**`FETCH_REVERSE`** The constant indicating that the rows in a result set will be processed in a reverse direction; last-to-first.
`static int`	**`FETCH_UNKNOWN`** The constant indicating that the order in which rows in a result set will be processed is unknown.

static int	**HOLD_CURSORS_OVER_COMMIT** The constant indicating that ResultSet objects should not be closed when the method Connection.commit is called.
static int	**TYPE_FORWARD_ONLY** The constant indicating the type for a ResultSet object whose cursor may move only forward.
static int	**TYPE_SCROLL_INSENSITIVE** The constant indicating the type for a ResultSet object that is scrollable but generally not sensitive to changes made by others.
static int	**TYPE_SCROLL_SENSITIVE** The constant indicating the type for a ResultSet object that is scrollable and generally sensitive to changes made by others.

Method Summary

boolean	**absolute**(int row) Moves the cursor to the given row number in this ResultSet object.
void	**afterLast**() Moves the cursor to the end of this ResultSet object, just after the last row.
void	**beforeFirst**() Moves the cursor to the front of this ResultSet object, just before the first row.
void	**cancelRowUpdates**() Cancels the updates made to the current row in this ResultSet object.
void	**clearWarnings**() Clears all warnings reported on this ResultSet object.
void	**close**() Releases this ResultSet object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.
void	**deleteRow**() Deletes the current row from this ResultSet object and from the underlying database.
int	**findColumn**(String columnName) Maps the given ResultSet column name to its ResultSet column index.

boolean	**first**() Moves the cursor to the first row in this ResultSet object.
Array	**getArray**(int i) Retrieves the value of the designated column in the current row of this ResultSet object as an Array object in the Java programming language.
Array	**getArray**(String colName) Retrieves the value of the designated column in the current row of this ResultSet object as an Array object in the Java programming language.
InputStream	**getAsciiStream**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a stream of ASCII characters.
InputStream	**getAsciiStream**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a stream of ASCII characters.
BigDecimal	**getBigDecimal**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a java.math.BigDecimal with full precision.
BigDecimal	**getBigDecimal**(int columnIndex, int scale) **Deprecated.**
BigDecimal	**getBigDecimal**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a java.math.BigDecimal with full precision.
BigDecimal	**getBigDecimal**(String columnName, int scale) **Deprecated.**
InputStream	**getBinaryStream**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a binary stream of uninterpreted bytes.
InputStream	**getBinaryStream**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a stream of uninterpreted bytes.
Blob	**getBlob**(int i) Retrieves the value of the designated column in the current row of this ResultSet object as a Blob object in the Java programming language.

Blob	**getBlob**(String colName) Retrieves the value of the designated column in the current row of this ResultSet object as a Blob object in the Java programming language.
boolean	**getBoolean**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a boolean in the Java programming language.
boolean	**getBoolean**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a boolean in the Java programming language.
byte	**getByte**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a byte in the Java programming language.
byte	**getByte**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a byte in the Java programming language.
byte[]	**getBytes**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a byte array in the Java programming language.
byte[]	**getBytes**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a byte array in the Java programming language.
Reader	**getCharacterStream**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a java.io.Reader object.
Reader	**getCharacterStream**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a java.io.Reader object.
Clob	**getClob**(int i) Retrieves the value of the designated column in the current row of this ResultSet object as a Clob object in the Java programming language.

Clob	**getClob**(String colName) Retrieves the value of the designated column in the current row of this ResultSet object as a Clob object in the Java programming language.
int	**getConcurrency**() Retrieves the concurrency mode of this ResultSet object.
String	**getCursorName**() Retrieves the name of the SQL cursor used by this ResultSet object.
Date	**getDate**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
Date	**getDate**(int columnIndex, Calendar cal) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
Date	**getDate**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
Date	**getDate**(String columnName, Calendar cal) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.
double	**getDouble**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a double in the Java programming language.
double	**getDouble**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a double in the Java programming language.
int	**getFetchDirection**() Retrieves the fetch direction for this ResultSet object.
int	**getFetchSize**() Retrieves the fetch size for this ResultSet object.
float	**getFloat**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a float in the Java programming language.

float	**getFloat**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a float in the Java programming language.
int	**getInt**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as an int in the Java programming language.
int	**getInt**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as an int in the Java programming language.
long	**getLong**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a long in the Java programming language.
long	**getLong**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a long in the Java programming language.
ResultSetMetaData	**getMetaData**() Retrieves the number, types and properties of this ResultSet object's columns.
Object	**getObject**(int columnIndex) Gets the value of the designated column in the current row of this ResultSet object as an Object in the Java programming language.
Object	**getObject**(int i, Map map) Retrieves the value of the designated column in the current row of this ResultSet object as an Object in the Java programming language.
Object	**getObject**(String columnName) Gets the value of the designated column in the current row of this ResultSet object as an Object in the Java programming language.
Object	**getObject**(String colName, Map map) Retrieves the value of the designated column in the current row of this ResultSet object as an Object in the Java programming language.
Ref	**getRef**(int i) Retrieves the value of the designated column in the current row of this ResultSet object as a Ref object in the Java programming language.

Ref	**getRef**(String colName) Retrieves the value of the designated column in the current row of this ResultSet object as a Ref object in the Java programming language.
int	**getRow**() Retrieves the current row number.
short	**getShort**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a short in the Java programming language.
short	**getShort**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a short in the Java programming language.
Statement	**getStatement**() Retrieves the Statement object that produced this ResultSet object.
String	**getString**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.
String	**getString**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.
Time	**getTime**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.
Time	**getTime**(int columnIndex, Calendar cal) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.
Time	**getTime**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.
Time	**getTime**(String columnName, Calendar cal) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.

Timestamp	**getTimestamp**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language.
Timestamp	**getTimestamp**(int columnIndex, Calendar cal) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language.
Timestamp	**getTimestamp**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object.
Timestamp	**getTimestamp**(String columnName, Calendar cal) Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language.
int	**getType**() Retrieves the type of this ResultSet object.
InputStream	**getUnicodeStream**(int columnIndex) **Deprecated.** *use getCharacterStream in place of getUnicodeStream*
InputStream	**getUnicodeStream**(String columnName) **Deprecated.** *use getCharacterStream instead*
URL	**getURL**(int columnIndex) Retrieves the value of the designated column in the current row of this ResultSet object as a java.net.URL object in the Java programming language.
URL	**getURL**(String columnName) Retrieves the value of the designated column in the current row of this ResultSet object as a java.net.URL object in the Java programming language.
SQLWarning	**getWarnings**() Retrieves the first warning reported by calls on this ResultSet object.
void	**insertRow**() Inserts the contents of the insert row into this ResultSet object and into the database.
boolean	**isAfterLast**() Retrieves whether the cursor is after the last row in this ResultSet object.

boolean	**isBeforeFirst**() Retrieves whether the cursor is before the first row in this ResultSet object.
boolean	**isFirst**() Retrieves whether the cursor is on the first row of this ResultSet object.
boolean	**isLast**() Retrieves whether the cursor is on the last row of this ResultSet object.
boolean	**last**() Moves the cursor to the last row in this ResultSet object.
void	**moveToCurrentRow**() Moves the cursor to the remembered cursor position, usually the current row.
void	**moveToInsertRow**() Moves the cursor to the insert row.
boolean	**next**() Moves the cursor down one row from its current position.
boolean	**previous**() Moves the cursor to the previous row in this ResultSet object.
void	**refreshRow**() Refreshes the current row with its most recent value in the database.
boolean	**relative**(int rows) Moves the cursor a relative number of rows, either positive or negative.
boolean	**rowDeleted**() Retrieves whether a row has been deleted.
boolean	**rowInserted**() Retrieves whether the current row has had an insertion.
boolean	**rowUpdated**() Retrieves whether the current row has been updated.
void	**setFetchDirection**(int direction) Gives a hint as to the direction in which the rows in this ResultSet object will be processed.

void	**setFetchSize**(int rows) Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed for this ResultSet object.
void	**updateArray**(int columnIndex, Array x) Updates the designated column with a java.sql.Array value.
void	**updateArray**(String columnName, Array x) Updates the designated column with a java.sql.Array value.
void	**updateAsciiStream**(int columnIndex, InputStream x, int length) Updates the designated column with an ascii stream value.
void	**updateAsciiStream**(String columnName, InputStream x, int length) Updates the designated column with an ascii stream value.
void	**updateBigDecimal**(int columnIndex, BigDecimal x) Updates the designated column with a java.math.BigDecimal value.
void	**updateBigDecimal**(String columnName, BigDecimal x) Updates the designated column with a java.sql.BigDecimal value.
void	**updateBinaryStream**(int columnIndex, InputStream x, int length) Updates the designated column with a binary stream value.
void	**updateBinaryStream**(String columnName, InputStream x, int length) Updates the designated column with a binary stream value.
void	**updateBlob**(int columnIndex, Blob x) Updates the designated column with a java.sql.Blob value.
void	**updateBlob**(String columnName, Blob x) Updates the designated column with a java.sql.Blob value.
void	**updateBoolean**(int columnIndex, boolean x) Updates the designated column with a boolean value.
void	**updateBoolean**(String columnName, boolean x) Updates the designated column with a boolean value.
void	**updateByte**(int columnIndex, byte x) Updates the designated column with a byte value.

void	**updateByte**(String columnName, byte x) Updates the designated column with a byte value.
void	**updateBytes**(int columnIndex, byte[] x) Updates the designated column with a byte array value.
void	**updateBytes**(String columnName, byte[] x) Updates the designated column with a byte array value.
void	**updateCharacterStream**(int columnIndex, Reader x, int length) Updates the designated column with a character stream value.
void	**updateCharacterStream**(String columnName, Reader reader, int length) Updates the designated column with a character stream value.
void	**updateClob**(int columnIndex, Clob x) Updates the designated column with a java.sql.Clob value.
void	**updateClob**(String columnName, Clob x) Updates the designated column with a java.sql.Clob value.
void	**updateDate**(int columnIndex, Date x) Updates the designated column with a java.sql.Date value.
void	**updateDate**(String columnName, Date x) Updates the designated column with a java.sql.Date value.
void	**updateDouble**(int columnIndex, double x) Updates the designated column with a double value.
void	**updateDouble**(String columnName, double x) Updates the designated column with a double value.
void	**updateFloat**(int columnIndex, float x) Updates the designated column with a float value.
void	**updateFloat**(String columnName, float x) Updates the designated column with a float value.
void	**updateInt**(int columnIndex, int x) Updates the designated column with an int value.
void	**updateInt**(String columnName, int x) Updates the designated column with an int value.
void	**updateLong**(int columnIndex, long x) Updates the designated column with a long value.
void	**updateLong**(String columnName, long x) Updates the designated column with a long value.

void	**updateNull**(int columnIndex) Gives a nullable column a null value.
void	**updateNull**(String columnName) Updates the designated column with a null value.
void	**updateObject**(int columnIndex, Object x) Updates the designated column with an Object value.
void	**updateObject**(int columnIndex, Object x, int scale) Updates the designated column with an Object value.
void	**updateObject**(String columnName, Object x) Updates the designated column with an Object value.
void	**updateObject**(String columnName, Object x, int scale) Updates the designated column with an Object value.
void	**updateRef**(int columnIndex, Ref x) Updates the designated column with a java.sql.Ref value.
void	**updateRef**(String columnName, Ref x) Updates the designated column with a java.sql.Ref value.
void	**updateRow**() Updates the underlying database with the new contents of the current row of this ResultSet object.
void	**updateShort**(int columnIndex, short x) Updates the designated column with a short value.
void	**updateShort**(String columnName, short x) Updates the designated column with a short value.
void	**updateString**(int columnIndex, String x) Updates the designated column with a String value.
void	**updateString**(String columnName, String x) Updates the designated column with a String value.
void	**updateTime**(int columnIndex, Time x) Updates the designated column with a java.sql.Time value.
void	**updateTime**(String columnName, Time x) Updates the designated column with a java.sql.Time value.
void	**updateTimestamp**(int columnIndex, Timestamp x) Updates the designated column with a java.sql.Timestamp value.

void	**updateTimestamp**(String columnName, Timestamp x) Updates the designated column with a java.sql.Timestamp value.
boolean	**wasNull**() Reports whether the last column read had a value of SQL NULL.

Field Detail

FETCH_FORWARD

```
public static final int FETCH_FORWARD
```

The constant indicating that the rows in a result set will be processed in a forward direction; first-to-last. This constant is used by the method setFetchDirection as a hint to the driver, which the driver may ignore.

Since:
1.2
See Also:
Constant Field Values

FETCH_REVERSE

```
public static final int FETCH_REVERSE
```

The constant indicating that the rows in a result set will be processed in a reverse direction; last-to-first. This constant is used by the method setFetchDirection as a hint to the driver, which the driver may ignore.

Since:
1.2
See Also:
Constant Field Values

FETCH_UNKNOWN

```
public static final int FETCH_UNKNOWN
```

The constant indicating that the order in which rows in a result set will be processed is unknown. This constant is used by the method `setFetchDirection` as a hint to the driver, which the driver may ignore.

See Also:
Constant Field Values

TYPE_FORWARD_ONLY

`public static final int` **`TYPE_FORWARD_ONLY`**

The constant indicating the type for a `ResultSet` object whose cursor may move only forward.

Since:
1.2
See Also:
Constant Field Values

TYPE_SCROLL_INSENSITIVE

`public static final int` **`TYPE_SCROLL_INSENSITIVE`**

The constant indicating the type for a `ResultSet` object that is scrollable but generally not sensitive to changes made by others.

Since:
1.2
See Also:
Constant Field Values

TYPE_SCROLL_SENSITIVE

`public static final int` **`TYPE_SCROLL_SENSITIVE`**

The constant indicating the type for a `ResultSet` object that is scrollable and generally

sensitive to changes made by others.

Since:
1.2
See Also:
Constant Field Values

CONCUR_READ_ONLY

```
public static final int CONCUR_READ_ONLY
```

The constant indicating the concurrency mode for a `ResultSet` object that may NOT be updated.

Since:
1.2
See Also:
Constant Field Values

CONCUR_UPDATABLE

```
public static final int CONCUR_UPDATABLE
```

The constant indicating the concurrency mode for a `ResultSet` object that may be updated.

Since:
1.2
See Also:
Constant Field Values

HOLD_CURSORS_OVER_COMMIT

```
public static final int HOLD_CURSORS_OVER_COMMIT
```

The constant indicating that `ResultSet` objects should not be closed when the method `Connection.commit` is called.

Since:

1.4

See Also:

Constant Field Values

CLOSE_CURSORS_AT_COMMIT

```
public static final int CLOSE_CURSORS_AT_COMMIT
```

The constant indicating that `ResultSet` objects should be closed when the method `Connection.commit` is called.

Since:

1.4

See Also:

Constant Field Values

Method Detail

next

```
public boolean next()
              throws SQLException
```

Moves the cursor down one row from its current position. A `ResultSet` cursor is initially positioned before the first row; the first call to the method `next` makes the first row the current row; the second call makes the second row the current row, and so on.

If an input stream is open for the current row, a call to the method `next` will implicitly close it. A `ResultSet` object's warning chain is cleared when a new row is read.

Returns:

`true` if the new current row is valid; `false` if there are no more rows

Throws:

`SQLException` - if a database access error occurs

---

close

```
public void close()
            throws SQLException
```

Releases this `ResultSet` object's database and JDBC resources immediately instead of waiting for this to happen when it is automatically closed.

Note: A `ResultSet` object is automatically closed by the `Statement` object that generated it when that `Statement` object is closed, re-executed, or is used to retrieve the next result from a sequence of multiple results. A `ResultSet` object is also automatically closed when it is garbage collected.

Throws:

`SQLException` - if a database access error occurs

---

wasNull

```
public boolean wasNull()
                throws SQLException
```

Reports whether the last column read had a value of SQL `NULL`. Note that you must first call one of the getter methods on a column to try to read its value and then call the method `wasNull` to see if the value read was SQL `NULL`.

Returns:

`true` if the last column value read was SQL `NULL` and `false` otherwise

Throws:

`SQLException` - if a database access error occurs

---

getString

```
public String getString(int columnIndex)
                 throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `String` in the Java programming language.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

Returns:

the column value; if the value is SQL `NULL`, the value returned is `null`

Throws:

`SQLException` - if a database access error occurs

---

getBoolean

```
public boolean getBoolean(int columnIndex)
                   throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `boolean` in the Java programming language.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

Returns:

the column value; if the value is SQL `NULL`, the value returned is `false`

Throws:

`SQLException` - if a database access error occurs

---

getByte

```
public byte getByte(int columnIndex)
             throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `byte` in the Java programming language.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

Returns:

the column value; if the value is SQL `NULL`, the value returned is `0`

Throws:

`SQLException` - if a database access error occurs

---

getShort

```
public short getShort(int columnIndex)
```

```
        throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `short` in the Java programming language.

Parameters:
`columnIndex` - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL `NULL`, the value returned is `0`
Throws:
`SQLException` - if a database access error occurs

---

getInt

```
public int getInt(int columnIndex)
           throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as an `int` in the Java programming language.

Parameters:
`columnIndex` - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL `NULL`, the value returned is `0`
Throws:
`SQLException` - if a database access error occurs

---

getLong

```
public long getLong(int columnIndex)
            throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `long` in the Java programming language.

Parameters:
`columnIndex` - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL `NULL`, the value returned is `0`
Throws:

SQLException - if a database access error occurs

---

getFloat

```
public float getFloat(int columnIndex)
               throws SQLException
```

Retrieves the value of the designated column in the current row of this ResultSet object as a float in the Java programming language.

Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the value returned is 0
Throws:
SQLException - if a database access error occurs

---

getDouble

```
public double getDouble(int columnIndex)
                throws SQLException
```

Retrieves the value of the designated column in the current row of this ResultSet object as a double in the Java programming language.

Parameters:
columnIndex - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL NULL, the value returned is 0
Throws:
SQLException - if a database access error occurs

---

getBigDecimal

```
public BigDecimal getBigDecimal(int columnIndex,
                                int scale)
                         throws SQLException
```

Deprecated.

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.BigDecimal` in the Java programming language.

Parameters:
`columnIndex` - the first column is 1, the second is 2, ...
`scale` - the number of digits to the right of the decimal point
Returns:
the column value; if the value is SQL `NULL`, the value returned is `null`
Throws:
`SQLException` - if a database access error occurs

---

getBytes

```
public byte[] getBytes(int columnIndex)
               throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `byte` array in the Java programming language. The bytes represent the raw values returned by the driver.

Parameters:
`columnIndex` - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL `NULL`, the value returned is `null`
Throws:
`SQLException` - if a database access error occurs

---

getDate

```
public Date getDate(int columnIndex)
            throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.Date` object in the Java programming language.

Parameters:
`columnIndex` - the first column is 1, the second is 2, ...

Returns:
the column value; if the value is SQL `NULL`, the value returned is `null`
Throws:
`SQLException` - if a database access error occurs

---

getTime

```
public Time getTime(int columnIndex)
             throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.Time` object in the Java programming language.

Parameters:
`columnIndex` - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL `NULL`, the value returned is `null`
Throws:
`SQLException` - if a database access error occurs

---

getTimestamp

```
public Timestamp getTimestamp(int columnIndex)
                       throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.Timestamp` object in the Java programming language.

Parameters:
`columnIndex` - the first column is 1, the second is 2, ...
Returns:
the column value; if the value is SQL `NULL`, the value returned is `null`
Throws:
`SQLException` - if a database access error occurs

---

getAsciiStream

```
public InputStream getAsciiStream(int columnIndex)
                           throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a stream of ASCII characters. The value can then be read in chunks from the stream. This method is particularly suitable for retrieving large LONGVARCHAR values. The JDBC driver will do any necessary conversion from the database format into ASCII.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a getter method implicitly closes the stream. Also, a stream may return `0` when the method `InputStream.available` is called whether there is data available or not.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

Returns:

a Java input stream that delivers the database column value as a stream of one-byte ASCII characters; if the value is SQL `NULL`, the value returned is `null`

Throws:

`SQLException` - if a database access error occurs

---

getUnicodeStream

```
public InputStream getUnicodeStream(int columnIndex)
                             throws SQLException
```

Deprecated. *use `getCharacterStream` in place of `getUnicodeStream`*

Retrieves the value of the designated column in the current row of this `ResultSet` object as as a stream of two-byte Unicode characters. The first byte is the high byte; the second byte is the low byte. The value can then be read in chunks from the stream. This method is particularly suitable for retrieving large `LONGVARCHAR`values. The JDBC driver will do any necessary conversion from the database format into Unicode.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a getter method implicitly closes the stream. Also, a stream may return `0` when the method `InputStream.available` is called, whether there is data available or not.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

Returns:

a Java input stream that delivers the database column value as a stream of two-byte Unicode characters; if the value is SQL `NULL`, the value returned is `null`

Throws:

`SQLException` - if a database access error occurs

---

getBinaryStream

```
public InputStream getBinaryStream(int columnIndex)
                            throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a binary stream of uninterpreted bytes. The value can then be read in chunks from the stream. This method is particularly suitable for retrieving large `LONGVARBINARY` values.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a getter method implicitly closes the stream. Also, a stream may return `0` when the method `InputStream.available` is called whether there is data available or not.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

Returns:

a Java input stream that delivers the database column value as a stream of uninterpreted bytes; if the value is SQL `NULL`, the value returned is `null`

Throws:

`SQLException` - if a database access error occurs

---

getString

```
public String getString(String columnName)
                 throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `String` in the Java programming language.

Parameters:

`columnName` - the SQL name of the column

Returns:

the column value; if the value is SQL `NULL`, the value returned is `null`

Throws:

SQLException - if a database access error occurs

---

getBoolean

```
public boolean getBoolean(String columnName)
                   throws SQLException
```

Retrieves the value of the designated column in the current row of this ResultSet object as a boolean in the Java programming language.

Parameters:
columnName - the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the value returned is false
Throws:
SQLException - if a database access error occurs

---

getByte

```
public byte getByte(String columnName)
             throws SQLException
```

Retrieves the value of the designated column in the current row of this ResultSet object as a byte in the Java programming language.

Parameters:
columnName - the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the value returned is 0
Throws:
SQLException - if a database access error occurs

---

getShort

```
public short getShort(String columnName)
               throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `short` in the Java programming language.

Parameters:
`columnName` - the SQL name of the column
Returns:
the column value; if the value is SQL `NULL`, the value returned is `0`
Throws:
`SQLException` - if a database access error occurs

---

getInt

```
public int getInt(String columnName)
           throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as an `int` in the Java programming language.

Parameters:
`columnName` - the SQL name of the column
Returns:
the column value; if the value is SQL `NULL`, the value returned is `0`
Throws:
`SQLException` - if a database access error occurs

---

getLong

```
public long getLong(String columnName)
            throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `long` in the Java programming language.

Parameters:
`columnName` - the SQL name of the column
Returns:
the column value; if the value is SQL `NULL`, the value returned is `0`
Throws:
`SQLException` - if a database access error occurs

---

getFloat

```
public float getFloat(String columnName)
               throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `float` in the Java programming language.

Parameters:
`columnName` - the SQL name of the column
Returns:
the column value; if the value is SQL `NULL`, the value returned is `0`
Throws:
`SQLException` - if a database access error occurs

---

getDouble

```
public double getDouble(String columnName)
                throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `double` in the Java programming language.

Parameters:
`columnName` - the SQL name of the column
Returns:
the column value; if the value is SQL `NULL`, the value returned is `0`
Throws:
`SQLException` - if a database access error occurs

---

getBigDecimal

```
public BigDecimal getBigDecimal(String columnName,
                                int scale)
                         throws SQLException
```

Deprecated.

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.math.BigDecimal` in the Java programming language.

Parameters:
`columnName` - the SQL name of the column
`scale` - the number of digits to the right of the decimal point
Returns:
the column value; if the value is SQL `NULL`, the value returned is `null`
Throws:
`SQLException` - if a database access error occurs

---

getBytes

```
public byte[] getBytes(String columnName)
                throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `byte` array in the Java programming language. The bytes represent the raw values returned by the driver.

Parameters:
`columnName` - the SQL name of the column
Returns:
the column value; if the value is SQL `NULL`, the value returned is `null`
Throws:
`SQLException` - if a database access error occurs

---

getDate

```
public Date getDate(String columnName)
             throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.Date` object in the Java programming language.

Parameters:
`columnName` - the SQL name of the column
Returns:
the column value; if the value is SQL `NULL`, the value returned is `null`

Throws:
SQLException - if a database access error occurs

---

getTime

```
public Time getTime(String columnName)
             throws SQLException
```

Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.

Parameters:
columnName - the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the value returned is null
Throws:
SQLException - if a database access error occurs

---

getTimestamp

```
public Timestamp getTimestamp(String columnName)
                      throws SQLException
```

Retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object.

Parameters:
columnName - the SQL name of the column
Returns:
the column value; if the value is SQL NULL, the value returned is null
Throws:
SQLException - if a database access error occurs

---

getAsciiStream

```
public InputStream getAsciiStream(String columnName)
                           throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a stream of ASCII characters. The value can then be read in chunks from the stream. This method is particularly suitable for retrieving large `LONGVARCHAR` values. The JDBC driver will do any necessary conversion from the database format into ASCII.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a getter method implicitly closes the stream. Also, a stream may return `0` when the method `available` is called whether there is data available or not.

Parameters:

`columnName` - the SQL name of the column

Returns:

a Java input stream that delivers the database column value as a stream of one-byte ASCII characters. If the value is SQL `NULL`, the value returned is `null`.

Throws:

`SQLException` - if a database access error occurs

---

getUnicodeStream

```
public InputStream getUnicodeStream(String columnName)
                             throws SQLException
```

Deprecated. *use `getCharacterStream` instead*

Retrieves the value of the designated column in the current row of this `ResultSet` object as a stream of two-byte Unicode characters. The first byte is the high byte; the second byte is the low byte. The value can then be read in chunks from the stream. This method is particularly suitable for retrieving large `LONGVARCHAR` values. The JDBC technology-enabled driver will do any necessary conversion from the database format into Unicode.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a getter method implicitly closes the stream. Also, a stream may return `0` when the method `InputStream.available` is called, whether there is data available or not.

Parameters:

`columnName` - the SQL name of the column

Returns:

a Java input stream that delivers the database column value as a stream of two-byte Unicode characters. If the value is SQL `NULL`, the value returned is `null`.

Throws:

SQLException - if a database access error occurs

---

getBinaryStream

```
public InputStream getBinaryStream(String columnName)
                            throws SQLException
```

Retrieves the value of the designated column in the current row of this ResultSet object as a stream of uninterpreted bytes. The value can then be read in chunks from the stream. This method is particularly suitable for retrieving large LONGVARBINARY values.

Note: All the data in the returned stream must be read prior to getting the value of any other column. The next call to a getter method implicitly closes the stream. Also, a stream may return 0 when the method available is called whether there is data available or not.

Parameters:

columnName - the SQL name of the column

Returns:

a Java input stream that delivers the database column value as a stream of uninterpreted bytes; if the value is SQL NULL, the result is null

Throws:

SQLException - if a database access error occurs

---

getWarnings

```
public SQLWarning getWarnings()
                        throws SQLException
```

Retrieves the first warning reported by calls on this ResultSet object. Subsequent warnings on this ResultSet object will be chained to the SQLWarning object that this method returns.

The warning chain is automatically cleared each time a new row is read. This method may not be called on a ResultSet object that has been closed; doing so will cause an SQLException to be thrown.

Note: This warning chain only covers warnings caused by ResultSet methods. Any warning caused by Statement methods (such as reading OUT parameters) will be chained on the Statement object.

Returns:

the first `SQLWarning` object reported or `null` if there are none

Throws:

`SQLException` - if a database access error occurs or this method is called on a closed result set

---

clearWarnings

```
public void clearWarnings()
                  throws SQLException
```

Clears all warnings reported on this `ResultSet` object. After this method is called, the method `getWarnings` returns `null` until a new warning is reported for this `ResultSet` object.

Throws:

`SQLException` - if a database access error occurs

---

getCursorName

```
public String getCursorName()
                     throws SQLException
```

Retrieves the name of the SQL cursor used by this `ResultSet` object.

In SQL, a result table is retrieved through a cursor that is named. The current row of a result set can be updated or deleted using a positioned update/delete statement that references the cursor name. To insure that the cursor has the proper isolation level to support update, the cursor's `SELECT` statement should be of the form `SELECT FOR UPDATE`. If `FOR UPDATE` is omitted, the positioned updates may fail.

The JDBC API supports this SQL feature by providing the name of the SQL cursor used by a `ResultSet` object. The current row of a `ResultSet` object is also the current row of this SQL cursor.

Note: If positioned update is not supported, a `SQLException` is thrown.

Returns:

the SQL name for this `ResultSet` object's cursor

Throws:

SQLException - if a database access error occurs

---

getMetaData

```
public ResultSetMetaData getMetaData()
                            throws SQLException
```

Retrieves the number, types and properties of this `ResultSet` object's columns.

Returns:

the description of this `ResultSet` object's columns

Throws:

SQLException - if a database access error occurs

---

getObject

```
public Object getObject(int columnIndex)
                 throws SQLException
```

Gets the value of the designated column in the current row of this `ResultSet` object as an `Object` in the Java programming language.

This method will return the value of the given column as a Java object. The type of the Java object will be the default Java object type corresponding to the column's SQL type, following the mapping for built-in types specified in the JDBC specification. If the value is an SQL `NULL`, the driver returns a Java `null`.

This method may also be used to read database-specific abstract data types. In the JDBC 2.0 API, the behavior of method `getObject` is extended to materialize data of SQL user-defined types. When a column contains a structured or distinct value, the behavior of this method is as if it were a call to: `getObject(columnIndex, this.getStatement().getConnection().getTypeMap())`.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

Returns:

a `java.lang.Object` holding the column value

Throws:

SQLException - if a database access error occurs

---

getObject

```
public Object getObject(String columnName)
                 throws SQLException
```

Gets the value of the designated column in the current row of this ResultSet object as an Object in the Java programming language.

This method will return the value of the given column as a Java object. The type of the Java object will be the default Java object type corresponding to the column's SQL type, following the mapping for built-in types specified in the JDBC specification. If the value is an SQL NULL, the driver returns a Java null.

This method may also be used to read database-specific abstract data types.

In the JDBC 2.0 API, the behavior of the method getObject is extended to materialize data of SQL user-defined types. When a column contains a structured or distinct value, the behavior of this method is as if it were a call to: getObject(columnIndex, this.getStatement().getConnection().getTypeMap()).

Parameters:

columnName - the SQL name of the column

Returns:

a java.lang.Object holding the column value

Throws:

SQLException - if a database access error occurs

---

findColumn

```
public int findColumn(String columnName)
              throws SQLException
```

Maps the given ResultSet column name to its ResultSet column index.

Parameters:

columnName - the name of the column

Returns:

the column index of the given column name

Throws:

SQLException - if the ResultSet object does not contain columnName or a database access error occurs

---

getCharacterStream

```
public Reader getCharacterStream(int columnIndex)
                          throws SQLException
```

Retrieves the value of the designated column in the current row of this ResultSet object as a java.io.Reader object.

Parameters:

columnIndex - the first column is 1, the second is 2, ...

Returns:

a java.io.Reader object that contains the column value; if the value is SQL NULL, the value returned is null in the Java programming language.

Throws:

SQLException - if a database access error occurs

Since:

1.2

---

getCharacterStream

```
public Reader getCharacterStream(String columnName)
                          throws SQLException
```

Retrieves the value of the designated column in the current row of this ResultSet object as a java.io.Reader object.

Parameters:

columnName - the name of the column

Returns:

a java.io.Reader object that contains the column value; if the value is SQL NULL, the value returned is null in the Java programming language

Throws:

SQLException - if a database access error occurs

Since:

1.2

---

getBigDecimal

```
public BigDecimal getBigDecimal(int columnIndex)
                         throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.math.BigDecimal` with full precision.

Parameters:
`columnIndex` - the first column is 1, the second is 2, ...
Returns:
the column value (full precision); if the value is SQL `NULL`, the value returned is `null` in the Java programming language.
Throws:
`SQLException` - if a database access error occurs
Since:
1.2

---

getBigDecimal

```
public BigDecimal getBigDecimal(String columnName)
                         throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.math.BigDecimal` with full precision.

Parameters:
`columnName` - the column name
Returns:
the column value (full precision); if the value is SQL `NULL`, the value returned is `null` in the Java programming language.
Throws:
`SQLException` - if a database access error occurs
Since:
1.2

---

isBeforeFirst

```
public boolean isBeforeFirst()
                       throws SQLException
```

Retrieves whether the cursor is before the first row in this `ResultSet` object.

Returns:

`true` if the cursor is before the first row; `false` if the cursor is at any other position or the result set contains no rows

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

isAfterLast

```
public boolean isAfterLast()
                     throws SQLException
```

Retrieves whether the cursor is after the last row in this `ResultSet` object.

Returns:

`true` if the cursor is after the last row; `false` if the cursor is at any other position or the result set contains no rows

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

isFirst

```
public boolean isFirst()
                throws SQLException
```

Retrieves whether the cursor is on the first row of this `ResultSet` object.

Returns:

`true` if the cursor is on the first row; `false` otherwise

Throws:

SQLException - if a database access error occurs

Since:

1.2

---

isLast

```
public boolean isLast()
               throws SQLException
```

Retrieves whether the cursor is on the last row of this `ResultSet` object. Note: Calling the method `isLast` may be expensive because the JDBC driver might need to fetch ahead one row in order to determine whether the current row is the last row in the result set.

Returns:

`true` if the cursor is on the last row; `false` otherwise

Throws:

SQLException - if a database access error occurs

Since:

1.2

---

beforeFirst

```
public void beforeFirst()
                throws SQLException
```

Moves the cursor to the front of this `ResultSet` object, just before the first row. This method has no effect if the result set contains no rows.

Throws:

SQLException - if a database access error occurs or the result set type is `TYPE_FORWARD_ONLY`

Since:

1.2

---

afterLast

```
public void afterLast()
```

```
            throws SQLException
```

Moves the cursor to the end of this `ResultSet` object, just after the last row. This method has no effect if the result set contains no rows.

Throws:

SQLException - if a database access error occurs or the result set type is `TYPE_FORWARD_ONLY`

Since:

1.2

---

first

```
public boolean first()
              throws SQLException
```

Moves the cursor to the first row in this `ResultSet` object.

Returns:

`true` if the cursor is on a valid row; `false` if there are no rows in the result set

Throws:

SQLException - if a database access error occurs or the result set type is `TYPE_FORWARD_ONLY`

Since:

1.2

---

last

```
public boolean last()
             throws SQLException
```

Moves the cursor to the last row in this `ResultSet` object.

Returns:

`true` if the cursor is on a valid row; `false` if there are no rows in the result set

Throws:

SQLException - if a database access error occurs or the result set type is `TYPE_FORWARD_ONLY`

Since:

1.2

---

getRow

```
public int getRow()
           throws SQLException
```

Retrieves the current row number. The first row is number 1, the second number 2, and so on.

Returns:
the current row number; `0` if there is no current row

Throws:
`SQLException` - if a database access error occurs

Since:
1.2

---

absolute

```
public boolean absolute(int row)
                throws SQLException
```

Moves the cursor to the given row number in this `ResultSet` object.

If the row number is positive, the cursor moves to the given row number with respect to the beginning of the result set. The first row is row 1, the second is row 2, and so on.

If the given row number is negative, the cursor moves to an absolute row position with respect to the end of the result set. For example, calling the method `absolute(-1)` positions the cursor on the last row; calling the method `absolute(-2)` moves the cursor to the next-to-last row, and so on.

An attempt to position the cursor beyond the first/last row in the result set leaves the cursor before the first row or after the last row.

Note: Calling `absolute(1)` is the same as calling `first()`. Calling `absolute(-1)` is the same as calling `last()`.

Parameters:
`row` - the number of the row to which the cursor should move. A positive number indicates the row number counting from the beginning of the result set; a negative number indicates the row number counting from the end of the result set

Returns:

`true` if the cursor is on the result set; `false` otherwise

Throws:

`SQLException` - if a database access error occurs, or the result set type is `TYPE_FORWARD_ONLY`

Since:

1.2

---

relative

```
public boolean relative(int rows)
                throws SQLException
```

Moves the cursor a relative number of rows, either positive or negative. Attempting to move beyond the first/last row in the result set positions the cursor before/after the the first/last row. Calling `relative(0)` is valid, but does not change the cursor position.

Note: Calling the method `relative(1)` is identical to calling the method `next()` and calling the method `relative(-1)` is identical to calling the method `previous()`.

Parameters:

`rows` - an `int` specifying the number of rows to move from the current row; a positive number moves the cursor forward; a negative number moves the cursor backward

Returns:

`true` if the cursor is on a row; `false` otherwise

Throws:

`SQLException` - if a database access error occurs, there is no current row, or the result set type is `TYPE_FORWARD_ONLY`

Since:

1.2

---

previous

```
public boolean previous()
                throws SQLException
```

Moves the cursor to the previous row in this `ResultSet` object.

Returns:

`true` if the cursor is on a valid row; `false` if it is off the result set

Throws:

`SQLException` - if a database access error occurs or the result set type is `TYPE_FORWARD_ONLY`

Since:

1.2

---

setFetchDirection

```
public void setFetchDirection(int direction)
                       throws SQLException
```

Gives a hint as to the direction in which the rows in this `ResultSet` object will be processed. The initial value is determined by the `Statement` object that produced this `ResultSet` object. The fetch direction may be changed at any time.

Parameters:

`direction` - an `int` specifying the suggested fetch direction; one of `ResultSet.FETCH_FORWARD`, `ResultSet.FETCH_REVERSE`, or `ResultSet.FETCH_UNKNOWN`

Throws:

`SQLException` - if a database access error occurs or the result set type is `TYPE_FORWARD_ONLY` and the fetch direction is not `FETCH_FORWARD`

Since:

1.2

See Also:

`Statement.setFetchDirection(int)`, `getFetchDirection()`

---

getFetchDirection

```
public int getFetchDirection()
                      throws SQLException
```

Retrieves the fetch direction for this `ResultSet` object.

Returns:

the current fetch direction for this `ResultSet` object

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

See Also:

setFetchDirection(int)

---

setFetchSize

```
public void setFetchSize(int rows)
                  throws SQLException
```

Gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed for this `ResultSet` object. If the fetch size specified is zero, the JDBC driver ignores the value and is free to make its own best guess as to what the fetch size should be. The default value is set by the `Statement` object that created the result set. The fetch size may be changed at any time.

Parameters:

`rows` - the number of rows to fetch

Throws:

SQLException - if a database access error occurs or the condition `0 <= rows <= Statement.getMaxRows()` is not satisfied

Since:

1.2

See Also:

getFetchSize()

---

getFetchSize

```
public int getFetchSize()
                throws SQLException
```

Retrieves the fetch size for this `ResultSet` object.

Returns:

the current fetch size for this `ResultSet` object

Throws:

SQLException - if a database access error occurs

Since:

1.2

See Also:

setFetchSize(int)

---

getType

```
public int getType()
           throws SQLException
```

Retrieves the type of this `ResultSet` object. The type is determined by the `Statement` object that created the result set.

Returns:
`ResultSet.TYPE_FORWARD_ONLY`, `ResultSet.TYPE_SCROLL_INSENSITIVE`, or `ResultSet.TYPE_SCROLL_SENSITIVE`

Throws:
`SQLException` - if a database access error occurs

Since:
1.2

---

getConcurrency

```
public int getConcurrency()
                  throws SQLException
```

Retrieves the concurrency mode of this `ResultSet` object. The concurrency used is determined by the `Statement` object that created the result set.

Returns:
the concurrency type, either `ResultSet.CONCUR_READ_ONLY` or `ResultSet.CONCUR_UPDATABLE`

Throws:
`SQLException` - if a database access error occurs

Since:
1.2

---

rowUpdated

```
public boolean rowUpdated()
                  throws SQLException
```

Retrieves whether the current row has been updated. The value returned depends on whether or not the result set can detect updates.

Returns:

`true` if both (1) the row has been visibly updated by the owner or another and (2) updates are detected

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

See Also:

`DatabaseMetaData.updatesAreDetected(int)`

---

rowInserted

```
public boolean rowInserted()
                    throws SQLException
```

Retrieves whether the current row has had an insertion. The value returned depends on whether or not this `ResultSet` object can detect visible inserts.

Returns:

`true` if a row has had an insertion and insertions are detected; `false` otherwise

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

See Also:

`DatabaseMetaData.insertsAreDetected(int)`

---

rowDeleted

```
public boolean rowDeleted()
                   throws SQLException
```

Retrieves whether a row has been deleted. A deleted row may leave a visible "hole" in a result set. This method can be used to detect holes in a result set. The value returned depends on whether or not this `ResultSet` object can detect deletions.

Returns:

`true` if a row was deleted and deletions are detected; `false` otherwise

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

See Also:

`DatabaseMetaData.deletesAreDetected(int)`

---

updateNull

```
public void updateNull(int columnIndex)
                throws SQLException
```

Gives a nullable column a null value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateBoolean

```
public void updateBoolean(int columnIndex,
                          boolean x)
                   throws SQLException
```

Updates the designated column with a `boolean` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

`x` - the new column value

Throws:

SQLException - if a database access error occurs

Since:

1.2

---

updateByte

```
public void updateByte(int columnIndex,
                       byte x)
                throws SQLException
```

Updates the designated column with a `byte` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

SQLException - if a database access error occurs

Since:

1.2

---

updateShort

```
public void updateShort(int columnIndex,
                        short x)
                 throws SQLException
```

Updates the designated column with a `short` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

SQLException - if a database access error occurs

Since:

1.2

---

updateInt

```
public void updateInt(int columnIndex,
                      int x)
               throws SQLException
```

Updates the designated column with an `int` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateLong

```
public void updateLong(int columnIndex,
                       long x)
                throws SQLException
```

Updates the designated column with a `long` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

updateFloat

```
public void updateFloat(int columnIndex,
                        float x)
                 throws SQLException
```

Updates the designated column with a `float` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateDouble

```
public void updateDouble(int columnIndex,
                         double x)
                  throws SQLException
```

Updates the designated column with a `double` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateBigDecimal

```
public void updateBigDecimal(int columnIndex,
                             BigDecimal x)
                      throws SQLException
```

Updates the designated column with a `java.math.BigDecimal` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateString

```
public void updateString(int columnIndex,
                         String x)
                  throws SQLException
```

Updates the designated column with a `String` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateBytes

```
public void updateBytes(int columnIndex,
                        byte[] x)
                 throws SQLException
```

Updates the designated column with a `byte` array value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateDate

```
public void updateDate(int columnIndex,
                       Date x)
                throws SQLException
```

Updates the designated column with a `java.sql.Date` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateTime

```
public void updateTime(int columnIndex,
                       Time x)
```

```
                throws SQLException
```

Updates the designated column with a `java.sql.Time` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateTimestamp

```
public void updateTimestamp(int columnIndex,
                            Timestamp x)
                     throws SQLException
```

Updates the designated column with a `java.sql.Timestamp` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateAsciiStream

```
public void updateAsciiStream(int columnIndex,
                              InputStream x,
                              int length)
                       throws SQLException
```

Updates the designated column with an ascii stream value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value
`length` - the length of the stream

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateBinaryStream

```
public void updateBinaryStream(int columnIndex,
                               InputStream x,
                               int length)
                        throws SQLException
```

Updates the designated column with a binary stream value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value
`length` - the length of the stream

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateCharacterStream

```
public void updateCharacterStream(int columnIndex,
                                  Reader x,
```

```
                    int length)
             throws SQLException
```

Updates the designated column with a character stream value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value
`length` - the length of the stream

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateObject

```
public void updateObject(int columnIndex,
                         Object x,
                         int scale)
                  throws SQLException
```

Updates the designated column with an `Object` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value
`scale` - for `java.sql.Types.DECIMA` or `java.sql.Types.NUMERIC` types, this is the number of digits after the decimal point. For all other types this value will be ignored.

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateObject

```
public void updateObject(int columnIndex,
                         Object x)
                  throws SQLException
```

Updates the designated column with an `Object` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateNull

```
public void updateNull(String columnName)
                throws SQLException
```

Updates the designated column with a `null` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateBoolean

```
public void updateBoolean(String columnName,
```

```
                    boolean x)
             throws SQLException
```

Updates the designated column with a `boolean` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateByte

```
public void updateByte(String columnName,
                       byte x)
                throws SQLException
```

Updates the designated column with a `byte` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateShort

```
public void updateShort(String columnName,
                        short x)
                 throws SQLException
```

Updates the designated column with a `short` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateInt

```
public void updateInt(String columnName,
                      int x)
               throws SQLException
```

Updates the designated column with an `int` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateLong

```
public void updateLong(String columnName,
                       long x)
                throws SQLException
```

Updates the designated column with a `long` value. The updater methods are used to update

column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateFloat

```
public void updateFloat(String columnName,
                        float x)
                 throws SQLException
```

Updates the designated column with a `float` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateDouble

```
public void updateDouble(String columnName,
                         double x)
                  throws SQLException
```

Updates the designated column with a `double` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update

the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateBigDecimal

```
public void updateBigDecimal(String columnName,
                             BigDecimal x)
                      throws SQLException
```

Updates the designated column with a `java.sql.BigDecimal` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateString

```
public void updateString(String columnName,
                         String x)
                  throws SQLException
```

Updates the designated column with a `String` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column

`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateBytes

```
public void updateBytes(String columnName,
                        byte[] x)
                 throws SQLException
```

Updates the designated column with a byte array value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column

`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateDate

```
public void updateDate(String columnName,
                       Date x)
                throws SQLException
```

Updates the designated column with a `java.sql.Date` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateTime

```
public void updateTime(String columnName,
                       Time x)
                throws SQLException
```

Updates the designated column with a `java.sql.Time` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateTimestamp

```
public void updateTimestamp(String columnName,
                            Timestamp x)
                     throws SQLException
```

Updates the designated column with a `java.sql.Timestamp` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

SQLException - if a database access error occurs

Since:

1.2

---

updateAsciiStream

```
public void updateAsciiStream(String columnName,
                              InputStream x,
                              int length)
                       throws SQLException
```

Updates the designated column with an ascii stream value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value
`length` - the length of the stream

Throws:

SQLException - if a database access error occurs

Since:

1.2

---

updateBinaryStream

```
public void updateBinaryStream(String columnName,
                               InputStream x,
                               int length)
                        throws SQLException
```

Updates the designated column with a binary stream value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column

`x` - the new column value
`length` - the length of the stream

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateCharacterStream

```
public void updateCharacterStream(String columnName,
                                  Reader reader,
                                  int length)
                           throws SQLException
```

Updates the designated column with a character stream value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`reader` - the `java.io.Reader` object containing the new column value
`length` - the length of the stream

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateObject

```
public void updateObject(String columnName,
                         Object x,
                         int scale)
                  throws SQLException
```

Updates the designated column with an `Object` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value
`scale` - for `java.sql.Types.DECIMAL` or `java.sql.Types.NUMERIC` types, this is the number of digits after the decimal point. For all other types this value will be ignored.

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

updateObject

```
public void updateObject(String columnName,
                         Object x)
                  throws SQLException
```

Updates the designated column with an `Object` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

insertRow

```
public void insertRow()
                throws SQLException
```

Inserts the contents of the insert row into this `ResultSet` object and into the database. The cursor must be on the insert row when this method is called.

Throws:

SQLException - if a database access error occurs, if this method is called when the cursor is not on the insert row, or if not all of non-nullable columns in the insert row have been given a value

Since:

1.2

---

updateRow

```
public void updateRow()
                throws SQLException
```

Updates the underlying database with the new contents of the current row of this `ResultSet` object. This method cannot be called when the cursor is on the insert row.

Throws:

SQLException - if a database access error occurs or if this method is called when the cursor is on the insert row

Since:

1.2

---

deleteRow

```
public void deleteRow()
                throws SQLException
```

Deletes the current row from this `ResultSet` object and from the underlying database. This method cannot be called when the cursor is on the insert row.

Throws:

SQLException - if a database access error occurs or if this method is called when the cursor is on the insert row

Since:

1.2

---

refreshRow

```
public void refreshRow()
```

```
                throws SQLException
```

Refreshes the current row with its most recent value in the database. This method cannot be called when the cursor is on the insert row.

The `refreshRow` method provides a way for an application to explicitly tell the JDBC driver to refetch a row(s) from the database. An application may want to call `refreshRow` when caching or prefetching is being done by the JDBC driver to fetch the latest value of a row from the database. The JDBC driver may actually refresh multiple rows at once if the fetch size is greater than one.

All values are refetched subject to the transaction isolation level and cursor sensitivity. If `refreshRow` is called after calling an updater method, but before calling the method `updateRow`, then the updates made to the row are lost. Calling the method `refreshRow` frequently will likely slow performance.

Throws:

SQLException - if a database access error occurs or if this method is called when the cursor is on the insert row

Since:

1.2

---

cancelRowUpdates

```
public void cancelRowUpdates()
                    throws SQLException
```

Cancels the updates made to the current row in this `ResultSet` object. This method may be called after calling an updater method(s) and before calling the method `updateRow` to roll back the updates made to a row. If no updates have been made or `updateRow` has already been called, this method has no effect.

Throws:

SQLException - if a database access error occurs or if this method is called when the cursor is on the insert row

Since:

1.2

---

moveToInsertRow

```
public void moveToInsertRow()
                    throws SQLException
```

Moves the cursor to the insert row. The current cursor position is remembered while the cursor is positioned on the insert row. The insert row is a special row associated with an updatable result set. It is essentially a buffer where a new row may be constructed by calling the updater methods prior to inserting the row into the result set. Only the updater, getter, and `insertRow` methods may be called when the cursor is on the insert row. All of the columns in a result set must be given a value each time this method is called before calling `insertRow`. An updater method must be called before a getter method can be called on a column value.

Throws:

`SQLException` - if a database access error occurs or the result set is not updatable

Since:

1.2

---

moveToCurrentRow

```
public void moveToCurrentRow()
                     throws SQLException
```

Moves the cursor to the remembered cursor position, usually the current row. This method has no effect if the cursor is not on the insert row.

Throws:

`SQLException` - if a database access error occurs or the result set is not updatable

Since:

1.2

---

getStatement

```
public Statement getStatement()
                      throws SQLException
```

Retrieves the `Statement` object that produced this `ResultSet` object. If the result set was generated some other way, such as by a `DatabaseMetaData` method, this method returns `null`.

Returns:

the `Statment` object that produced this `ResultSet` object or `null` if the result set was produced some other way

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getObject

```
public Object getObject(int i,
                        Map map)
                 throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as an `Object` in the Java programming language. If the value is an SQL `NULL`, the driver returns a Java `null`. This method uses the given `Map` object for the custom mapping of the SQL structured or distinct type that is being retrieved.

Parameters:

`i` - the first column is 1, the second is 2, ...

`map` - a `java.util.Map` object that contains the mapping from SQL type names to classes in the Java programming language

Returns:

an `Object` in the Java programming language representing the SQL value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getRef

```
public Ref getRef(int i)
           throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `Ref` object in the Java programming language.

Parameters:

`i` - the first column is 1, the second is 2, ...

Returns:

a `Ref` object representing an SQL `REF` value

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getBlob

```
public Blob getBlob(int i)
            throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `Blob` object in the Java programming language.

Parameters:

`i` - the first column is 1, the second is 2, ...

Returns:

a `Blob` object representing the SQL `BLOB` value in the specified column

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getClob

```
public Clob getClob(int i)
            throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `Clob` object in the Java programming language.

Parameters:

`i` - the first column is 1, the second is 2, ...

Returns:

a `Clob` object representing the SQL `CLOB` value in the specified column

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getArray

```
public Array getArray(int i)
               throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as an `Array` object in the Java programming language.

Parameters:
`i` - the first column is 1, the second is 2, ...
Returns:
an `Array` object representing the SQL `ARRAY` value in the specified column
Throws:
`SQLException` - if a database access error occurs
Since:
1.2

---

getObject

```
public Object getObject(String colName,
                        Map map)
                 throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as an `Object` in the Java programming language. If the value is an SQL `NULL`, the driver returns a Java `null`. This method uses the specified `Map` object for custom mapping if appropriate.

Parameters:
`colName` - the name of the column from which to retrieve the value
`map` - a `java.util.Map` object that contains the mapping from SQL type names to classes in the Java programming language
Returns:
an `Object` representing the SQL value in the specified column
Throws:
`SQLException` - if a database access error occurs
Since:
1.2

---

getRef

```
public Ref getRef(String colName)
           throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `Ref` object in the Java programming language.

Parameters:
`colName` - the column name
Returns:
a `Ref` object representing the SQL `REF` value in the specified column
Throws:
`SQLException` - if a database access error occurs
Since:
1.2

---

getBlob

```
public Blob getBlob(String colName)
            throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `Blob` object in the Java programming language.

Parameters:
`colName` - the name of the column from which to retrieve the value
Returns:
a `Blob` object representing the SQL `BLOB` value in the specified column
Throws:
`SQLException` - if a database access error occurs
Since:
1.2

---

getClob

```
public Clob getClob(String colName)
            throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `Clob` object in the Java programming language.

Parameters:
`colName` - the name of the column from which to retrieve the value
Returns:
a `Clob` object representing the SQL `CLOB` value in the specified column
Throws:
`SQLException` - if a database access error occurs
Since:
1.2

---

getArray

```
public Array getArray(String colName)
               throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as an `Array` object in the Java programming language.

Parameters:
`colName` - the name of the column from which to retrieve the value
Returns:
an `Array` object representing the SQL ARRAY value in the specified column
Throws:
`SQLException` - if a database access error occurs
Since:
1.2

---

getDate

```
public Date getDate(int columnIndex,
                    Calendar cal)
             throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.Date` object in the Java programming language. This method uses the given calendar to construct an appropriate millisecond value for the date if the underlying database does not store timezone information.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

`cal` - the `java.util.Calendar` object to use in constructing the date

Returns:

the column value as a `java.sql.Date` object; if the value is SQL `NULL`, the value returned is `null` in the Java programming language

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getDate

```
public Date getDate(String columnName,
                    Calendar cal)
             throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.Date` object in the Java programming language. This method uses the given calendar to construct an appropriate millisecond value for the date if the underlying database does not store timezone information.

Parameters:

`columnName` - the SQL name of the column from which to retrieve the value

`cal` - the `java.util.Calendar` object to use in constructing the date

Returns:

the column value as a `java.sql.Date` object; if the value is SQL `NULL`, the value returned is `null` in the Java programming language

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getTime

```
public Time getTime(int columnIndex,
                    Calendar cal)
             throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.Time` object in the Java programming language. This method uses the given calendar to construct an appropriate millisecond value for the time if the underlying database does not store timezone information.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`cal` - the `java.util.Calendar` object to use in constructing the time

Returns:

the column value as a `java.sql.Time` object; if the value is SQL `NULL`, the value returned is `null` in the Java programming language

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getTime

```
public Time getTime(String columnName,
                    Calendar cal)
             throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.Time` object in the Java programming language. This method uses the given calendar to construct an appropriate millisecond value for the time if the underlying database does not store timezone information.

Parameters:

`columnName` - the SQL name of the column
`cal` - the `java.util.Calendar` object to use in constructing the time

Returns:

the column value as a `java.sql.Time` object; if the value is SQL `NULL`, the value returned is `null` in the Java programming language

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getTimestamp

```
public Timestamp getTimestamp(int columnIndex,
                              Calendar cal)
                       throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.Timestamp` object in the Java programming language. This method uses the given calendar to construct an appropriate millisecond value for the timestamp if the underlying database does not store timezone information.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...

`cal` - the `java.util.Calendar` object to use in constructing the timestamp

Returns:

the column value as a `java.sql.Timestamp` object; if the value is SQL `NULL`, the value returned is `null` in the Java programming language

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getTimestamp

```
public Timestamp getTimestamp(String columnName,
                              Calendar cal)
                       throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.sql.Timestamp` object in the Java programming language. This method uses the given calendar to construct an appropriate millisecond value for the timestamp if the underlying database does not store timezone information.

Parameters:

`columnName` - the SQL name of the column

`cal` - the `java.util.Calendar` object to use in constructing the date

Returns:

the column value as a `java.sql.Timestamp` object; if the value is SQL `NULL`, the value returned is `null` in the Java programming language

Throws:

`SQLException` - if a database access error occurs

Since:

1.2

---

getURL

```
public URL getURL(int columnIndex)
           throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.net.URL` object in the Java programming language.

Parameters:
`columnIndex` - the index of the column 1 is the first, 2 is the second,...
Returns:
the column value as a `java.net.URL` object; if the value is SQL `NULL`, the value returned is `null` in the Java programming language
Throws:
`SQLException` - if a database access error occurs, or if a URL is malformed
Since:
1.4

---

getURL

```
public URL getURL(String columnName)
           throws SQLException
```

Retrieves the value of the designated column in the current row of this `ResultSet` object as a `java.net.URL` object in the Java programming language.

Parameters:
`columnName` - the SQL name of the column
Returns:
the column value as a `java.net.URL` object; if the value is SQL `NULL`, the value returned is `null` in the Java programming language
Throws:
`SQLException` - if a database access error occurs or if a URL is malformed
Since:
1.4

---

updateRef

```
public void updateRef(int columnIndex,
                      Ref x)
               throws SQLException
```

Updates the designated column with a `java.sql.Ref` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

updateRef

```
public void updateRef(String columnName,
                      Ref x)
               throws SQLException
```

Updates the designated column with a `java.sql.Ref` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

updateBlob

```
public void updateBlob(int columnIndex,
                       Blob x)
```

```
                throws SQLException
```

Updates the designated column with a `java.sql.Blob` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

updateBlob

```
public void updateBlob(String columnName,
                       Blob x)
                throws SQLException
```

Updates the designated column with a `java.sql.Blob` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

updateClob

```
public void updateClob(int columnIndex,
                       Clob x)
                throws SQLException
```

Updates the designated column with a `java.sql.Clob` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

updateClob

```
public void updateClob(String columnName,
                       Clob x)
                throws SQLException
```

Updates the designated column with a `java.sql.Clob` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

updateArray

```
public void updateArray(int columnIndex,
                        Array x)
                 throws SQLException
```

Updates the designated column with a `java.sql.Array` value. The updater methods are

used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnIndex` - the first column is 1, the second is 2, ...
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

updateArray

```
public void updateArray(String columnName,
                        Array x)
                 throws SQLException
```

Updates the designated column with a `java.sql.Array` value. The updater methods are used to update column values in the current row or the insert row. The updater methods do not update the underlying database; instead the `updateRow` or `insertRow` methods are called to update the database.

Parameters:

`columnName` - the name of the column
`x` - the new column value

Throws:

`SQLException` - if a database access error occurs

Since:

1.4

---

documentation redistribution policy.

java.sql

Class SQLWarning

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.sql.SQLException
              └ java.sql.SQLWarning
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

DataTruncation

---

public class **SQLWarning**
extends SQLException

An exception that provides information on database access warnings. Warnings are silently chained to the object whose method caused it to be reported.

Warnings may be retrieved from `Connection`, `Statement`, and `ResultSet` objects. Trying to retrieve a warning on a connection after it has been closed will cause an exception to be thrown. Similarly, trying to retrieve a warning on a statement after it has been closed or on a result set after it has been closed will cause an exception to be thrown. Note that closing a statement also closes a result set that it might have produced.

See Also:

`Connection.getWarnings()`, `Statement.getWarnings()`, `ResultSet.getWarnings()`, Serialized Form

Constructor Summary

SQLWarning()	Constructs a default SQLWarning object.
SQLWarning(String reason)	Constructs an SQLWarning object with the given value for a reason; SQLstate defaults to null, and vendorCode defaults to 0.
SQLWarning(String reason, String SQLstate)	Constructs an SQLWarning object with the given reason and SQLState; the vendorCode defaults to 0.
SQLWarning(String reason, String SQLstate, int vendorCode)	Constructs a fully-specified SQLWarning object initialized with the given values.

Method Summary

SQLWarning	**getNextWarning**() Retrieves the warning chained to this SQLWarning object.
void	**setNextWarning**(SQLWarning w) Adds an SQLWarning object to the end of the chain.

Methods inherited from class java.sql.SQLException

getErrorCode, getNextException, getSQLState, setNextException

Methods inherited from class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

SQLWarning

```
public SQLWarning(String reason,
```

```
                String SQLstate,
                int vendorCode)
```

Constructs a fully-specified `SQLWarning` object initialized with the given values.

Parameters:

`reason` - a description of the warning
`SQLstate` - an XOPEN code identifying the warning
`vendorCode` - a database vendor-specific warning code

SQLWarning

```
public SQLWarning(String reason,
                  String SQLstate)
```

Constructs an `SQLWarning` object with the given reason and SQLState; the vendorCode defaults to 0.

Parameters:

`reason` - a description of the warning
`SQLstate` - an XOPEN code identifying the warning

SQLWarning

```
public SQLWarning(String reason)
```

Constructs an `SQLWarning` object with the given value for a reason; SQLstate defaults to `null`, and vendorCode defaults to 0.

Parameters:

`reason` - a description of the warning

SQLWarning

```
public SQLWarning()
```

Constructs a default `SQLWarning` object. The reason defaults to `null`, SQLState defaults to

`null`, and vendorCode defaults to 0.

Method Detail

getNextWarning

```
public SQLWarning getNextWarning()
```

Retrieves the warning chained to this `SQLWarning` object.

Returns:
the next `SQLException` in the chain; `null` if none

See Also:
`setNextWarning(java.sql.SQLWarning)`

---

setNextWarning

```
public void setNextWarning(SQLWarning w)
```

Adds an `SQLWarning` object to the end of the chain.

Parameters:
`w` - the new end of the `SQLException` chain

See Also:
`getNextWarning()`

---

java.sql

Class SQLException

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.sql.SQLException
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

BatchUpdateException, RowSetWarning, SerialException, SQLWarning, SyncFactoryException, SyncProviderException

---

public class **SQLException**
extends Exception

An exception that provides information on a database access error or other errors.

Each `SQLException` provides several kinds of information:

- a string describing the error. This is used as the Java Exception message, available via the method `getMesage`.
- a "SQLstate" string, which follows either the XOPEN SQLstate conventions or the SQL 99 conventions. The values of the SQLState string are described in the appropriate spec. The `DatabaseMetaData` method `getSQLStateType` can be used to discover whether the driver returns the XOPEN type or the SQL 99 type.
- an integer error code that is specific to each vendor. Normally this will be the actual error code returned by the underlying database.
- a chain to a next Exception. This can be used to provide additional error information.

See Also:

Serialized Form

Constructor Summary

SQLException() Constructs an SQLException object; the reason field defaults to null, the SQLState field defaults to null, and the vendorCode field defaults to 0.
SQLException(String reason) Constructs an SQLException object with a reason; the SQLState field defaults to null, and the vendorCode field defaults to 0.
SQLException(String reason, String SQLState) Constructs an SQLException object with the given reason and SQLState; the vendorCode field defaults to 0.
SQLException(String reason, String SQLState, int vendorCode) Constructs a fully-specified SQLException object.

Method Summary

int	**getErrorCode**() Retrieves the vendor-specific exception code for this SQLException object.
SQLException	**getNextException**() Retrieves the exception chained to this SQLException object.
String	**getSQLState**() Retrieves the SQLState for this SQLException object.
void	**setNextException**(SQLException ex) Adds an SQLException object to the end of the chain.

Methods inherited from class java.lang.Throwable

fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

SQLException

```
public SQLException(String reason,
                    String SQLState,
                    int vendorCode)
```

Constructs a fully-specified `SQLException` object.

Parameters:

`reason` - a description of the exception
`SQLState` - an XOPEN or SQL 99 code identifying the exception
`vendorCode` - a database vendor-specific exception code

SQLException

```
public SQLException(String reason,
                    String SQLState)
```

Constructs an `SQLException` object with the given reason and SQLState; the `vendorCode` field defaults to 0.

Parameters:

`reason` - a description of the exception
`SQLState` - an XOPEN or SQL 99 code identifying the exception

SQLException

```
public SQLException(String reason)
```

Constructs an `SQLException` object with a reason; the `SQLState` field defaults to `null`, and the `vendorCode` field defaults to 0.

Parameters:

`reason` - a description of the exception

SQLException

```
public SQLException()
```

Constructs an `SQLException` object; the `reason` field defaults to null, the `SQLState` field defaults to `null`, and the `vendorCode` field defaults to 0.

Method Detail

getSQLState

```
public String getSQLState()
```

Retrieves the SQLState for this `SQLException` object.

Returns:
the SQLState value

---

getErrorCode

```
public int getErrorCode()
```

Retrieves the vendor-specific exception code for this `SQLException` object.

Returns:
the vendor's error code

---

getNextException

```
public SQLException getNextException()
```

Retrieves the exception chained to this `SQLException` object.

Returns:
the next `SQLException` object in the chain; `null` if there are none

See Also:
`setNextException(java.sql.SQLException)`

setNextException

```
public void setNextException(SQLException ex)
```

Adds an `SQLException` object to the end of the chain.

Parameters:

`ex` - the new exception that will be added to the end of the `SQLException` chain

See Also:

`getNextException()`

```
try {
      // JDBC code
} catch (SQLException e) {
      while (e != null) {
            System.err.println("SQLState: " + e.getSQLState());
    System.err.println("Message:  " + e.getMessage());
    System.err.println("Vendor:   " + e.getErrorCode());
    System.err.println("-----");
    e = e.getNextException();
      }
}
```

---

java.lang

Class Throwable

java.lang.Object
  └ **java.lang.Throwable**

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

Error, Exception

---

public class **Throwable**
extends Object
implements Serializable

The `Throwable` class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java `throw` statement. Similarly, only this class or one of its subclasses can be the argument type in a `catch` clause.

Instances of two subclasses, `Error` and `Exception`, are conventionally used to indicate that exceptional situations have occurred. Typically, these instances are freshly created in the context of the exceptional situation so as to include relevant information (such as stack trace data).

A throwable contains a snapshot of the execution stack of its thread at the time it was created. It can also contain a message string that gives more information about the error. Finally, it can contain a *cause*: another throwable that caused this throwable to get thrown. The cause facility is new in release 1.4. It is also known as the *chained exception* facility, as the cause can, itself, have a cause, and so on, leading to a "chain" of exceptions, each caused by another.

One reason that a throwable may have a cause is that the class that throws it is built atop a lower layered abstraction, and an operation on the upper layer fails due to a failure in the lower layer. It would be bad design to let the throwable thrown by the lower layer propagate outward, as it is generally unrelated to the abstraction provided by the upper layer. Further, doing so would tie the API of the upper layer to the details of its implementation, assuming the lower layer's exception was

a checked exception. Throwing a "wrapped exception" (i.e., an exception containing a cause) allows the upper layer to communicate the details of the failure to its caller without incurring either of these shortcomings. It preserves the flexibility to change the implementation of the upper layer without changing its API (in particular, the set of exceptions thrown by its methods).

A second reason that a throwable may have a cause is that the method that throws it must conform to a general-purpose interface that does not permit the method to throw the cause directly. For example, suppose a persistent collection conforms to the `Collection` interface, and that its persistence is implemented atop `java.io`. Suppose the internals of the `put` method can throw an `IOException`. The implementation can communicate the details of the `IOException` to its caller while conforming to the `Collection` interface by wrapping the `IOException` in an appropriate unchecked exception. (The specification for the persistent collection should indicate that it is capable of throwing such exceptions.)

A cause can be associated with a throwable in two ways: via a constructor that takes the cause as an argument, or via the `initCause(Throwable)` method. New throwable classes that wish to allow causes to be associated with them should provide constructors that take a cause and delegate (perhaps indirectly) to one of the `Throwable` constructors that takes a cause. For example:

```
     try {
         lowLevelOp();
     } catch (LowLevelException le) {
         throw new HighLevelException(le);  // Chaining-aware
constructor
     }
```

Because the `initCause` method is public, it allows a cause to be associated with any throwable, even a "legacy throwable" whose implementation predates the addition of the exception chaining mechanism to `Throwable`. For example:

```
     try {
         lowLevelOp();
     } catch (LowLevelException le) {
         throw (HighLevelException)
                 new HighLevelException().initCause(le);  // Legacy
constructor
     }
```

Prior to release 1.4, there were many throwables that had their own non-standard exception chaining mechanisms (`ExceptionInInitializerError`, `ClassNotFoundException`, `UndeclaredThrowableException`, `InvocationTargetException`, `WriteAbortedException`, `PrivilegedActionException`, `PrinterIOException`, `RemoteException` and `NamingException`). All of these throwables have been retrofitted to

use the standard exception chaining mechanism, while continuing to implement their "legacy" chaining mechanisms for compatibility.

Further, as of release 1.4, many general purpose `Throwable` classes (for example `Exception`, `RuntimeException`, `Error`) have been retrofitted with constructors that take a cause. This was not strictly necessary, due to the existence of the `initCause` method, but it is more convenient and expressive to delegate to a constructor that takes a cause.

By convention, class `Throwable` and its subclasses have two constructors, one that takes no arguments and one that takes a `String` argument that can be used to produce a detail message. Further, those subclasses that might likely have a cause associated with them should have two more constructors, one that takes a `Throwable` (the cause), and one that takes a `String` (the detail message) and a `Throwable` (the cause).

Also introduced in release 1.4 is the `getStackTrace()` method, which allows programmatic access to the stack trace information that was previously available only in text form, via the various forms of the `printStackTrace()` method. This information has been added to the *serialized representation* of this class so `getStackTrace` and `printStackTrace` will operate properly on a throwable that was obtained by deserialization.

Since:

JDK1.0

See Also:

Serialized Form

Constructor Summary

`Throwable``()` Constructs a new throwable with `null` as its detail message.
`Throwable``(String message)` Constructs a new throwable with the specified detail message.
`Throwable``(String message, Throwable cause)` Constructs a new throwable with the specified detail message and cause.
`Throwable``(Throwable cause)` Constructs a new throwable with the specified cause and a detail message of `(cause==null ? null : cause.toString())` (which typically contains the class and detail message of `cause`).

Method Summary

Throwable	**fillInStackTrace**() Fills in the execution stack trace.
Throwable	**getCause**() Returns the cause of this throwable or null if the cause is nonexistent or unknown.
String	**getLocalizedMessage**() Creates a localized description of this throwable.
String	**getMessage**() Returns the detail message string of this throwable.
StackTraceElement[]	**getStackTrace**() Provides programmatic access to the stack trace information printed by printStackTrace().
Throwable	**initCause**(Throwable cause) Initializes the *cause* of this throwable to the specified value.
void	**printStackTrace**() Prints this throwable and its backtrace to the standard error stream.
void	**printStackTrace**(PrintStream s) Prints this throwable and its backtrace to the specified print stream.
void	**printStackTrace**(PrintWriter s) Prints this throwable and its backtrace to the specified print writer.
void	**setStackTrace**(StackTraceElement[] stackTrace) Sets the stack trace elements that will be returned by getStackTrace() and printed by printStackTrace() and related methods.
String	**toString**() Returns a short description of this throwable.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Throwable

`public Throwable()`

Constructs a new throwable with `null` as its detail message. The cause is not initialized, and may subsequently be initialized by a call to `initCause(java.lang.Throwable)`.

The `fillInStackTrace()` method is called to initialize the stack trace data in the newly created throwable.

Throwable

`public Throwable(String message)`

Constructs a new throwable with the specified detail message. The cause is not initialized, and may subsequently be initialized by a call to `initCause(java.lang.Throwable)`.

The `fillInStackTrace()` method is called to initialize the stack trace data in the newly created throwable.

Parameters:

`message` - the detail message. The detail message is saved for later retrieval by the `getMessage()` method.

Throwable

`public Throwable(String message, Throwable cause)`

Constructs a new throwable with the specified detail message and cause.

Note that the detail message associated with `cause` is *not* automatically incorporated in this throwable's detail message.

The `fillInStackTrace()` method is called to initialize the stack trace data in the newly created throwable.

Parameters:

`message` - the detail message (which is saved for later retrieval by the `getMessage()` method).

`cause` - the cause (which is saved for later retrieval by the `getCause()` method). (A `null` value is permitted, and indicates that the cause is nonexistent or unknown.)

Since:

1.4

Throwable

`public Throwable(Throwable cause)`

Constructs a new throwable with the specified cause and a detail message of `(cause==null ? null : cause.toString())` (which typically contains the class and detail message of `cause`). This constructor is useful for throwables that are little more than wrappers for other throwables (for example, `PrivilegedActionException`).

The `fillInStackTrace()` method is called to initialize the stack trace data in the newly created throwable.

Parameters:

`cause` - the cause (which is saved for later retrieval by the `getCause()` method). (A `null` value is permitted, and indicates that the cause is nonexistent or unknown.)

Since:

1.4

Method Detail

getMessage

`public String getMessage()`

Returns the detail message string of this throwable.

Returns:

the detail message string of this `Throwable` instance (which may be `null`).

---

getLocalizedMessage

`public String getLocalizedMessage()`

Creates a localized description of this throwable. Subclasses may override this method in order to produce a locale-specific message. For subclasses that do not override this method, the default implementation returns the same result as `getMessage()`.

Returns:
The localized description of this throwable.
Since:
JDK1.1

---

getCause

`public Throwable getCause()`

Returns the cause of this throwable or `null` if the cause is nonexistent or unknown. (The cause is the throwable that caused this throwable to get thrown.)

This implementation returns the cause that was supplied via one of the constructors requiring a `Throwable`, or that was set after creation with the `initCause(Throwable)` method. While it is typically unnecessary to override this method, a subclass can override it to return a cause set by some other means. This is appropriate for a "legacy chained throwable" that predates the addition of chained exceptions to `Throwable`. Note that it is *not* necessary to override any of the `PrintStackTrace` methods, all of which invoke the `getCause` method to determine the cause of a throwable.

Returns:
the cause of this throwable or `null` if the cause is nonexistent or unknown.
Since:
1.4

---

initCause

`public Throwable initCause(Throwable cause)`

Initializes the *cause* of this throwable to the specified value. (The cause is the throwable that caused this throwable to get thrown.)

This method can be called at most once. It is generally called from within the constructor, or immediately after creating the throwable. If this throwable was created with `Throwable(Throwable)` or `Throwable(String,Throwable)`, this method cannot be called even once.

Parameters:

`cause` - the cause (which is saved for later retrieval by the `getCause()` method). (A `null` value is permitted, and indicates that the cause is nonexistent or unknown.)

Returns:

a reference to this `Throwable` instance.

Throws:

`IllegalArgumentException` - if `cause` is this throwable. (A throwable cannot be its own cause.)

`IllegalStateException` - if this throwable was created with `Throwable(Throwable)` or `Throwable(String,Throwable)`, or this method has already been called on this throwable.

Since:

1.4

---

toString

```
public String toString()
```

Returns a short description of this throwable. If this `Throwable` object was created with a non-null detail message string, then the result is the concatenation of three strings:

- The name of the actual class of this object
- ": " (a colon and a space)
- The result of the `getMessage()` method for this object

If this `Throwable` object was created with a `null` detail message string, then the name of the actual class of this object is returned.

Overrides:

`toString` in class `Object`

Returns:

a string representation of this throwable.

---

printStackTrace

```
public void printStackTrace()
```

Prints this throwable and its backtrace to the standard error stream. This method prints a stack trace for this `Throwable` object on the error output stream that is the value of the field `System.err`. The first line of output contains the result of the `toString()` method for this object. Remaining lines represent data previously recorded by the method

fillInStackTrace(). The format of this information depends on the implementation, but the following example may be regarded as typical:

```
java.lang.NullPointerException
        at MyClass.mash(MyClass.java:9)
        at MyClass.crunch(MyClass.java:6)
        at MyClass.main(MyClass.java:3)
```

This example was produced by running the program:

```
class MyClass {
    public static void main(String[] args) {
        crunch(null);
    }
    static void crunch(int[] a) {
        mash(a);
    }
    static void mash(int[] b) {
        System.out.println(b[0]);
    }
}
```

The backtrace for a throwable with an initialized, non-null cause should generally include the backtrace for the cause. The format of this information depends on the implementation, but the following example may be regarded as typical:

```
HighLevelException: MidLevelException: LowLevelException
        at Junk.a(Junk.java:13)
        at Junk.main(Junk.java:4)
Caused by: MidLevelException: LowLevelException
        at Junk.c(Junk.java:23)
        at Junk.b(Junk.java:17)
        at Junk.a(Junk.java:11)
        ... 1 more
Caused by: LowLevelException
        at Junk.e(Junk.java:30)
        at Junk.d(Junk.java:27)
        at Junk.c(Junk.java:21)
        ... 3 more
```

Note the presence of lines containing the characters "...". These lines indicate that the remainder of the stack trace for this exception matches the indicated number of frames from the bottom of the stack trace of the exception that was caused by this exception (the "enclosing" exception). This

shorthand can greatly reduce the length of the output in the common case where a wrapped exception is thrown from same method as the "causative exception" is caught. The above example was produced by running the program:

```
public class Junk {
    public static void main(String args[]) {
        try {
            a();
        } catch(HighLevelException e) {
            e.printStackTrace();
        }
    }
    static void a() throws HighLevelException {
        try {
            b();
        } catch(MidLevelException e) {
            throw new HighLevelException(e);
        }
    }
    static void b() throws MidLevelException {
        c();
    }
    static void c() throws MidLevelException {
        try {
            d();
        } catch(LowLevelException e) {
            throw new MidLevelException(e);
        }
    }
    static void d() throws LowLevelException {
       e();
    }
    static void e() throws LowLevelException {
        throw new LowLevelException();
    }
}

class HighLevelException extends Exception {
    HighLevelException(Throwable cause) { super(cause); }
}

class MidLevelException extends Exception {
    MidLevelException(Throwable cause)  { super(cause); }
}

class LowLevelException extends Exception {
```

```
}
```

---

printStackTrace

```
public void printStackTrace(PrintStream s)
```

Prints this throwable and its backtrace to the specified print stream.

Parameters:
`s` - `PrintStream` to use for output

---

printStackTrace

```
public void printStackTrace(PrintWriter s)
```

Prints this throwable and its backtrace to the specified print writer.

Parameters:
`s` - `PrintWriter` to use for output
Since:
JDK1.1

---

fillInStackTrace

```
public Throwable fillInStackTrace()
```

Fills in the execution stack trace. This method records within this `Throwable` object information about the current state of the stack frames for the current thread.

Returns:
a reference to this `Throwable` instance.
See Also:
`printStackTrace()`

---

getStackTrace

`public StackTraceElement[] getStackTrace()`

Provides programmatic access to the stack trace information printed by `printStackTrace()`. Returns an array of stack trace elements, each representing one stack frame. The zeroth element of the array (assuming the array's length is non-zero) represents the top of the stack, which is the last method invocation in the sequence. Typically, this is the point at which this throwable was created and thrown. The last element of the array (assuming the array's length is non-zero) represents the bottom of the stack, which is the first method invocation in the sequence.

Some virtual machines may, under some circumstances, omit one or more stack frames from the stack trace. In the extreme case, a virtual machine that has no stack trace information concerning this throwable is permitted to return a zero-length array from this method. Generally speaking, the array returned by this method will contain one element for every frame that would be printed by `printStackTrace`.

Returns:

an array of stack trace elements representing the stack trace pertaining to this throwable.

Since:

1.4

---

setStackTrace

`public void setStackTrace(StackTraceElement[] stackTrace)`

Sets the stack trace elements that will be returned by `getStackTrace()` and printed by `printStackTrace()` and related methods. This method, which is designed for use by RPC frameworks and other advanced systems, allows the client to override the default stack trace that is either generated by `fillInStackTrace()` when a throwable is constructed or deserialized when a throwable is read from a serialization stream.

Parameters:

`stackTrace` - the stack trace elements to be associated with this `Throwable`. The specified array is copied by this call; changes in the specified array after the method invocation returns will have no affect on this `Throwable`'s stack trace.

Throws:

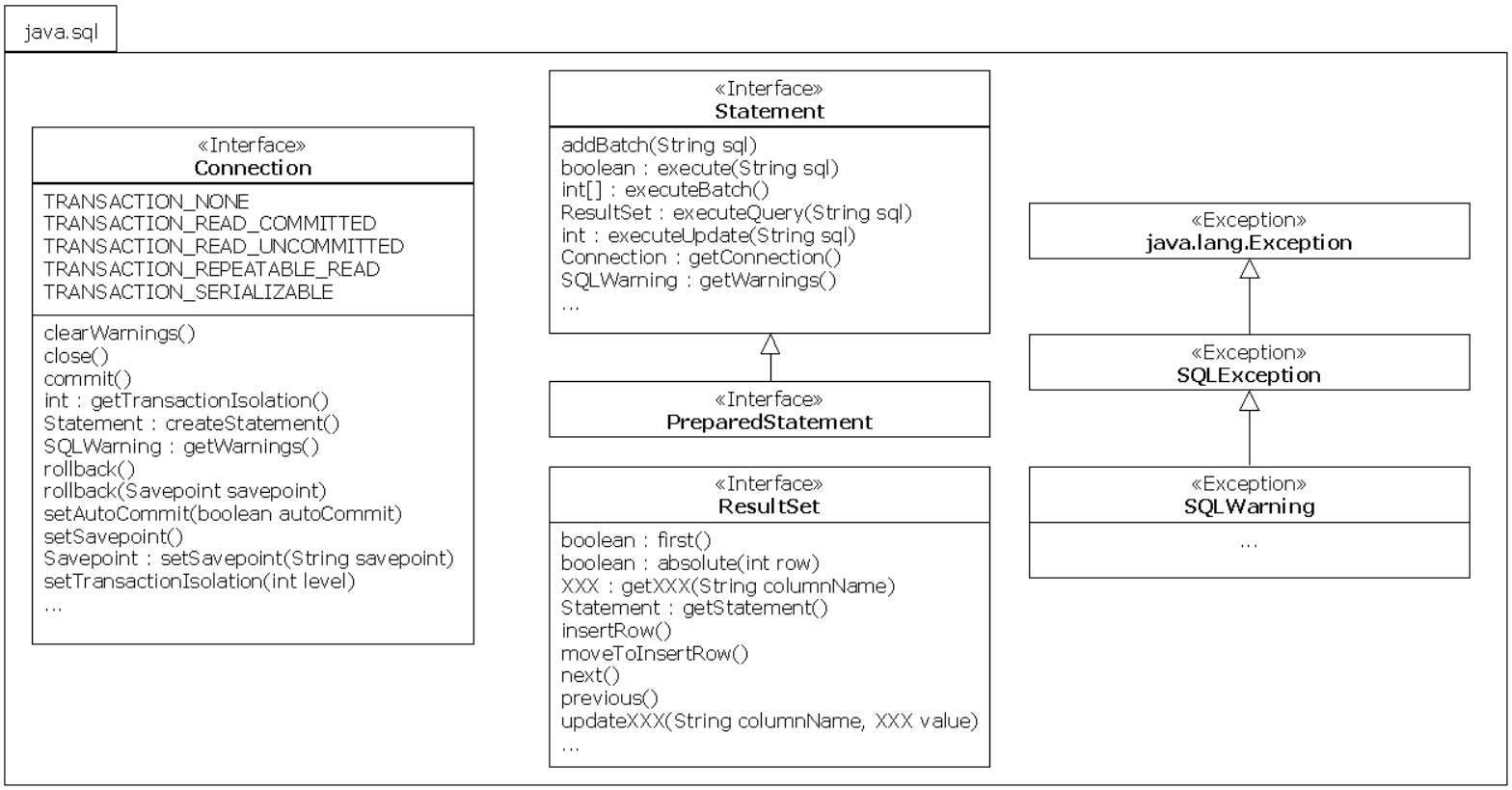
`NullPointerException` - if `stackTrace` is `null`, or if any of the elements of `stackTrace` are `null`

Since:

1.4

Submit a bug or feature

For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.


java.sql
«Interface»
Connection
TRANSACTION_NONE
TRANSACTION_READ_COMMITTED
TRANSACTION_READ_UNCOMMITTED
TRANSACTION_REPEATABLE_READ
TRANSACTION_SERIALIZABLE
clearWarnings()
close()
commit()
int : getTransactionIsolation()
Statement : createStatement()
SQLWarning : getWarnings()
rollback()
rollback(Savepoint savepoint)
setAutoCommit(boolean autoCommit)
setSavepoint()
Savepoint : setSavepoint(String savepoint)
setTransactionIsolation(int level)
...
«Interface»
Statement
addBatch(String sql)
boolean : execute(String sql)
int[] : executeBatch()
ResultSet : executeQuery(String sql)
int : executeUpdate(String sql)
Connection : getConnection()
SQLWarning : getWarnings()
...
«Interface»
PreparedStatement
«Interface»
ResultSet
boolean : first()
boolean : absolute(int row)
XXX : getXXX(String columnName)
Statement : getStatement()
insertRow()
moveToInsertRow()
next()
previous()
updateXXX(String columnName, XXX value)
...
«Exception»
java.lang.Exception
«Exception»
SQLException
«Exception»
SQLWarning
...

java.lang

Class Class

java.lang.Object
  └ **java.lang.Class**

All Implemented Interfaces:

Serializable

---

public final class **Class**
extends Object
implements Serializable

Instances of the class `Class` represent classes and interfaces in a running Java application. Every array also belongs to a class that is reflected as a `Class` object that is shared by all arrays with the same element type and number of dimensions. The primitive Java types (`boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, and `double`), and the keyword `void` are also represented as `Class` objects.

`Class` has no public constructor. Instead `Class` objects are constructed automatically by the Java Virtual Machine as classes are loaded and by calls to the `defineClass` method in the class loader.

The following example uses a `Class` object to print the class name of an object:

```
     void printClassName(Object obj) {
         System.out.println("The class of " + obj +
                            " is " + obj.getClass().
getName());
     }
```

It is also possible to get the `Class` object for a named type (or for void) using a class literal (JLS Section 15.8.2). For example:

```
     System.out.println("The name of class Foo is: "+Foo.
class.getName());
```

Since:

JDK1.0

See Also:

`ClassLoader.defineClass(byte[], int, int)`, Serialized Form

Method Summary

boolean	**desiredAssertionStatus**() Returns the assertion status that would be assigned to this class if it were to be initialized at the time this method is invoked.
static Class	**forName**(String className) Returns the `Class` object associated with the class or interface with the given string name.
static Class	**forName**(String name, boolean initialize, ClassLoader loader) Returns the `Class` object associated with the class or interface with the given string name, using the given class loader.
Class[]	**getClasses**() Returns an array containing `Class` objects representing all the public classes and interfaces that are members of the class represented by this `Class` object.
ClassLoader	**getClassLoader**() Returns the class loader for the class.
Class	**getComponentType**() Returns the `Class` representing the component type of an array.
Constructor	**getConstructor**(Class[] parameterTypes) Returns a `Constructor` object that reflects the specified public constructor of the class represented by this `Class` object.
Constructor[]	**getConstructors**() Returns an array containing `Constructor` objects reflecting all the public constructors of the class represented by this `Class` object.
Class[]	**getDeclaredClasses**() Returns an array of `Class` objects reflecting all the classes and interfaces declared as members of the class represented by this `Class` object.

Constructor	**getDeclaredConstructor**(Class[] parameterTypes) Returns a Constructor object that reflects the specified constructor of the class or interface represented by this Class object.
Constructor[]	**getDeclaredConstructors**() Returns an array of Constructor objects reflecting all the constructors declared by the class represented by this Class object.
Field	**getDeclaredField**(String name) Returns a Field object that reflects the specified declared field of the class or interface represented by this Class object.
Field[]	**getDeclaredFields**() Returns an array of Field objects reflecting all the fields declared by the class or interface represented by this Class object.
Method	**getDeclaredMethod**(String name, Class[] parameterTypes) Returns a Method object that reflects the specified declared method of the class or interface represented by this Class object.
Method[]	**getDeclaredMethods**() Returns an array of Method objects reflecting all the methods declared by the class or interface represented by this Class object.
Class	**getDeclaringClass**() If the class or interface represented by this Class object is a member of another class, returns the Class object representing the class in which it was declared.
Field	**getField**(String name) Returns a Field object that reflects the specified public member field of the class or interface represented by this Class object.
Field[]	**getFields**() Returns an array containing Field objects reflecting all the accessible public fields of the class or interface represented by this Class object.
Class[]	**getInterfaces**() Determines the interfaces implemented by the class or interface represented by this object.
Method	**getMethod**(String name, Class[] parameterTypes) Returns a Method object that reflects the specified public member method of the class or interface represented by this Class object.

Method[]	**getMethods**() Returns an array containing Method objects reflecting all the public *member* methods of the class or interface represented by this Class object, including those declared by the class or interface and and those inherited from superclasses and superinterfaces.
int	**getModifiers**() Returns the Java language modifiers for this class or interface, encoded in an integer.
String	**getName**() Returns the name of the entity (class, interface, array class, primitive type, or void) represented by this Class object, as a String.
Package	**getPackage**() Gets the package for this class.
ProtectionDomain	**getProtectionDomain**() Returns the ProtectionDomain of this class.
URL	**getResource**(String name) Finds a resource with a given name.
InputStream	**getResourceAsStream**(String name) Finds a resource with a given name.
Object[]	**getSigners**() Gets the signers of this class.
Class	**getSuperclass**() Returns the Class representing the superclass of the entity (class, interface, primitive type or void) represented by this Class.
boolean	**isArray**() Determines if this Class object represents an array class.
boolean	**isAssignableFrom**(Class cls) Determines if the class or interface represented by this Class object is either the same as, or is a superclass or superinterface of, the class or interface represented by the specified Class parameter.
boolean	**isInstance**(Object obj) Determines if the specified Object is assignment-compatible with the object represented by this Class.
boolean	**isInterface**() Determines if the specified Class object represents an interface type.

boolean	**isPrimitive**() Determines if the specified Class object represents a primitive type.
Object	**newInstance**() Creates a new instance of the class represented by this Class object.
String	**toString**() Converts the object to a string.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Method Detail

toString

```
public String toString()
```

Converts the object to a string. The string representation is the string "class" or "interface", followed by a space, and then by the fully qualified name of the class in the format returned by getName. If this Class object represents a primitive type, this method returns the name of the primitive type. If this Class object represents void this method returns "void".

Overrides:

toString in class Object

Returns:

a string representation of this class object.

---

forName

```
public static Class forName(String className)
                    throws ClassNotFoundException
```

Returns the Class object associated with the class or interface with the given string name. Invoking this method is equivalent to:

```
Class.forName(className, true, currentLoader)
```

where `currentLoader` denotes the defining class loader of the current class.

For example, the following code fragment returns the runtime `Class` descriptor for the class named `java.lang.Thread`:

```
Class t = Class.forName("java.lang.Thread")
```

A call to `forName("X")` causes the class named `X` to be initialized.

Parameters:
`className` - the fully qualified name of the desired class.
Returns:
the `Class` object for the class with the specified name.
Throws:
`LinkageError` - if the linkage fails
`ExceptionInInitializerError` - if the initialization provoked by this method fails
`ClassNotFoundException` - if the class cannot be located

---

forName

```
public static Class forName(String name,
                            boolean initialize,
                            ClassLoader loader)
                     throws ClassNotFoundException
```

Returns the `Class` object associated with the class or interface with the given string name, using the given class loader. Given the fully qualified name for a class or interface (in the same format returned by `getName`) this method attempts to locate, load, and link the class or interface. The specified class loader is used to load the class or interface. If the parameter `loader` is null, the class is loaded through the bootstrap class loader. The class is initialized only if the `initialize` parameter is `true` and if it has not been initialized earlier.

If `name` denotes a primitive type or void, an attempt will be made to locate a user-defined class in the unnamed package whose name is `name`. Therefore, this method cannot be used to obtain any of the `Class` objects representing primitive types or void.

If `name` denotes an array class, the component type of the array class is loaded but not

initialized.

For example, in an instance method the expression:

```
Class.forName("Foo")
```

is equivalent to:

```
Class.forName("Foo", true, this.getClass().
getClassLoader())
```

Note that this method throws errors related to loading, linking or initializing as specified in Sections 12.2, 12.3 and 12.4 of *The Java Language Specification*. Note that this method does not check whether the requested class is accessible to its caller.

If the `loader` is `null`, and a security manager is present, and the caller's class loader is not null, then this method calls the security manager's `checkPermission` method with a `RuntimePermission("getClassLoader")` permission to ensure it's ok to access the bootstrap class loader.

Parameters:

`name` - fully qualified name of the desired class
`initialize` - whether the class must be initialized
`loader` - class loader from which the class must be loaded

Returns:

class object representing the desired class

Throws:

`LinkageError` - if the linkage fails
`ExceptionInInitializerError` - if the initialization provoked by this method fails
`ClassNotFoundException` - if the class cannot be located by the specified class loader

Since:

1.2

See Also:

`forName(String)`, `ClassLoader`

---

newInstance

`public Object` **`newInstance`**`()`

```
throws InstantiationException,
       IllegalAccessException
```

Creates a new instance of the class represented by this `Class` object. The class is instantiated as if by a `new` expression with an empty argument list. The class is initialized if it has not already been initialized.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.PUBLIC` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

Returns:

a newly allocated instance of the class represented by this object.

Throws:

`IllegalAccessException` - if the class or its nullary constructor is not accessible.

`InstantiationException` - if this `Class` represents an abstract class, an interface, an array class, a primitive type, or void; or if the class has no nullary constructor; or if the instantiation fails for some other reason.

`ExceptionInInitializerError` - if the initialization provoked by this method fails.

`SecurityException` - if there is no permission to create a new instance.

---

isInstance

```
public boolean isInstance(Object obj)
```

Determines if the specified `Object` is assignment-compatible with the object represented by this `Class`. This method is the dynamic equivalent of the Java language `instanceof` operator. The method returns `true` if the specified `Object` argument is non-null and can be cast to the reference type represented by this `Class` object without raising a `ClassCastException.` It returns `false` otherwise.

Specifically, if this `Class` object represents a declared class, this method returns `true` if the specified `Object` argument is an instance of the represented class (or of any of its subclasses); it returns `false` otherwise. If this `Class` object represents an array class, this method returns `true` if the specified `Object` argument can be converted to an object of the array class by an identity conversion or by a widening reference conversion; it returns `false` otherwise. If this `Class` object represents an interface, this method returns `true` if the class or any superclass of the specified `Object` argument implements this interface; it returns

`false` otherwise. If this `Class` object represents a primitive type, this method returns `false`.

Parameters:
`obj` - the object to check
Returns:
true if `obj` is an instance of this class
Since:
JDK1.1

---

isAssignableFrom

```
public boolean isAssignableFrom(Class cls)
```

Determines if the class or interface represented by this `Class` object is either the same as, or is a superclass or superinterface of, the class or interface represented by the specified `Class` parameter. It returns `true` if so; otherwise it returns `false`. If this `Class` object represents a primitive type, this method returns `true` if the specified `Class` parameter is exactly this `Class` object; otherwise it returns `false`.

Specifically, this method tests whether the type represented by the specified `Class` parameter can be converted to the type represented by this `Class` object via an identity conversion or via a widening reference conversion. See *The Java Language Specification*, sections 5.1.1 and 5.1.4 , for details.

Parameters:
`cls` - the `Class` object to be checked
Returns:
the `boolean` value indicating whether objects of the type `cls` can be assigned to objects of this class
Throws:
`NullPointerException` - if the specified Class parameter is null.
Since:
JDK1.1

---

isInterface

```
public boolean isInterface()
```

Determines if the specified `Class` object represents an interface type.

Returns:

`true` if this object represents an interface; `false` otherwise.

---

isArray

```
public boolean isArray()
```

Determines if this `Class` object represents an array class.

Returns:

`true` if this object represents an array class; `false` otherwise.

Since:

JDK1.1

---

isPrimitive

```
public boolean isPrimitive()
```

Determines if the specified `Class` object represents a primitive type.

There are nine predefined `Class` objects to represent the eight primitive types and void. These are created by the Java Virtual Machine, and have the same names as the primitive types that they represent, namely `boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, and `double`.

These objects may only be accessed via the following public static final variables, and are the only `Class` objects for which this method returns `true`.

Returns:

true if and only if this class represents a primitive type

Since:

JDK1.1

See Also:

`Boolean.TYPE`, `Character.TYPE`, `Byte.TYPE`, `Short.TYPE`, `Integer.TYPE`, `Long.TYPE`, `Float.TYPE`, `Double.TYPE`, `Void.TYPE`

---

getName

```
public String getName()
```

Returns the name of the entity (class, interface, array class, primitive type, or void) represented by this `Class` object, as a `String`.

If this class object represents a reference type that is not an array type then the binary name of the class is returned, as specified by the Java Language Specification, Second Edition.

If this class object represents a primitive type or void, then the name returned is a `String` equal to the Java language keyword corresponding to the primitive type or void.

If this class object represents a class of arrays, then the internal form of the name consists of the name of the element type preceded by one or more '`[`' characters representing the depth of the array nesting. The encoding of element type names is as follows:

Element Type	Encoding
boolean	Z
byte	B
char	C
class or interface	L*classname;*
double	D
float	F
int	I
long	J
short	S

The class or interface name *classname* is the binary name of the class specified above.

Examples:

```
String.class.getName()
    returns "java.lang.String"
byte.class.getName()
    returns "byte"
(new Object[3]).getClass().getName()
    returns "[Ljava.lang.Object;"
(new int[3][4][5][6][7][8][9]).getClass().getName()
    returns "[[[[[[[I"
```

Returns:

the name of the class or interface represented by this object.

---

getClassLoader

`public ClassLoader getClassLoader()`

Returns the class loader for the class. Some implementations may use null to represent the bootstrap class loader. This method will return null in such implementations if this class was loaded by the bootstrap class loader.

If a security manager is present, and the caller's class loader is not null and the caller's class loader is not the same as or an ancestor of the class loader for the class whose class loader is requested, then this method calls the security manager's `checkPermission` method with a `RuntimePermission("getClassLoader")` permission to ensure it's ok to access the class loader for the class.

If this object represents a primitive type or void, null is returned.

Returns:

the class loader that loaded the class or interface represented by this object.

Throws:

`SecurityException` - if a security manager exists and its `checkPermission` method denies access to the class loader for the class.

See Also:

`ClassLoader`, `SecurityManager.checkPermission(java.security.Permission)`, `RuntimePermission`

---

getSuperclass

`public Class getSuperclass()`

Returns the `Class` representing the superclass of the entity (class, interface, primitive type or void) represented by this `Class`. If this `Class` represents either the `Object` class, an interface, a primitive type, or void, then null is returned. If this object represents an array class then the `Class` object representing the `Object` class is returned.

Returns:

the superclass of the class represented by this object.

---

getPackage

`public Package getPackage()`

Gets the package for this class. The class loader of this class is used to find the package. If the class was loaded by the bootstrap class loader the set of packages loaded from CLASSPATH is searched to find the package of the class. Null is returned if no package object was created by the class loader of this class.

Packages have attributes for versions and specifications only if the information was defined in the manifests that accompany the classes, and if the class loader created the package instance with the attributes from the manifest.

Returns:

the package of the class, or null if no package information is available from the archive or codebase.

---

getInterfaces

`public Class[] getInterfaces()`

Determines the interfaces implemented by the class or interface represented by this object.

If this object represents a class, the return value is an array containing objects representing all interfaces implemented by the class. The order of the interface objects in the array corresponds to the order of the interface names in the `implements` clause of the declaration of the class represented by this object. For example, given the declaration:

```
class Shimmer implements FloorWax, DessertTopping
{ ... }
```

suppose the value of `s` is an instance of `Shimmer`; the value of the expression:

```
s.getClass().getInterfaces()[0]
```

is the `Class`object that represents interface `FloorWax`; and the value of:

```
s.getClass().getInterfaces()[1]
```

is the `Class`object that represents interface `DessertTopping`.

If this object represents an interface, the array contains objects representing all interfaces extended by the interface. The order of the interface objects in the array corresponds to the order of the interface names in the `extends` clause of the declaration of the interface represented by this object.

If this object represents a class or interface that implements no interfaces, the method returns an array of length 0.

If this object represents a primitive type or void, the method returns an array of length 0.

Returns:
an array of interfaces implemented by this class.

---

getComponentType

`public Class getComponentType()`

Returns the `Class` representing the component type of an array. If this class does not represent an array class this method returns null.

Returns:
the `Class` representing the component type of this class if this class is an array
Since:
JDK1.1
See Also:
`Array`

---

getModifiers

`public int getModifiers()`

Returns the Java language modifiers for this class or interface, encoded in an integer. The modifiers consist of the Java Virtual Machine's constants for `public`, `protected`, `private`, `final`, `static`, `abstract` and `interface`; they should be decoded using the methods of class `Modifier`.

If the underlying class is an array class, then its `public`, `private` and `protected` modifiers are the same as those of its component type. If this `Class` represents a primitive

type or void, its `public` modifier is always `true`, and its `protected` and `private` modifiers are always `false`. If this object represents an array class, a primitive type or void, then its `final` modifier is always `true` and its interface modifier is always `false`. The values of its other modifiers are not determined by this specification.

The modifier encodings are defined in *The Java Virtual Machine Specification*, table 4.1.

Returns:
the `int` representing the modifiers for this class
Since:
JDK1.1
See Also:
`Modifier`

---

getSigners

```
public Object[] getSigners()
```

Gets the signers of this class.

Returns:
the signers of this class, or null if there are no signers. In particular, this method returns null if this object represents a primitive type or void.
Since:
JDK1.1

---

getDeclaringClass

```
public Class getDeclaringClass()
```

If the class or interface represented by this `Class` object is a member of another class, returns the `Class` object representing the class in which it was declared. This method returns null if this class or interface is not a member of any other class. If this `Class` object represents an array class, a primitive type, or void,then this method returns null.

Returns:
the declaring class for this class
Since:
JDK1.1

---

getClasses

```
public Class[] getClasses()
```

Returns an array containing `Class` objects representing all the public classes and interfaces that are members of the class represented by this `Class` object. This includes public class and interface members inherited from superclasses and public class and interface members declared by the class. This method returns an array of length 0 if this `Class` object has no public member classes or interfaces. This method also returns an array of length 0 if this `Class` object represents a primitive type, an array class, or void.

For this class and each of its superclasses, the following security checks are performed: If there is a security manager, the security manager's `checkMemberAccess` method is called with `this` and `Member.PUBLIC` as its arguments, where `this` is this class or the superclass whose members are being determined. If the class is in a package, then the security manager's `checkPackageAccess` method is also called with the package name as its argument. Either of these calls could result in a SecurityException.

Returns:
the array of `Class` objects representing the public members of this class
Throws:
`SecurityException` - if access to the information is denied.
Since:
JDK1.1
See Also:
`SecurityManager.checkMemberAccess(Class, int)`, `SecurityManager.checkPackageAccess(String)`

---

getFields

```
public Field[] getFields()
                  throws SecurityException
```

Returns an array containing `Field` objects reflecting all the accessible public fields of the class or interface represented by this `Class` object. The elements in the array returned are not sorted and are not in any particular order. This method returns an array of length 0 if the class or interface has no accessible public fields, or if it represents an array class, a primitive type, or void.

Specifically, if this `Class` object represents a class, this method returns the public fields of

this class and of all its superclasses. If this `Class` object represents an interface, this method returns the fields of this interface and of all its superinterfaces.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.PUBLIC` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

The implicit length field for array class is not reflected by this method. User code should use the methods of class `Array` to manipulate arrays.

See *The Java Language Specification*, sections 8.2 and 8.3.

Returns:

the array of `Field` objects representing the public fields

Throws:

`SecurityException` - if access to the information is denied.

Since:

JDK1.1

See Also:

`Field`, `SecurityManager.checkMemberAccess(Class, int)`, `SecurityManager.checkPackageAccess(String)`

---

getMethods

```
public Method[] getMethods()
                     throws SecurityException
```

Returns an array containing `Method` objects reflecting all the public *member* methods of the class or interface represented by this `Class` object, including those declared by the class or interface and and those inherited from superclasses and superinterfaces. The elements in the array returned are not sorted and are not in any particular order. This method returns an array of length 0 if this `Class` object represents a class or interface that has no public member methods, or if this `Class` object represents an array class, primitive type, or void.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.PUBLIC` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

The class initialization method `<clinit>` is not included in the returned array. If the class declares multiple public member methods with the same parameter types, they are all included in the returned array.

See *The Java Language Specification*, sections 8.2 and 8.4.

Returns:

the array of `Method` objects representing the public methods of this class

Throws:

`SecurityException` - if access to the information is denied.

Since:

JDK1.1

See Also:

`Method`, `SecurityManager.checkMemberAccess(Class, int)`, `SecurityManager.checkPackageAccess(String)`

---

getConstructors

```
public Constructor[] getConstructors()
                              throws SecurityException
```

Returns an array containing `Constructor` objects reflecting all the public constructors of the class represented by this `Class` object. An array of length 0 is returned if the class has no public constructors, or if the class is an array class, or if the class reflects a primitive type or void.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.PUBLIC` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

Returns:

the array containing `Method` objects for all the declared public constructors of this class matches the specified `parameterTypes`

Throws:

`SecurityException` - if access to the information is denied.

Since:

JDK1.1

See Also:

`Constructor`, `SecurityManager.checkMemberAccess(Class, int)`, `SecurityManager.checkPackageAccess(String)`

getField

```
public Field getField(String name)
               throws NoSuchFieldException,
                      SecurityException
```

Returns a `Field` object that reflects the specified public member field of the class or interface represented by this `Class` object. The `name` parameter is a `String` specifying the simple name of the desired field.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.PUBLIC` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

The field to be reflected is determined by the algorithm that follows. Let C be the class represented by this object:

1. If C declares a public field with the name specified, that is the field to be reflected.
2. If no field was found in step 1 above, this algorithm is applied recursively to each direct superinterface of C. The direct superinterfaces are searched in the order they were declared.
3. If no field was found in steps 1 and 2 above, and C has a superclass S, then this algorithm is invoked recursively upon S. If C has no superclass, then a `NoSuchFieldException` is thrown.

See *The Java Language Specification*, sections 8.2 and 8.3.

Parameters:

`name` - the field name

Returns:

the `Field` object of this class specified by `name`

Throws:

`NoSuchFieldException` - if a field with the specified name is not found.

`NullPointerException` - if `name` is `null`

`SecurityException` - if access to the information is denied.

Since:

JDK1.1

See Also:

`Field`, `SecurityManager.checkMemberAccess(Class, int)`,

SecurityManager.checkPackageAccess(String)

---

getMethod

```
public Method getMethod(String name,
                        Class[] parameterTypes)
                 throws NoSuchMethodException,
                        SecurityException
```

Returns a `Method` object that reflects the specified public member method of the class or interface represented by this `Class` object. The `name` parameter is a `String` specifying the simple name the desired method. The `parameterTypes` parameter is an array of `Class` objects that identify the method's formal parameter types, in declared order. If `parameterTypes` is `null`, it is treated as if it were an empty array.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.PUBLIC` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

If the `name` is "<init>"or "<clinit>" a `NoSuchMethodException` is raised. Otherwise, the method to be reflected is determined by the algorithm that follows. Let C be the class represented by this object:

1. C is searched for any *matching methods*. If no matching method is found, the algorithm of step 1 is invoked recursively on the superclass of C.
2. If no method was found in step 1 above, the superinterfaces of C are searched for a matching method. If any such method is found, it is reflected.

To find a matching method in a class C: If C declares exactly one public method with the specified name and exactly the same formal parameter types, that is the method reflected. If more than one such method is found in C, and one of these methods has a return type that is more specific than any of the others, that method is reflected; otherwise one of the methods is chosen arbitrarily.

See *The Java Language Specification*, sections 8.2 and 8.4.

Parameters:

`name` - the name of the method

`parameterTypes` - the list of parameters

Returns:

the `Method` object that matches the specified `name` and `parameterTypes`

Throws:

NoSuchMethodException - if a matching method is not found or if the name is "<init>"or "<clinit>".

NullPointerException - if `name` is `null`

SecurityException - if access to the information is denied.

Since:

JDK1.1

See Also:

Method, SecurityManager.checkMemberAccess(Class, int), SecurityManager.checkPackageAccess(String)

---

getConstructor

```
public Constructor getConstructor(Class[] parameterTypes)
                           throws NoSuchMethodException,
                                  SecurityException
```

Returns a `Constructor` object that reflects the specified public constructor of the class represented by this `Class` object. The `parameterTypes` parameter is an array of `Class` objects that identify the constructor's formal parameter types, in declared order.

The constructor to reflect is the public constructor of the class represented by this `Class` object whose formal parameter types match those specified by `parameterTypes`.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.PUBLIC` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

Parameters:

`parameterTypes` - the parameter array

Returns:

the `Method` object of the public constructor that matches the specified `parameterTypes`

Throws:

NoSuchMethodException - if a matching method is not found.

SecurityException - if access to the information is denied.

Since:

JDK1.1

See Also:

Constructor, SecurityManager.checkMemberAccess(Class, int),

SecurityManager.checkPackageAccess(String)

---

getDeclaredClasses

```
public Class[] getDeclaredClasses()
                            throws SecurityException
```

Returns an array of `Class` objects reflecting all the classes and interfaces declared as members of the class represented by this `Class` object. This includes public, protected, default (package) access, and private classes and interfaces declared by the class, but excludes inherited classes and interfaces. This method returns an array of length 0 if the class declares no classes or interfaces as members, or if this `Class` object represents a primitive type, an array class, or void.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.DECLARED` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

Returns:

the array of `Class` objects representing all the declared members of this class

Throws:

SecurityException - if access to the information is denied.

Since:

JDK1.1

See Also:

SecurityManager.checkMemberAccess(Class, int), SecurityManager.checkPackageAccess(String)

---

getDeclaredFields

```
public Field[] getDeclaredFields()
                            throws SecurityException
```

Returns an array of `Field` objects reflecting all the fields declared by the class or interface represented by this `Class` object. This includes public, protected, default (package) access, and private fields, but excludes inherited fields. The elements in the array returned are not sorted and are not in any particular order. This method returns an array of length 0 if the class or interface declares no fields, or if this `Class` object represents a primitive type, an array

class, or void.

See *The Java Language Specification*, sections 8.2 and 8.3.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.DECLARED` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

Returns:

the array of `Field` objects representing all the declared fields of this class

Throws:

`SecurityException` - if access to the information is denied.

Since:

JDK1.1

See Also:

`Field`, `SecurityManager.checkMemberAccess(Class, int)`, `SecurityManager.checkPackageAccess(String)`

---

getDeclaredMethods

```
public Method[] getDeclaredMethods()
                               throws SecurityException
```

Returns an array of `Method` objects reflecting all the methods declared by the class or interface represented by this `Class` object. This includes public, protected, default (package) access, and private methods, but excludes inherited methods. The elements in the array returned are not sorted and are not in any particular order. This method returns an array of length 0 if the class or interface declares no methods, or if this `Class` object represents a primitive type, an array class, or void. The class initialization method `<clinit>` is not included in the returned array. If the class declares multiple public member methods with the same parameter types, they are all included in the returned array.

See *The Java Language Specification*, section 8.2.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.DECLARED` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

Returns:

the array of `Method` objects representing all the declared methods of this class

Throws:

`SecurityException` - if access to the information is denied.

Since:

JDK1.1

See Also:

`Method`, `SecurityManager.checkMemberAccess(Class, int)`, `SecurityManager.checkPackageAccess(String)`

---

getDeclaredConstructors

```
public Constructor[] getDeclaredConstructors()
                                          throws SecurityException
```

Returns an array of `Constructor` objects reflecting all the constructors declared by the class represented by this `Class` object. These are public, protected, default (package) access, and private constructors. The elements in the array returned are not sorted and are not in any particular order. If the class has a default constructor, it is included in the returned array. This method returns an array of length 0 if this `Class` object represents an interface, a primitive type, an array class, or void.

See *The Java Language Specification*, section 8.2.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.DECLARED` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

Returns:

the array of `Method` objects representing all the declared constructors of this class

Throws:

`SecurityException` - if access to the information is denied.

Since:

JDK1.1

See Also:

`Constructor`, `SecurityManager.checkMemberAccess(Class, int)`, `SecurityManager.checkPackageAccess(String)`

---

getDeclaredField

```
public Field getDeclaredField(String name)
                       throws NoSuchFieldException,
                              SecurityException
```

Returns a `Field` object that reflects the specified declared field of the class or interface represented by this `Class` object. The `name` parameter is a `String` that specifies the simple name of the desired field. Note that this method will not reflect the `length` field of an array class.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.DECLARED` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

Parameters:

`name` - the name of the field

Returns:

the `Field` object for the specified field in this class

Throws:

`NoSuchFieldException` - if a field with the specified name is not found.

`NullPointerException` - if `name` is `null`

`SecurityException` - if access to the information is denied.

Since:

JDK1.1

See Also:

`Field`, `SecurityManager.checkMemberAccess(Class, int)`, `SecurityManager.checkPackageAccess(String)`

---

getDeclaredMethod

```
public Method getDeclaredMethod(String name,
                                Class[] parameterTypes)
                         throws NoSuchMethodException,
                                SecurityException
```

Returns a `Method` object that reflects the specified declared method of the class or interface represented by this `Class` object. The `name` parameter is a `String` that specifies the simple name of the desired method, and the `parameterTypes` parameter is an array of `Class`

objects that identify the method's formal parameter types, in declared order. If more than one method with the same parameter types is declared in a class, and one of these methods has a return type that is more specific than any of the others, that method is returned; otherwise one of the methods is chosen arbitrarily. If the name is "<init>"or "<clinit>" a `NoSuchMethodException` is raised.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.DECLARED` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these calls could result in a SecurityException.

Parameters:

`name` - the name of the method
`parameterTypes` - the parameter array

Returns:

the `Method` object for the method of this class matching the specified name and parameters

Throws:

`NoSuchMethodException` - if a matching method is not found.
`NullPointerException` - if `name` is `null`
`SecurityException` - if access to the information is denied.

Since:

JDK1.1

See Also:

`Method`, `SecurityManager.checkMemberAccess(Class, int)`, `SecurityManager.checkPackageAccess(String)`

---

getDeclaredConstructor

```
public Constructor getDeclaredConstructor(Class[] parameterTypes)
                                    throws NoSuchMethodException,
                                           SecurityException
```

Returns a `Constructor` object that reflects the specified constructor of the class or interface represented by this `Class` object. The `parameterTypes` parameter is an array of `Class` objects that identify the constructor's formal parameter types, in declared order.

If there is a security manager, this method first calls the security manager's `checkMemberAccess` method with `this` and `Member.DECLARED` as its arguments. If the class is in a package, then this method also calls the security manager's `checkPackageAccess` method with the package name as its argument. Either of these

calls could result in a SecurityException.

Parameters:
`parameterTypes` - the parameter array
Returns:
The `Method` object for the constructor with the specified parameter list
Throws:
`NoSuchMethodException` - if a matching method is not found.
`SecurityException` - if access to the information is denied.
Since:
JDK1.1
See Also:
`Constructor`, `SecurityManager.checkMemberAccess(Class, int)`, `SecurityManager.checkPackageAccess(String)`

---

getResourceAsStream

`public InputStream` **`getResourceAsStream`**`(String name)`

Finds a resource with a given name. This method returns null if no resource with this name is found. The rules for searching resources associated with a given class are implemented by the defining class loader of the class.

This method delegates the call to its class loader, after making these changes to the resource name: if the resource name starts with "/", it is unchanged; otherwise, the package name is prepended to the resource name after converting "." to "/". If this object was loaded by the bootstrap loader, the call is delegated to `ClassLoader.getSystemResourceAsStream`.

Parameters:
`name` - name of the desired resource
Returns:
a `java.io.InputStream` object.
Throws:
`NullPointerException` - if `name` is `null`.
Since:
JDK1.1
See Also:
`ClassLoader`

---

getResource

```
public URL getResource(String name)
```

Finds a resource with a given name. This method returns null if no resource with this name is found. The rules for searching resources associated with a given class are implemented by the * defining class loader of the class.

This method delegates the call to its class loader, after making these changes to the resource name: if the resource name starts with "/", it is unchanged; otherwise, the package name is prepended to the resource name after converting "." to "/". If this object was loaded by the bootstrap loader, the call is delegated to `ClassLoader.getSystemResource`.

Parameters:

`name` - name of the desired resource

Returns:

a `java.net.URL` object.

Since:

JDK1.1

See Also:

`ClassLoader`

---

getProtectionDomain

```
public ProtectionDomain getProtectionDomain()
```

Returns the `ProtectionDomain` of this class. If there is a security manager installed, this method first calls the security manager's `checkPermission` method with a `RuntimePermission("getProtectionDomain")` permission to ensure it's ok to get the `ProtectionDomain`.

Returns:

the ProtectionDomain of this class

Throws:

`SecurityException` - if a security manager exists and its `checkPermission` method doesn't allow getting the ProtectionDomain.

Since:

1.2

See Also:

`ProtectionDomain`, `SecurityManager.checkPermission(java.security.Permission)`, `RuntimePermission`

desiredAssertionStatus

`public boolean` **`desiredAssertionStatus`**`()`

Returns the assertion status that would be assigned to this class if it were to be initialized at the time this method is invoked. If this class has had its assertion status set, the most recent setting will be returned; otherwise, if any package default assertion status pertains to this class, the most recent setting for the most specific pertinent package default assertion status is returned; otherwise, if this class is not a system class (i.e., it has a class loader) its class loader's default assertion status is returned; otherwise, the system class default assertion status is returned.

Few programmers will have any need for this method; it is provided for the benefit of the JRE itself. (It allows a class to determine at the time that it is initialized whether assertions should be enabled.) Note that this method is not guaranteed to return the actual assertion status that was (or will be) associated with the specified class when it was (or will be) initialized.

Returns:

the desired assertion status of the specified class.

Since:

1.4

See Also:

`ClassLoader.setClassAssertionStatus(java.lang.String, boolean)`, `ClassLoader.setPackageAssertionStatus(java.lang.String, boolean)`, `ClassLoader.setDefaultAssertionStatus(boolean)`

java.sql

Interface Driver

public interface **Driver**

The interface that every driver class must implement.

The Java SQL framework allows for multiple database drivers.

Each driver should supply a class that implements the Driver interface.

The DriverManager will try to load as many drivers as it can find and then for any given connection request, it will ask each driver in turn to try to connect to the target URL.

It is strongly recommended that each Driver class should be small and standalone so that the Driver class can be loaded and queried without bringing in vast quantities of supporting code.

When a Driver class is loaded, it should create an instance of itself and register it with the DriverManager. This means that a user can load and register a driver by calling

```
Class.forName("foo.bah.Driver")
```

See Also:

DriverManager, Connection

Method Summary

boolean	**acceptsURL**(String url) Retrieves whether the driver thinks that it can open a connection to the given URL.
Connection	**connect**(String url, Properties info) Attempts to make a database connection to the given URL.

int	**getMajorVersion**() Retrieves the driver's major version number.
int	**getMinorVersion**() Gets the driver's minor version number.
DriverPropertyInfo[]	**getPropertyInfo**(String url, Properties info) Gets information about the possible properties for this driver.
boolean	**jdbcCompliant**() Reports whether this driver is a genuine JDBC Compliant™ driver.

Method Detail

connect

```
public Connection connect(String url,
                          Properties info)
                   throws SQLException
```

Attempts to make a database connection to the given URL. The driver should return "null" if it realizes it is the wrong kind of driver to connect to the given URL. This will be common, as when the JDBC driver manager is asked to connect to a given URL it passes the URL to each loaded driver in turn.

The driver should throw an `SQLException` if it is the right driver to connect to the given URL but has trouble connecting to the database.

The `java.util.Properties` argument can be used to pass arbitrary string tag/value pairs as connection arguments. Normally at least "user" and "password" properties should be included in the `Properties` object.

Parameters:

`url` - the URL of the database to which to connect

`info` - a list of arbitrary string tag/value pairs as connection arguments. Normally at least a "user" and "password" property should be included.

Returns:

a `Connection` object that represents a connection to the URL

Throws:

`SQLException` - if a database access error occurs

acceptsURL

```
public boolean acceptsURL(String url)
                   throws SQLException
```

Retrieves whether the driver thinks that it can open a connection to the given URL. Typically drivers will return `true` if they understand the subprotocol specified in the URL and `false` if they do not.

Parameters:
`url` - the URL of the database
Returns:
`true` if this driver understands the given URL; `false` otherwise
Throws:
`SQLException` - if a database access error occurs

---

getPropertyInfo

```
public DriverPropertyInfo[] getPropertyInfo(String url,
                                            Properties info)
                                     throws SQLException
```

Gets information about the possible properties for this driver.

The `getPropertyInfo` method is intended to allow a generic GUI tool to discover what properties it should prompt a human for in order to get enough information to connect to a database. Note that depending on the values the human has supplied so far, additional values may become necessary, so it may be necessary to iterate though several calls to the `getPropertyInfo` method.

Parameters:
`url` - the URL of the database to which to connect
`info` - a proposed list of tag/value pairs that will be sent on connect open
Returns:
an array of `DriverPropertyInfo` objects describing possible properties. This array may be an empty array if no properties are required.
Throws:
`SQLException` - if a database access error occurs

---

getMajorVersion

```
public int getMajorVersion()
```

Retrieves the driver's major version number. Initially this should be 1.

Returns:
this driver's major version number

---

getMinorVersion

```
public int getMinorVersion()
```

Gets the driver's minor version number. Initially this should be 0.

Returns:
this driver's minor version number

---

jdbcCompliant

```
public boolean jdbcCompliant()
```

Reports whether this driver is a genuine JDBC Compliant™ driver. A driver may only report `true` here if it passes the JDBC compliance tests; otherwise it is required to return `false`.

JDBC compliance requires full support for the JDBC API and full support for SQL 92 Entry Level. It is expected that JDBC compliant drivers will be available for all the major commercial databases.

This method is not intended to encourage the development of non-JDBC compliant drivers, but is a recognition of the fact that some vendors are interested in using the JDBC API and framework for lightweight databases that do not support full database functionality, or for special databases such as document information retrieval where a SQL implementation may not be feasible.

Returns:
`true` if this driver is JDBC Compliant; `false` otherwise

Submit a bug or feature

For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

---

java.sql

Class DriverManager

java.lang.Object
  └ **java.sql.DriverManager**

---

public class **DriverManager**
extends Object

The basic service for managing a set of JDBC drivers.
NOTE: The `DataSource` interface, new in the JDBC 2.0 API, provides another way to connect to a data source. The use of a `DataSource` object is the preferred means of connecting to a data source.

As part of its initialization, the `DriverManager` class will attempt to load the driver classes referenced in the "jdbc.drivers" system property. This allows a user to customize the JDBC Drivers used by their applications. For example in your ~/.hotjava/properties file you might specify:

```
jdbc.drivers=foo.bah.Driver:wombat.sql.Driver:bad.taste.ourDriver
```

A program can also explicitly load JDBC drivers at any time. For example, the my.sql.Driver is loaded with the following statement:

```
Class.forName("my.sql.Driver");
```

When the method `getConnection` is called, the `DriverManager` will attempt to locate a suitable driver from amongst those loaded at initialization and those loaded explicitly using the same classloader as the current applet or application.

Starting with the Java 2 SDK, Standard Edition, version 1.3, a logging stream can be set only if the proper permission has been granted. Normally this will be done with the tool PolicyTool, which can be used to grant `permission java.sql.SQLPermission "setLog"`.

See Also:

Driver, Connection

Method Summary

static void	**deregisterDriver**(Driver driver) Drops a driver from the DriverManager's list.
static Connection	**getConnection**(String url) Attempts to establish a connection to the given database URL.
static Connection	**getConnection**(String url, Properties info) Attempts to establish a connection to the given database URL.
static Connection	**getConnection**(String url, String user, String password) Attempts to establish a connection to the given database URL.
static Driver	**getDriver**(String url) Attempts to locate a driver that understands the given URL.
static Enumeration	**getDrivers**() Retrieves an Enumeration with all of the currently loaded JDBC drivers to which the current caller has access.
static int	**getLoginTimeout**() Gets the maximum time in seconds that a driver can wait when attempting to log in to a database.
static PrintStream	**getLogStream**() **Deprecated.**
static PrintWriter	**getLogWriter**() Retrieves the log writer.
static void	**println**(String message) Prints a message to the current JDBC log stream.
static void	**registerDriver**(Driver driver) Registers the given driver with the DriverManager.
static void	**setLoginTimeout**(int seconds) Sets the maximum time in seconds that a driver will wait while attempting to connect to a database.
static void	**setLogStream**(PrintStream out) **Deprecated.**

static void	**setLogWriter**(PrintWriter out) Sets the logging/tracing PrintWriter object that is used by the DriverManager and all drivers.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

getLogWriter

```
public static PrintWriter getLogWriter()
```

Retrieves the log writer. The getLogWriter and setLogWriter methods should be used instead of the get/setlogStream methods, which are deprecated.

Returns:
a java.io.PrintWriter object
Since:
1.2
See Also:
setLogWriter(java.io.PrintWriter)

---

setLogWriter

```
public static void setLogWriter(PrintWriter out)
```

Sets the logging/tracing PrintWriter object that is used by the DriverManager and all drivers.

There is a minor versioning problem created by the introduction of the method setLogWriter. The method setLogWriter cannot create a PrintStream object that will be returned by getLogStream---the Java platform does not provide a backward conversion. As a result, a new application that uses setLogWriter and also uses a JDBC 1.0 driver that uses getLogStream will likely not see debugging information written by that driver.

In the Java 2 SDK, Standard Edition, version 1.3 release, this method checks to see that there is an `SQLPermission` object before setting the logging stream. If a `SecurityManager` exists and its `checkPermission` method denies setting the log writer, this method throws a `java.lang.SecurityException`.

Parameters:

`out` - the new logging/tracing `PrintStream` object; `null` to disable logging and tracing

Throws:

`SecurityException` - if a security manager exists and its `checkPermission` method denies setting the log writer

Since:

1.2

See Also:

`SecurityManager.checkPermission(java.security.Permission)`, `getLogWriter()`

---

getConnection

```
public static Connection getConnection(String url,
                                       Properties info)
                                throws SQLException
```

Attempts to establish a connection to the given database URL. The `DriverManager` attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:

`url` - a database url of the form `jdbc:`*`subprotocol`*`:`*`subname`*

`info` - a list of arbitrary string tag/value pairs as connection arguments; normally at least a "user" and "password" property should be included

Returns:

a Connection to the URL

Throws:

`SQLException` - if a database access error occurs

---

getConnection

```
public static Connection getConnection(String url,
                                       String user,
```

```
                                String password)
                         throws SQLException
```

Attempts to establish a connection to the given database URL. The `DriverManager` attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:

`url` - a database url of the form `jdbc:`*`subprotocol`*`:`*`subname`*
`user` - the database user on whose behalf the connection is being made
`password` - the user's password

Returns:

a connection to the URL

Throws:

`SQLException` - if a database access error occurs

---

getConnection

```
public static Connection getConnection(String url)
                                throws SQLException
```

Attempts to establish a connection to the given database URL. The `DriverManager` attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:

`url` - a database url of the form `jdbc:`*`subprotocol`*`:`*`subname`*

Returns:

a connection to the URL

Throws:

`SQLException` - if a database access error occurs

---

getDriver

```
public static Driver getDriver(String url)
                        throws SQLException
```

Attempts to locate a driver that understands the given URL. The `DriverManager` attempts to select an appropriate driver from the set of registered JDBC drivers.

Parameters:

`url` - a database URL of the form `jdbc:`*`subprotocol`*`:`*`subname`*

Returns:

a `Driver` object representing a driver that can connect to the given URL

Throws:

`SQLException` - if a database access error occurs

---

registerDriver

```
public static void registerDriver(Driver driver)
                           throws SQLException
```

Registers the given driver with the `DriverManager`. A newly-loaded driver class should call the method `registerDriver` to make itself known to the `DriverManager`.

Parameters:

`driver` - the new JDBC Driver that is to be registered with the `DriverManager`

Throws:

`SQLException` - if a database access error occurs

---

deregisterDriver

```
public static void deregisterDriver(Driver driver)
                             throws SQLException
```

Drops a driver from the `DriverManager`'s list. Applets can only deregister drivers from their own classloaders.

Parameters:

`driver` - the JDBC Driver to drop

Throws:

`SQLException` - if a database access error occurs

---

getDrivers

```
public static Enumeration getDrivers()
```

Retrieves an Enumeration with all of the currently loaded JDBC drivers to which the current

caller has access.

Note: The classname of a driver can be found using `d.getClass().getName()`

Returns:
the list of JDBC Drivers loaded by the caller's class loader

---

setLoginTimeout

```
public static void setLoginTimeout(int seconds)
```

Sets the maximum time in seconds that a driver will wait while attempting to connect to a database.

Parameters:
`seconds` - the login time limit in seconds
See Also:
`getLoginTimeout()`

---

getLoginTimeout

```
public static int getLoginTimeout()
```

Gets the maximum time in seconds that a driver can wait when attempting to log in to a database.

Returns:
the driver login time limit in seconds
See Also:
`setLoginTimeout(int)`

---

setLogStream

```
public static void setLogStream(PrintStream out)
```

Deprecated.

Sets the logging/tracing PrintStream that is used by the `DriverManager` and all drivers.

In the Java 2 SDK, Standard Edition, version 1.3 release, this method checks to see that there is an `SQLPermission` object before setting the logging stream. If a `SecurityManager` exists and its `checkPermission` method denies setting the log writer, this method throws a `java.lang.SecurityException`.

Parameters:

`out` - the new logging/tracing PrintStream; to disable, set to `null`

Throws:

`SecurityException` - if a security manager exists and its `checkPermission` method denies setting the log stream

See Also:

`SecurityManager.checkPermission(java.security.Permission)`, `getLogStream()`

---

getLogStream

```
public static PrintStream getLogStream()
```

Deprecated.

Retrieves the logging/tracing PrintStream that is used by the `DriverManager` and all drivers.

Returns:

the logging/tracing PrintStream; if disabled, is `null`

See Also:

`setLogStream(java.io.PrintStream)`

---

println

```
public static void println(String message)
```

Prints a message to the current JDBC log stream.

Parameters:

`message` - a log or tracing message

Submit a bug or feature

For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

```
import java.sql.DriverManager;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;

public class JDBCConnect {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;
		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}
	}
}
```

```
import java.sql.Driver;
import java.sql.DriverManager;
import java.util.Enumeration;

public class JDBCDriver {

	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}

		System.out.println(
			"DriverManager:\nlogin timeout=" + DriverManager.getLoginTimeout());

		Enumeration e = DriverManager.getDrivers();
		while (e.hasMoreElements()) {
			Driver drv = (Driver) e.nextElement();

			System.out.println(
				"Driver="
					+ drv.getClass().getName()
					+ "\nmajor version="
					+ drv.getMajorVersion()
					+ "\nminor version="
					+ drv.getMinorVersion()
					+ "\nJDBC compliant="
					+ drv.jdbcCompliant());
		}

	}
}
```

DriverManager:
login timeout=0
Driver=com.mysql.jdbc.Driver
major version=3
minor version=0
JDBC compliant=false

Core Java

Java Naming and Directory Interface (JNDI)

Downloads
- Early Access

Reference
- API Specifications
- Documentation
- FAQs
- Code Samples & Apps
- Technical Articles & Tips
- Industry Support

Community
- User Groups
- Books & Authors
- Forums
- Bug Database

Learning
- Tutorials & Code Camps

The Java Naming and Directory Interface (JNDI) is part of the Java platform, providing applications based on Java technology with a unified interface to multiple naming and directory services. You can build powerful and portable directory-enabled applications using this industry standard. » Read More

JNDI and J2EE Technologies
JNDI works in concert with other technologies in the Java 2 Platform, Enterprise Edition (J2EE) to organize and locate components in a distributed computing environment.

What's New

May 15 2003
JNDI/LDAP Booster Pack 1.0 Download the booster pack, which contains support for a number of popular LDAP controls and extensions, groups, and Java RMI/CORBA objects. It replaces the booster pack that was bundled with the LDAP 1.2.4 service provider.

November 7, 2002
JNDI Tutorial See the latest version of the JNDI Tutorial.

September 16, 2002
Java 2 SDK, Standard Edition, v 1.4.1 This release of the Java platform includes several JNDI-related enhancements, including support for connection pooling for the LDAP service provider, and automatic discovery of LDAP and DNS services.

September 5, 2002
JNDI/DSML v2 service providers early access release available.

Community

Events
2004 JavaOne Conference. San Francisco , CA Be there! Join the thousands of developers worldwide who come to the JavaOne conference each year in San Francisco to immerse themselves in Java technology, the latest innovations, the community, and the learning opportunities. » Read More

Subscribe to Newsletters. Members of Sun Developer Network can sign up to receive these (and other) newsletters. Not yet a member? **Join us!**

Java Technology Fundamentals
New to Java? Learn the basics of the Java programming language and keep up-to-date on additions to the New-to-Java Programming Center.

Core Java Technologies Newsletter
Find out about new enterprise Java technologies, products, tools, and resources for developers.

Core Java Technologies Tech Tips
Get expert tips, sample code solutions, and techniques for developing in the Java 2 Platform, Standard Edition (J2SE).
» Read More

Related Links

Popular Downloads
- J2SE 1.5.0 Beta 1
- J2SE 1.4.2
- Java Web Services Developer Pack 1.3

Technical Topics
- Performance
- Web Services
- Security
- Desktop

Products and Technologies
- J2SE 1.5
- Desktop Java
- Sun Java Desktop System
- Sun Java Studio Standard IDE

Sun Resources
- java.net
- Java Upgrade Program
- New to Java Center
- Professional Certification
- Professional Training
- JavaOne Online
- Java Community Process

Feedback

If you have comments, questions, or feedback on JNDI, write to us at jndi@java.sun.com.

We also maintain a "jndi-interest" mailing list open to any discussion about JNDI. To subscribe, send an email message with exactly the following syntax in the message body to listserv@java.sun.com:

```
subscribe jndi-interest your real name
```

To remove yourself from the list, use `signoff` instead of `subscribe`.

Your questions may have already been answered. You can browse an archive of messages previously sent to the "jndi-interest" list.

javax.sql

Interface DataSource

public interface **DataSource**

A factory for connections to the physical data source that this `DataSource` object represents. An alternative to the `DriverManager` facility, a `DataSource` object is the preferred means of getting a connection. An object that implements the `DataSource` interface will typically be registered with a naming service based on the Java™ Naming and Directory (JNDI) API.

The `DataSource` interface is implemented by a driver vendor. There are three types of implementations:

1. Basic implementation -- produces a standard `Connection` object
2. Connection pooling implementation -- produces a `Connection` object that will automatically participate in connection pooling. This implementation works with a middle-tier connection pooling manager.
3. Distributed transaction implementation -- produces a `Connection` object that may be used for distributed transactions and almost always participates in connection pooling. This implementation works with a middle-tier transaction manager and almost always with a connection pooling manager.

A `DataSource` object has properties that can be modified when necessary. For example, if the data source is moved to a different server, the property for the server can be changed. The benefit is that because the data source's properties can be changed, any code accessing that data source does not need to be changed.

A driver that is accessed via a `DataSource` object does not register itself with the `DriverManager`. Rather, a `DataSource` object is retrieved though a lookup operation and then used to create a `Connection` object. With a basic implementation, the connection obtained through a `DataSource` object is identical to a connection obtained through the `DriverManager` facility.

Since:

1.4

Method Summary

Connection	**getConnection**() Attempts to establish a connection with the data source that this DataSource object represents.
Connection	**getConnection**(String username, String password) Attempts to establish a connection with the data source that this DataSource object represents.
int	**getLoginTimeout**() Gets the maximum time in seconds that this data source can wait while attempting to connect to a database.
PrintWriter	**getLogWriter**() Retrieves the log writer for this DataSource object.
void	**setLoginTimeout**(int seconds) Sets the maximum time in seconds that this data source will wait while attempting to connect to a database.
void	**setLogWriter**(PrintWriter out) Sets the log writer for this DataSource object to the given java.io.PrintWriter object.

Method Detail

getConnection

```
public Connection getConnection()
                           throws SQLException
```

Attempts to establish a connection with the data source that this DataSource object represents.

Returns:
a connection to the data source

Throws:
SQLException - if a database access error occurs

---

getConnection

```
public Connection getConnection(String username,
                                String password)
                         throws SQLException
```

Attempts to establish a connection with the data source that this `DataSource` object represents.

Parameters:

`username` - the database user on whose behalf the connection is being made
`password` - the user's password

Returns:

a connection to the data source

Throws:

`SQLException` - if a database access error occurs

---

getLogWriter

```
public PrintWriter getLogWriter()
                         throws SQLException
```

Retrieves the log writer for this `DataSource` object.

The log writer is a character output stream to which all logging and tracing messages for this data source will be printed. This includes messages printed by the methods of this object, messages printed by methods of other objects manufactured by this object, and so on. Messages printed to a data source specific log writer are not printed to the log writer associated with the `java.sql.Drivermanager` class. When a `DataSource` object is created, the log writer is initially null; in other words, the default is for logging to be disabled.

Returns:

the log writer for this data source or null if logging is disabled

Throws:

`SQLException` - if a database access error occurs

See Also:

`setLogWriter(java.io.PrintWriter)`

---

setLogWriter

```
public void setLogWriter(PrintWriter out)
```

```
                    throws SQLException
```

Sets the log writer for this `DataSource` object to the given `java.io.PrintWriter` object.

The log writer is a character output stream to which all logging and tracing messages for this data source will be printed. This includes messages printed by the methods of this object, messages printed by methods of other objects manufactured by this object, and so on. Messages printed to a data source- specific log writer are not printed to the log writer associated with the `java.sql.Drivermanager` class. When a `DataSource` object is created the log writer is initially null; in other words, the default is for logging to be disabled.

Parameters:

`out` - the new log writer; to disable logging, set to null

Throws:

`SQLException` - if a database access error occurs

See Also:

`getLogWriter()`

---

setLoginTimeout

```
public void setLoginTimeout(int seconds)
                    throws SQLException
```

Sets the maximum time in seconds that this data source will wait while attempting to connect to a database. A value of zero specifies that the timeout is the default system timeout if there is one; otherwise, it specifies that there is no timeout. When a `DataSource` object is created, the login timeout is initially zero.

Parameters:

`seconds` - the data source login time limit

Throws:

`SQLException` - if a database access error occurs.

See Also:

`getLoginTimeout()`

---

getLoginTimeout

```
public int getLoginTimeout()
                    throws SQLException
```

Gets the maximum time in seconds that this data source can wait while attempting to connect to a database. A value of zero means that the timeout is the default system timeout if there is one; otherwise, it means that there is no timeout. When a `DataSource` object is created, the login timeout is initially zero.

Returns:

the data source login time limit

Throws:

`SQLException` - if a database access error occurs.

See Also:

`setLoginTimeout(int)`

```
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;

public class JDBCConnect2Server {

	public static void main(String[] args) {
		Hashtable env = new Hashtable();
		env.put(
			Context.INITIAL_CONTEXT_FACTORY,
			"com.sun.jndi.fscontext.RefFSContextFactory");
		env.put(Context.PROVIDER_URL, "file:/tmp/registry");

		MysqlDataSource ds = new MysqlDataSource();
		ds.setDatabaseName("jdbctest");
		Context ctx = null;
		try {
			ctx = new InitialContext(env);
		} catch (NamingException ne) {
			ne.printStackTrace();
		}

		try {
			ctx.rebind("jdbc/mySrc", ds);
		} catch (NamingException ne) {
			ne.printStackTrace();
		}
	}
}
```

```
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public class JDBCConnect2 {
	public static void main(String[] args) {
		Hashtable env = new Hashtable();
		env.put(
			Context.INITIAL_CONTEXT_FACTORY,
			"com.sun.jndi.fscontext.RefFSContextFactory");
		env.put(Context.PROVIDER_URL, "file:/tmp/registry");
		Context ctx = null;
		try {
			ctx = new InitialContext(env);
		} catch (NamingException ne) {
			ne.printStackTrace();
		}
		DataSource ds = null;
		try {
			ds = (DataSource) ctx.lookup("jdbc/mySrc");
		} catch (NamingException ne) {
			ne.printStackTrace();
		}
		Connection con = null;
		try {
			con = ds.getConnection("mario", "thePassword");
		} catch (SQLException sqle) {
			Throwable t = sqle;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}
	}
}
```

```
import java.sql.DriverManager;
import java.sql.SQLException;
import javax.sql.PooledConnection;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.jdbc2.optional.MysqlPooledConnection;

public class JDBCConnection3 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException cnfe) {
			System.err.println("Driver class not found");
			cnfe.printStackTrace();
		}
		Connection con = null;
		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		PooledConnection pc = new MysqlPooledConnection(con);

		java.sql.Connection con1 = null;
		try {
			con1 = pc.getConnection();
		} catch (SQLException sqle) {
			Throwable t = sqle;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
```

```
            }
        }
}
```

javax.sql

Interface PooledConnection

All Known Subinterfaces:

XAConnection

public interface **PooledConnection**

An object that provides hooks for connection pool management. A `PooledConnection` object represents a physical connection to a data source. The connection can be recycled rather than being closed when an application is finished with it, thus reducing the number of connections that need to be made.

An application programmer does not use the `PooledConnection` interface directly; rather, it is used by a middle tier infrastructure that manages the pooling of connections.

When an application calls the method `DataSource.getConnection`, it gets back a `Connection` object. If connection pooling is being done, that `Connection` object is actually a handle to a `PooledConnection` object, which is a physical connection.

The connection pool manager, typically the application server, maintains a pool of `PooledConnection` objects. If there is a `PooledConnection` object available in the pool, the connection pool manager returns a `Connection` object that is a handle to that physical connection. If no `PooledConnection` object is available, the connection pool manager calls the `PooledConnection` method `getConnection` to create a new physical connection and returns a handle to it.

When an application closes a connection, it calls the `Connection` method `close`. When connection pooling is being done, the connection pool manager is notified because it has registered itself as a `ConnectionEventListener` object using the `ConnectionPool` method `addConnectionEventListener`. The connection pool manager deactivates the handle to the `PooledConnection` object and returns the `PooledConnection` object to the pool of connections so that it can be used again. Thus, when an application closes its connection, the underlying physical connection is recycled rather than being closed.

The physical connection is not closed until the connection pool manager calls the

`PooledConnection` method `close`. This method is generally called to have an orderly shutdown of the server or if a fatal error has made the connection unusable.

Since:

1.4

Method Summary

void	**addConnectionEventListener**(ConnectionEventListener listener) Registers the given event listener so that it will be notified when an event occurs on this `PooledConnection` object.
void	**close**() Closes the physical connection that this `PooledConnection` object represents.
Connection	**getConnection**() Creates and returns a `Connection` object that is a handle for the physical connection that this `PooledConnection` object represents.
void	**removeConnectionEventListener**(ConnectionEventListener listener) Removes the given event listener from the list of components that will be notified when an event occurs on this `PooledConnection` object.

Method Detail

getConnection

```
public Connection getConnection()
                           throws SQLException
```

Creates and returns a `Connection` object that is a handle for the physical connection that this `PooledConnection` object represents. The connection pool manager calls this method when an application has called the method `DataSource.getConnection` and there are no `PooledConnection` objects available. See the `interface description` for more information.

Returns:

a `Connection` object that is a handle to this `PooledConnection` object

Throws:

SQLException - if a database access error occurs

---

close

```
public void close()
            throws SQLException
```

Closes the physical connection that this PooledConnection object represents. An application never calls this method directly; it is called by the connection pool module, or manager.

See the interface description for more information.

Throws:

SQLException - if a database access error occurs

---

addConnectionEventListener

```
public void addConnectionEventListener
(ConnectionEventListener listener)
```

Registers the given event listener so that it will be notified when an event occurs on this PooledConnection object.

Parameters:

listener - a component, usually the connection pool manager, that has implemented the ConnectionEventListener interface and wants to be notified when the connection is closed or has an error

See Also:

removeConnectionEventListener(javax.sql.ConnectionEventListener)

---

removeConnectionEventListener

```
public void removeConnectionEventListener
(ConnectionEventListener listener)
```

Removes the given event listener from the list of components that will be notified when an event occurs on this `PooledConnection` object.

Parameters:

`listener` - a component, usually the connection pool manager, that has implemented the `ConnectionEventListener` interface and been registered with this `PooledConnection` object as a listener

See Also:

`addConnectionEventListener(javax.sql.ConnectionEventListener)`

```
import java.sql.DriverManager;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCCreateTable {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement  stmt = null;
		try {
			stmt = (Statement) con.createStatement();
		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}
```

```
        String createTab = new String("CREATE TABLE EMPLOYEE(" +
            "FNAME VARCHAR(10) NOT NULL," +
            "MINIT VARCHAR(1)," +
            "LNAME VARCHAR(10) NOT NULL," +
            "SSN INTEGER(9) NOT NULL," +
            "BDATE DATE," +
            "ADDRESS VARCHAR(30)," +
            "SEX ENUM('M','F')," +
            "SALARY REAL(7,2) UNSIGNED," +
            "SUPERSSN INTEGER(9)," +
            "DNO INTEGER(1));");
        try {
            System.out.println("result="+stmt.executeUpdate(createTab));
        } catch (SQLException e3) {
            System.err.println("Error creating table EMPLOYEE");
            Throwable t = e3;
            while (t != null) {
            System.err.println("Type: " + t.getClass().getName());
        System.err.println("Message: " + t.getMessage());
        System.err.println("-----");
        t = t.getCause();
            }
        }
    }
}
```

result=0

```
import java.sql.DriverManager;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCAlterTable {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
				System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt = (Statement) con.createStatement();
		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}
```

```
            String createTab =
                  new String("ALTER TABLE EMPLOYEE ADD PRIMARY KEY (SSN);");
            try {
                  System.out.println("result=" + stmt.executeUpdate(createTab));
            } catch (SQLException e3) {
                  System.err.println("Error altering table EMPLOYEE");
                  Throwable t = e3;
                  while (t != null) {
                  System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                  }
            }
      }
}
```

```
import java.sql.DriverManager;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCInsert1 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt = (Statement) con.createStatement();
		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}
```

```
		try {
			System.out.println("result=" + stmt.executeUpdate("INSERT INTO EMPLOYEE
VALUES('John', 'B', 'Smith', 123456789, '1965-01-09', '731 Fondren, Houston, TX', 'M', 30000,
333445555, 5);"));
			System.out.println("result=" + stmt.executeUpdate("INSERT INTO EMPLOYEE
VALUES('Franklin', 'T', 'Wong', 333445555, '1955-12-08', '638 Voss, Houston, TX', 'M', 40000,
888665555, 5);"));
			System.out.println("result=" + stmt.executeUpdate("INSERT INTO EMPLOYEE
VALUES('Alicia', 'J', 'Zelaya', 999887777, '1968-07-19', '3321 Castle, Spring, TX', 'F', 25000,
987654321, 4);"));
			System.out.println("result=" + stmt.executeUpdate("INSERT INTO EMPLOYEE
VALUES('Jennifer', 'S', 'Wallace', 987654321, '1941-06-20', '291 Berry, Bellaire, TX', 'F', 43000,
888665555, 4);"));
			System.out.println("result=" + stmt.executeUpdate("INSERT INTO EMPLOYEE
VALUES('Ramesh', 'K', 'Narayan', 666884444, '1962-09-15', '975 Fire Oak, Humble, TX', 'M',
38000, 333445555, 5);"));
			System.out.println("result=" + stmt.executeUpdate("INSERT INTO EMPLOYEE
VALUES('Joyce', 'A', 'English', 453453453, '1972-07-31', '5631 Rice, Houston, TX', 'F', 25000,
333445555, 5);"));
			System.out.println("result=" + stmt.executeUpdate("INSERT INTO EMPLOYEE
VALUES('Ahmad', 'V', 'Jabbar', 987987987, '1969-03-29', '980 Dallas, Houston, TX', 'M', 25000,
987654321, 4);"));
			System.out.println("result=" + stmt.executeUpdate("INSERT INTO EMPLOYEE
VALUES('James', 'E', 'Borg', 888665555, '1937-11-10', '450 Stone, Houston, TX', 'M', 55000, null,
1);"));
		} catch (SQLException e3) {
			System.err.println("Error inserting values into table EMPLOYEE");
			Throwable t = e3;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}
	}
}
```

result=1
result=1
result=1
result=1
result=1
result=1
result=1
result=1

```
import java.sql.DriverManager;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCInsert2 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt = (Statement) con.createStatement();
		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}
```

```
            try {
                stmt.addBatch("INSERT INTO EMPLOYEE VALUES('John', 'B', 'Smith',
123456789, '1965-01-09', '731 Fondren, Houston, TX', 'M', 30000, 333445555, 5);");
                stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Franklin', 'T', 'Wong',
333445555, '1955-12-08', '638 Voss, Houston, TX', 'M', 40000, 888665555, 5);");
                stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Alicia', 'J', 'Zelaya',
999887777, '1968-07-19', '3321 Castle, Spring, TX', 'F', 25000, 987654321, 4);");
                stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Jennifer', 'S', 'Wallace',
987654321, '1941-06-20', '291 Berry, Bellaire, TX', 'F', 43000, 888665555, 4);");
                stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Ramesh', 'K', 'Narayan',
666884444, '1962-09-15', '975 Fire Oak, Humble, TX', 'M', 38000, 333445555, 5);");
                stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Joyce', 'A', 'English',
453453453, '1972-07-31', '5631 Rice, Houston, TX', 'F', 25000, 333445555, 5);");
                stmt.addBatch("INSERT INTO EMPLOYEE VALUES('Ahmad', 'V', 'Jabbar',
987987987, '1969-03-29', '980 Dallas, Houston, TX', 'M', 25000, 987654321, 4);");
                stmt.addBatch("INSERT INTO EMPLOYEE VALUES('James', 'E', 'Borg',
888665555, '1937-11-10', '450 Stone, Houston, TX', 'M', 55000, null, 1);");
                int[] insertCounts = stmt.executeBatch();
            } catch (SQLException e3) {
                System.err.println("Error inserting values into table EMPLOYEE");
                Throwable t = e3;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }
        }
}
```

```
import java.sql.DriverManager;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCUpdate1 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt = (Statement) con.createStatement();
		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}
```

```
            try {
                stmt.addBatch("ALTER TABLE EMPLOYEE ADD BDAY VARCHAR(10);");
                stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Sunday' WHERE
DAYOFWEEK(BDATE)=1;");
                stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Monday' WHERE
DAYOFWEEK(BDATE)=2;");
                stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Tuesday' WHERE
DAYOFWEEK(BDATE)=3;");
                stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Wednesday' WHERE
DAYOFWEEK(BDATE)=4;");
                stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Thursday' WHERE
DAYOFWEEK(BDATE)=5;");
                stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Friday' WHERE DAYOFWEEK
(BDATE)=6;");
                stmt.addBatch("UPDATE EMPLOYEE SET BDAY='Saturday' WHERE
DAYOFWEEK(BDATE)=7;");
                int[] result = stmt.executeBatch();
                for (int i=0; i<result.length;i++){
                    System.out.println("Statement No "+i+" changed "+result[i]+" rows");
                }
            } catch (SQLException e3) {
                System.err.println("Error inserting values into table EMPLOYEE");
                Throwable t = e3;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }
        }
}
```

Statement No 0 changed 8 rows
Statement No 1 changed 0 rows
Statement No 2 changed 1 rows
Statement No 3 changed 0 rows
Statement No 4 changed 1 rows
Statement No 5 changed 1 rows
Statement No 6 changed 2 rows
Statement No 7 changed 3 rows

```
import java.sql.DriverManager;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.PreparedStatement;
import com.mysql.jdbc.Statement;

public class JDBCUpdate2 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		PreparedStatement pstmt = null;
		try {
			stmt = (Statement) con.createStatement();
			pstmt = (PreparedStatement) con.prepareStatement("UPDATE EMPLOYEE SET
BDAY=? WHERE DAYOFWEEK(BDATE)=?;");

		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
```

```
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }

            try {
                String[] days=
{"Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"} ;
                stmt.addBatch("ALTER TABLE EMPLOYEE ADD BDAY VARCHAR(10);");
                for (int i=1; i<8;i++){
                    pstmt.setString(1,days[i-1]);
                    pstmt.setInt(2,i);
                    pstmt.addBatch();
                }
                int[] result = stmt.executeBatch();
                for (int i=0; i<result.length;i++){
                    System.out.println("Statement No "+i+" changed "+result[i]+" rows");
                }
            } catch (SQLException e3) {
                System.err.println("Error inserting values into table EMPLOYEE");
                Throwable t = e3;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }
        }
}
```

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;

import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCSelect1 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt = (Statement) con.createStatement();
		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
```

```
            t = t.getCause();
                }
            }

            try {
                ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
                ResultSetMetaData rsmd = rs.getMetaData();
                int noColumns = rsmd.getColumnCount();
                for (int i = 1; i < noColumns; i++) {
                    System.out.print(rsmd.getColumnLabel(i) + "\t");
                }
                System.out.println();

                while (rs.isLast() == false) {
                    rs.next();
                    for (int i = 1; i < noColumns; i++) {
                        System.out.print( rs.getObject(i)+"\t" );
                    }
                    System.out.println();
                }

            } catch (SQLException e3) {
                System.err.println("Error selecting values from table EMPLOYEE");
                Throwable t = e3;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }
        }
}
```

FNAME MINIT LNAME SSN BDATE ADDRESS SEX SALARY SUPERSSN DNO
John B Smith 123456789 1965-01-09 731 Fondren, Houston, TX M 30000.0 333445555 5
Franklin T Wong 333445555 1955-12-08 638 Voss, Houston, TX M 40000.0 888665555 5
Alicia J Zelaya 999887777 1968-07-19 3321 Castle, Spring, TX F 25000.0 987654321 4
Jennifer S Wallace 987654321 1941-06-20 291 Berry, Bellaire, TX F 43000.0 888665555 4
Ramesh K Narayan 666884444 1962-09-15 975 Fire Oak, Humble, TX M 38000.0 333445555 5
Joyce A English 453453453 1972-07-31 5631 Rice, Houston, TX F 25000.0 333445555 5
Ahmad V Jabbar 987987987 1969-03-29 980 Dallas, Houston, TX M 25000.0 987654321 4
James E Borg 888665555 1937-11-10 450 Stone, Houston, TX M 55000.0 null 1

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCSelect5 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt =
				(Statement) con.createStatement();
		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
```

```
            }
        }

        try {
            ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
            rs.afterLast();
            while (rs.previous()){
                System.out.println(rs.getString("FNAME"));
            }
        } catch (SQLException e3) {
            System.err.println("Error selecting values from table EMPLOYEE");
            Throwable t = e3;
            while (t != null) {
            System.err.println("Type: " + t.getClass().getName());
        System.err.println("Message: " + t.getMessage());
        System.err.println("-----");
        t = t.getCause();
            }
        }
    }
}
```

James
Ahmad
Joyce
Ramesh
Jennifer
Alicia
Franklin
John

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCSelect6 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt = (Statement) con.createStatement();
		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
```

```
        }

        try {
            int position = 0;
            ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
            rs.last();
            int size = rs.getRow();
            for (int i = 0; i < size; i++) {
                position = (position + 3) % size;
                rs.absolute(position + 1);
                System.out.println(
                    "position=" + (position + 1) + ": " + rs.getString("FNAME"));
            }
        } catch (SQLException e3) {
            System.err.println("Error selecting values from table EMPLOYEE");
            Throwable t = e3;
            while (t != null) {
            System.err.println("Type: " + t.getClass().getName());
        System.err.println("Message: " + t.getMessage());
        System.err.println("-----");
        t = t.getCause();
            }
        }
    }
}
```

position=4: Jennifer
position=7: Ahmad
position=2: Franklin
position=5: Ramesh
position=8: James
position=3: Alicia
position=6: Joyce
position=1: John

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;

import com.mysql.jdbc.Connection;

public class JDBCSelect2 {
	private static void printResultSet(ResultSet rs) throws SQLException {
		ResultSetMetaData rsmd = rs.getMetaData();
		int noColumns = rsmd.getColumnCount();
		for (int i = 1; i < noColumns; i++) {
			System.out.print(rsmd.getColumnLabel(i) + "\t");
		}
		System.out.println();

		while (rs.isLast() == false) {
			rs.next();
			for (int i = 1; i < noColumns; i++) {
				System.out.print( rs.getObject(i)+"\t" );
			}
			System.out.println();
		}

	}
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
```

```
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }

            Statement stmt = null;
            try {
                stmt = (Statement) con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
            } catch (SQLException e2) {
                System.err.println("Error creating SQL-Statement");
                Throwable t = e2;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }

            try {
                ResultSet uprs = (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
                printResultSet(uprs);
                uprs.moveToInsertRow();
                uprs.updateString("FNAME","Mario");
                uprs.updateString("LNAME","Jeckle");
                uprs.updateInt("SSN",111111111);
                uprs.insertRow();
                uprs = (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
                printResultSet(uprs);
            } catch (SQLException e3) {
                System.err.println("Error selecting values from table EMPLOYEE");
                Throwable t = e3;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }
        }
}
```

FNAME MINIT LNAME SSN BDATE ADDRESS SEX SALARY SUPERSSN DNO
John B Smith 123456789 1965-01-09 731 Fondren, Houston, TX M 30000.0 333445555 5
Franklin T Wong 333445555 1955-12-08 638 Voss, Houston, TX M 40000.0 888665555 5
Joyce A English 453453453 1972-07-31 5631 Rice, Houston, TX F 25000.0 333445555 5
Ramesh K Narayan 666884444 1962-09-15 975 Fire Oak, Humble, TX M 38000.0 333445555 5
James E Borg 888665555 1937-11-10 450 Stone, Houston, TX M 55000.0 null 1
Jennifer S Wallace 987654321 1941-06-20 291 Berry, Bellaire, TX F 43000.0 888665555 4
Ahmad V Jabbar 987987987 1969-03-29 980 Dallas, Houston, TX M 25000.0 987654321 4
Alicia J Zelaya 999887777 1968-07-19 3321 Castle, Spring, TX F 25000.0 987654321 4
FNAME MINIT LNAME SSN BDATE ADDRESS SEX SALARY SUPERSSN DNO
Mario null Jeckle 111111111 null null null null null null
John B Smith 123456789 1965-01-09 731 Fondren, Houston, TX M 30000.0 333445555 5
Franklin T Wong 333445555 1955-12-08 638 Voss, Houston, TX M 40000.0 888665555 5
Joyce A English 453453453 1972-07-31 5631 Rice, Houston, TX F 25000.0 333445555 5
Ramesh K Narayan 666884444 1962-09-15 975 Fire Oak, Humble, TX M 38000.0 333445555 5
James E Borg 888665555 1937-11-10 450 Stone, Houston, TX M 55000.0 null 1
Jennifer S Wallace 987654321 1941-06-20 291 Berry, Bellaire, TX F 43000.0 888665555 4
Ahmad V Jabbar 987987987 1969-03-29 980 Dallas, Houston, TX M 25000.0 987654321 4
Alicia J Zelaya 999887777 1968-07-19 3321 Castle, Spring, TX F 25000.0 987654321 4

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import com.mysql.jdbc.Connection;

public class JDBCSelect3 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt =
				(Statement) con.createStatement(
					ResultSet.TYPE_SCROLL_SENSITIVE,
					ResultSet.CONCUR_UPDATABLE);
		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
```

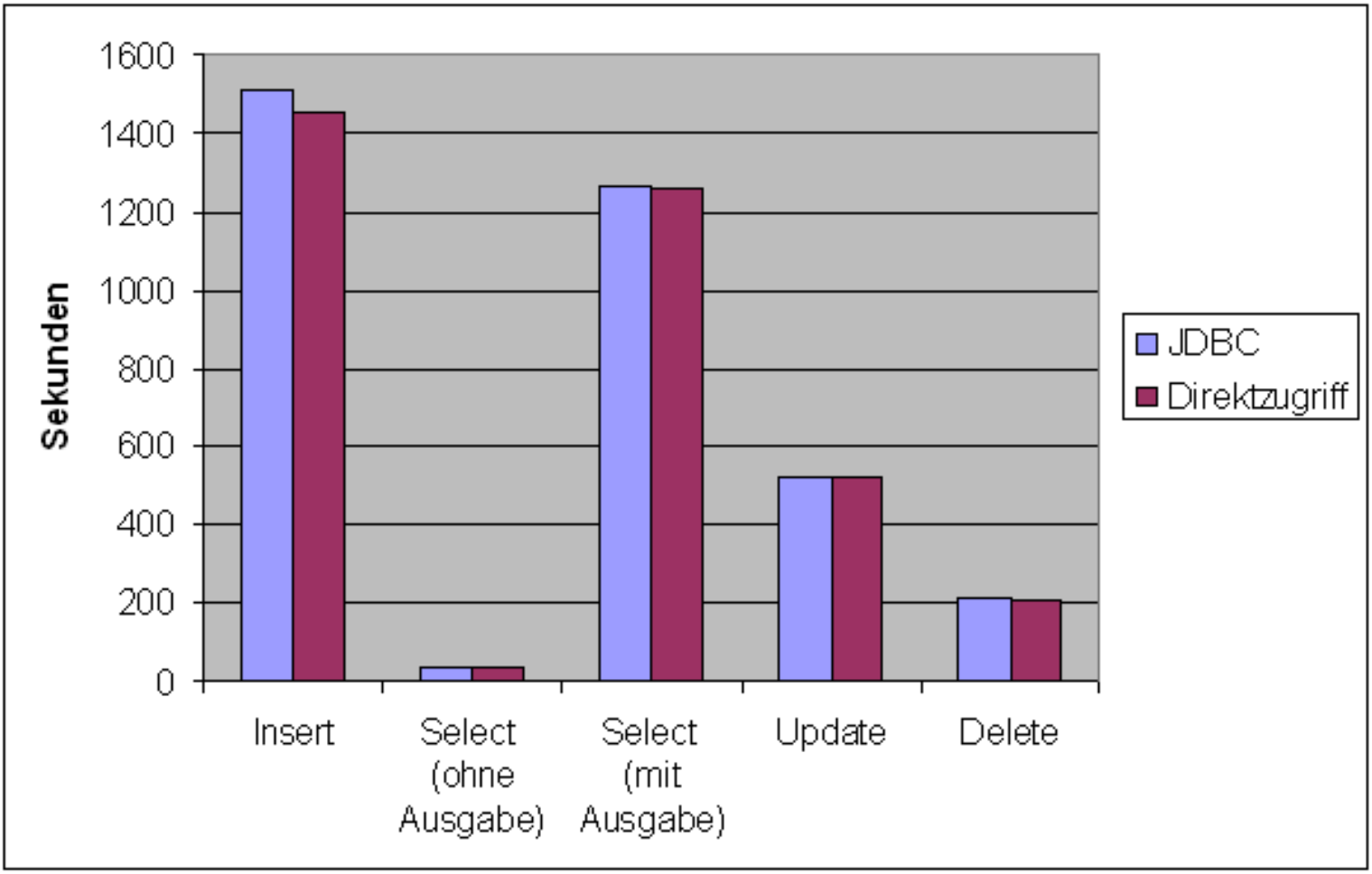
```
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }

            try {
                ResultSet uprs =
                    (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
                int namePos = uprs.findColumn("LNAME");

                while (uprs.isLast() == false) {
                    uprs.next();
                    if (uprs.getString(namePos).compareTo("Wallace") == 0) {
                        uprs.updateString(namePos, "Doe");
                        uprs.updateRow();
                    }
                }

            } catch (SQLException e3) {
                System.err.println("Error selecting values from table EMPLOYEE");
                Throwable t = e3;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }
        }
}
```

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import com.mysql.jdbc.Connection;

public class JDBCSelect4 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException e) {
			System.err.println("Driver class not found");
			e.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = e1;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt =
				(Statement) con.createStatement(
					ResultSet.TYPE_SCROLL_SENSITIVE,
					ResultSet.CONCUR_UPDATABLE);
		} catch (SQLException e2) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e2;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
```

```
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }

            try {
                ResultSet uprs =
                    (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
                int namePos = uprs.findColumn("LNAME");

                while (uprs.isLast() == false) {
                    uprs.next();
                    if (uprs.getString(namePos).compareTo("Smith") == 0) {
                        uprs.deleteRow();
                    }
                }

            } catch (SQLException e3) {
                System.err.println("Error selecting values from table EMPLOYEE");
                Throwable t = e3;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }
        }
}
```

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCSelect7 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException cnfe) {
			System.err.println("Driver class not found");
			cnfe.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException e) {
			System.err.println("Error establishing database connection");
			Throwable t = e;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt =
				(Statement) con.createStatement(
					ResultSet.TYPE_SCROLL_SENSITIVE,
					ResultSet.CONCUR_UPDATABLE);
		} catch (SQLException e) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = e;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
```

```
            System.err.println("-----");
            t = t.getCause();
                }
            }

            try {
                ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
                rs.absolute(5);
                System.out.println(rs.getString("FNAME"));

                System.out.println("sleeping ...");
                Thread.sleep(6000);
                System.out.println("awake ...");

                if (rs.rowUpdated() == true) {
                    rs.refreshRow();
                    System.out.println(rs.getString("FNAME"));
                }

            } catch (SQLException e) {
                System.err.println("Error selecting values from table EMPLOYEE");
                Throwable t = e;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            } catch (InterruptedException ie) {
                ie.printStackTrace();
            }
        }
}
```


1600
1400
1200
1000
800
600
400
200
0
Sekunden
Insert
Select (ohne Ausgabe)
Select (mit Ausgabe)
Update
Delete
JDBC
Direktzugriff

Network Working Group Paul J. Leach, Microsoft
INTERNET-DRAFT Rich Salz, Certco
<draft-leach-uuids-guids-01.txt>
Category: Standards Track
Expires August 4, 1998 February 4, 1998

UUIDs and GUIDs

STATUS OF THIS MEMO

ABSTRACT

This specification defines the format of UUIDs (Universally Unique IDentifier), also known as GUIDs (Globally Unique IDentifier). A UUID is 128 bits long, and if generated according to the one of the mechanisms in this document, is either guaranteed to be different

from all other UUIDs/GUIDs generated until 3400 A.D. or extremely likely to be different (depending on the mechanism chosen). UUIDs were originally used in the Network Computing System (NCS) [1] and later in the Open Software Foundation's (OSF) Distributed Computing Environment [2].

This specification is derived from the latter specification with the kind permission of the OSF.

Table of Contents

[Page 1]

Internet-Draft UUIDs and GUIDs (DRAFT) 02/04/98

1. Introduction

This specification defines the format of UUIDs (Universally Unique IDentifiers), also known as GUIDs (Globally Unique IDentifiers). A UUID is 128 bits long, and if generated according to the one of the mechanisms in this document, is either guaranteed to be different from all other UUIDs/GUIDs generated until 3400 A.D. or extremely likely to be different (depending on the mechanism chosen).

2. Motivation

One of the main reasons for using UUIDs is that no centralized authority is required to administer them (beyond the one that allocates IEEE 802.1 node identifiers). As a result, generation on demand can be completely automated, and they can be used for a wide variety of purposes. The UUID generation algorithm described here supports very high allocation rates: 10 million per second per machine if you need it, so that they could even be used as transaction IDs.

UUIDs are fixed-size (128-bits) which is reasonably small relative to other alternatives. This fixed, relatively small size lends itself well to sorting, ordering, and hashing of all sorts, storing in databases, simple allocation, and ease of programming in general.

3. Specification

A UUID is an identifier that is unique across both space and time, with respect to the space of all UUIDs. To be precise, the UUID consists of a finite bit space. Thus the time value used for constructing a UUID is limited and will roll over in the future (approximately at A.D. 3400, based on the specified algorithm). A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network.

The generation of UUIDs does not require that a registration authority be contacted for each identifier. Instead, it requires a unique value over space for each UUID generator. This spatially unique value is specified as an IEEE 802 address, which is usually already available to network-connected systems. This 48-bit address can be assigned based on an address block obtained through the IEEE registration authority. This section of the UUID specification assumes the availability of an IEEE 802 address to a system desiring to generate a UUID, but if one is not available section 4 specifies a way to generate a probabilistically unique one that can not conflict with any properly assigned IEEE 802 address.

3.1 Format

In its most general form, all that can be said of the UUID format is that a UUID is 16 octets, and that some bits of octet 8 of the UUID called the variant field (specified in the next section) determine finer structure.

3.1.1 Variant

The variant field determines the layout of the UUID. That is, the interpretation of all other bits in the UUID depends on the setting of the bits in the variant field. The variant field consists of a variable number of the msbs of octet 8 of the UUID.

The following table lists the contents of the variant field.

Msb0	Msb1	Msb2	Description
0	-	-	Reserved, NCS backward compatibility.
1	0	-	The variant specified in this document.
1	1	0	Reserved, Microsoft Corporation backward compatibility

1 1 1 Reserved for future definition.

Other UUID variants may not interoperate with the UUID variant specified in this document, where interoperability is defined as the applicability of operations such as string conversion and lexical ordering across different systems. However, UUIDs allocated according to the stricture of different variants, though they may define different interpretations of the bits outside the variant field, will not result in duplicate UUID allocation, because of the differing values of the variant field itself.

The remaining fields described below (version, timestamp, etc.) are defined only for the UUID variant noted above.

3.1.2 UUID layout

The following table gives the format of a UUID for the variant specified herein. The UUID consists of a record of 16 octets. To minimize confusion about bit assignments within octets, the UUID record definition is defined only in terms of fields that are integral numbers of octets. The fields are in order of significance for comparison purposes, with "time_low" the most significant, and "node" the least significant.

Field	Data Type	Octet #	Note
time_low	unsigned 32 bit integer	0-3	The low field of the timestamp.
time_mid	unsigned 16 bit integer	4-5	The middle field of the timestamp.
time_hi_and_version	unsigned 16 bit integer	6-7	The high field of the timestamp multiplexed with the version number.

clock_seq_hi_and_rserved	unsigned 8 bit integer	8	The high field of the clock sequence multiplexed with the variant.
clock_seq_low	unsigned 8 bit integer	9	The low field of the clock sequence.
node	unsigned 48 bit integer	10-15	The spatially unique node identifier.

3.1.3 Version

The version number is in the most significant 4 bits of the time stamp (time_hi_and_version).

The following table lists currently defined versions of the UUID.

Msb0	Msb1	Msb2	Msb3	Version	Description
0	0	0	1	1	The time-based version specified in this document.
0	0	1	0	2	Reserved for DCE Security version, with embedded POSIX UIDs.
0	0	1	1	3	The name-based version specified in this document
0	1	0	0	4	The randomly or pseudo-randomly generated version specified in this document

3.1.4 Timestamp

The timestamp is a 60 bit value. For UUID version 1, this is represented by Coordinated Universal Time (UTC) as a count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582 (the date of Gregorian reform to the Christian calendar).

For systems that do not have UTC available, but do have local time, they MAY use local time instead of UTC, as long as they do so consistently throughout the system. This is NOT RECOMMENDED, however, and it should be noted that all that is needed to generate UTC, given local time, is a time zone offset.

For UUID version 3, it is a 60 bit value constructed from a name.

For UUID version 4, it is a randomly or pseudo-randomly generated 60 bit value.

3.1.5 Clock sequence

For UUID version 1, the clock sequence is used to help avoid duplicates that could arise when the clock is set backwards in time or if the node ID changes.

If the clock is set backwards, or even might have been set backwards (e.g., while the system was powered off), and the UUID generator can not be sure that no UUIDs were generated with timestamps larger than the value to which the clock was set, then the clock sequence has to be changed. If the previous value of the clock sequence is known, it can be just incremented; otherwise it should be set to a random or high-quality pseudo random value.

Similarly, if the node ID changes (e.g. because a network card has been moved between machines), setting the clock sequence to a random number minimizes the probability of a duplicate due to slight differences in the clock settings of the machines. (If the value of clock sequence associated with the changed node ID were known, then the clock sequence could just be incremented, but that is unlikely.)

The clock sequence MUST be originally (i.e., once in the lifetime of a system) initialized to a random number to minimize the correlation across systems. This provides maximum protection against node identifiers that may move or switch from system to system rapidly. The initial value MUST NOT be correlated to the node identifier.

For UUID version 3, it is a 14 bit value constructed from a name.

For UUID version 4, it is a randomly or pseudo-randomly generated 14 bit value.

3.1.6 Node

For UUID version 1, the node field consists of the IEEE address, usually the host address. For systems with multiple IEEE 802 addresses, any available address can be used. The lowest addressed octet (octet number 10) contains the global/local bit and the unicast/multicast bit, and is the first octet of the address transmitted on an 802.3 LAN.

For systems with no IEEE address, a randomly or pseudo-randomly generated value may be used (see section 4). The multicast bit must be set in such addresses, in order that they will never conflict with addresses obtained from network cards.

For UUID version 3, the node field is a 48 bit value constructed from a name.

For UUID version 4, the node field is a randomly or pseudo-randomly generated 48 bit value.

3.1.7 Nil UUID

The nil UUID is special form of UUID that is specified to have all 128 bits set to 0 (zero).

3.2 Algorithms for creating a time-based UUID

Various aspects of the algorithm for creating a version 1 UUID are discussed in the following sections. UUID generation requires a guarantee of uniqueness within the node ID for a given variant and version. Interoperability is provided by complying with the specified data structure.

3.2.1 Basic algorithm

The following algorithm is simple, correct, and inefficient:

. Obtain a system wide global lock

. From a system wide shared stable store (e.g., a file), read the UUID generator state: the values of the time stamp, clock sequence, and node ID used to generate the last UUID.

. Get the current time as a 60 bit count of 100-nanosecond intervals since 00:00:00.00, 15 October 1582

. Get the current node ID

. If the state was unavailable (non-existent or corrupted), or the saved node ID is different than the current node ID, generate a random clock sequence value

. If the state was available, but the saved time stamp is later than the current time stamp, increment the clock sequence value

. Format a UUID from the current time stamp, clock sequence, and node ID values according to the structure in section 3.1 (see section 3.2.6 for more details)

. Save the state (current time stamp, clock sequence, and node ID) back to the stable store

. Release the system wide global lock

If UUIDs do not need to be frequently generated, the above algorithm may be perfectly adequate. For higher performance requirements, however, issues with the basic algorithm include:

. Reading the state from stable storage each time is inefficient

. The resolution of the system clock may not be 100-nanoseconds

. Writing the state to stable storage each time is inefficient

. Sharing the state across process boundaries may be inefficient

Each of these issues can be addressed in a modular fashion by local improvements in the functions that read and write the state and read the clock. We address each of them in turn in the following sections.

3.2.2 Reading stable storage

The state only needs to be read from stable storage once at boot time, if it is read into a system wide shared volatile store (and updated whenever the stable store is updated).

If an implementation does not have any stable store available, then it can always say that the values were unavailable. This is the least desirable implementation, because it will increase the frequency of creation of new clock sequence numbers, which increases the probability of duplicates.

If the node ID can never change (e.g., the net card is inseparable from the system), or if any change also reinitializes the clock sequence to a random value, then instead of keeping it in stable store, the current node ID may be returned.

3.2.3 System clock resolution

The time stamp is generated from the system time, whose resolution may be less than the resolution of the UUID time stamp.

If UUIDs do not need to be frequently generated, the time stamp can simply be the system time multiplied by the number of 100-nanosecond intervals per system time interval.

If a system overruns the generator by requesting too many UUIDs within a single system time interval, the UUID service MUST either: return an error, or stall the UUID generator until the system clock catches up.

A high resolution time stamp can be simulated by keeping a count of how many UUIDs have been generated with the same value of the system time, and using it to construction the low-order bits of the time stamp. The count will range between zero and the number of 100-nanosecond intervals per system time interval.

Note: if the processors overrun the UUID generation frequently, additional node identifiers can be allocated to the system, which will permit higher speed allocation by making multiple UUIDs potentially available for each time stamp value.

3.2.4 Writing stable storage

The state does not always need to be written to stable store every time a UUID is generated. The timestamp in the stable store can be periodically set to a value larger than any yet used in a UUID; as long as the generated UUIDs have time stamps less than that value, and the clock sequence and node ID remain unchanged, only the shared volatile copy of the state needs to be updated. Furthermore, if the time stamp value in stable store is in the future by less than the typical time it takes the system to reboot, a crash will not cause a reinitialization of the clock sequence.

3.2.5 Sharing state across processes

If it is too expensive to access shared state each time a UUID is generated, then the system wide generator can be implemented to allocate a block of time stamps each time it is called, and a per-process generator can allocate from that block until it is exhausted.

3.2.6 UUID Generation details

UUIDs are generated according to the following algorithm:

- Determine the values for the UTC-based timestamp and clock sequence to be used in the UUID, as described above.

- For the purposes of this algorithm, consider the timestamp to be a 60-bit unsigned integer and the clock sequence to be a 14-bit unsigned integer. Sequentially number the bits in a field, starting from 0 (zero) for the least significant bit.

- Set the time_low field equal to the least significant 32-bits (bits numbered 0 to 31 inclusive) of the time stamp in the same order of significance.

- Set the time_mid field equal to the bits numbered 32 to 47 inclusive of the time stamp in the same order of significance.

- Set the 12 least significant bits (bits numbered 0 to 11 inclusive) of the time_hi_and_version field equal to the bits numbered 48 to 59 inclusive of the time stamp in the same order of significance.

- Set the 4 most significant bits (bits numbered 12 to 15 inclusive) of the time_hi_and_version field to the 4-bit version number corresponding to the UUID version being created, as shown in the table in section 3.1.3.

- Set the clock_seq_low field to the 8 least significant bits (bits numbered 0 to 7 inclusive) of the clock sequence in the same order of significance.

- Set the 6 least significant bits (bits numbered 0 to 5 inclusive) of the clock_seq_hi_and_reserved field to the 6 most significant bits (bits numbered 8 to 13 inclusive) of the clock sequence in the same order of significance.

- Set the 2 most significant bits (bits numbered 6 and 7) of the clock_seq_hi_and_reserved to 0 and 1, respectively.

- Set the node field to the 48-bit IEEE address in the same order of significance as the address.

3.3 Algorithm for creating a name-based UUID

The version 3 UUID is meant for generating UUIDs from "names" that are drawn from, and unique within, some "name space". Some examples of names (and, implicitly, name spaces) might be DNS names, URLs, ISO Object IDs (OIDs), reserved words in a programming language, or X.500

Distinguished Names (DNs); thus, the concept of name and name space should be broadly construed, and not limited to textual names. The mechanisms or conventions for allocating names from, and ensuring their uniqueness within, their name spaces are beyond the scope of this specification.

The requirements for such UUIDs are as follows:

. The UUIDs generated at different times from the same name in the same namespace MUST be equal

. The UUIDs generated from two different names in the same namespace should be different (with very high probability)

. The UUIDs generated from the same name in two different namespaces should be different with (very high probability)

. If two UUIDs that were generated from names are equal, then they were generated from the same name in the same namespace (with very high probability).

The algorithm for generating the a UUID from a name and a name space are as follows:

. Allocate a UUID to use as a "name space ID" for all UUIDs generated from names in that name space

. Convert the name to a canonical sequence of octets (as defined by the standards or conventions of its name space); put the name space ID in network byte order

. Compute the MD5 [3] hash of the name space ID concatenated with the name

. Set octets 0-3 of time_low field to octets 0-3 of the MD5 hash

. Set octets 0-1 of time_mid field to octets 4-5 of the MD5 hash

. Set octets 0-1 of time_hi_and_version field to octets 6-7 of the

MD5 hash

. Set the clock_seq_hi_and_reserved field to octet 8 of the MD5 hash

. Set the clock_seq_low field to octet 9 of the MD5 hash

. Set octets 0-5 of the node field to octets 10-15 of the MD5 hash

. Set the 2 most significant bits (bits numbered 6 and 7) of the clock_seq_hi_and_reserved to 0 and 1, respectively.

. Set the 4 most significant bits (bits numbered 12 to 15 inclusive) of the time_hi_and_version field to the 4-bit version number corresponding to the UUID version being created, as shown in the table above.

. Convert the resulting UUID to local byte order.

3.4 Algorithms for creating a UUID from truly random or pseudo-random numbers

The version 4 UUID is meant for generating UUIDs from truly-random or pseudo-random numbers.

The algorithm is as follows:

. Set the 2 most significant bits (bits numbered 6 and 7) of the clock_seq_hi_and_reserved to 0 and 1, respectively.

. Set the 4 most significant bits (bits numbered 12 to 15 inclusive) of the time_hi_and_version field to the 4-bit version number corresponding to the UUID version being created, as shown in the table above.

. Set all the other bits to randomly (or pseudo-randomly) chosen values.

Here are several possible ways to generate the random values:

. Use a physical source of randomness: for example, a white noise generator, radioactive decay, or a lava lamp.

. Use a cryptographic strength random number generator.

3.5 String Representation of UUIDs

For use in human readable text, a UUID string representation is specified as a sequence of fields, some of which are separated by single dashes.

Each field is treated as an integer and has its value printed as a zero-filled hexadecimal digit string with the most significant digit first. The hexadecimal values a to f inclusive are output as lower case characters, and are case insensitive on input. The sequence is the same as the UUID constructed type.

The formal definition of the UUID string representation is provided by the following extended BNF:

```
UUID                   = <time_low> "-" <time_mid> "-"
                         <time_high_and_version> "-"
                         <clock_seq_and_reserved>
                         <clock_seq_low> "-" <node>
time_low               = 4*<hexOctet>
time_mid               = 2*<hexOctet>
time_high_and_version  = 2*<hexOctet>
clock_seq_and_reserved = <hexOctet>
clock_seq_low          = <hexOctet>
node                   = 6*<hexOctet
hexOctet               = <hexDigit> <hexDigit>
hexDigit =
      "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
      | "a" | "b" | "c" | "d" | "e" | "f"
      | "A" | "B" | "C" | "D" | "E" | "F"
```

The following is an example of the string representation of a UUID:

f81d4fae-7dec-11d0-a765-00a0c91e6bf6

3.6 Comparing UUIDs for equality

Consider each field of the UUID to be an unsigned integer as shown in the table in section 3.1. Then, to compare a pair of UUIDs, arithmetically compare the corresponding fields from each UUID in order of significance and according to their data type. Two UUIDs are equal if and only if all the corresponding fields are equal.

Note: as a practical matter, on many systems comparison of two UUIDs for equality can be performed simply by comparing the 128 bits of their in-memory representation considered as a 128 bit unsigned integer. Here, it is presumed that by the time the in-memory representation is obtained the appropriate byte-order canonicalizations have been carried out.

3.7 Comparing UUIDs for relative order

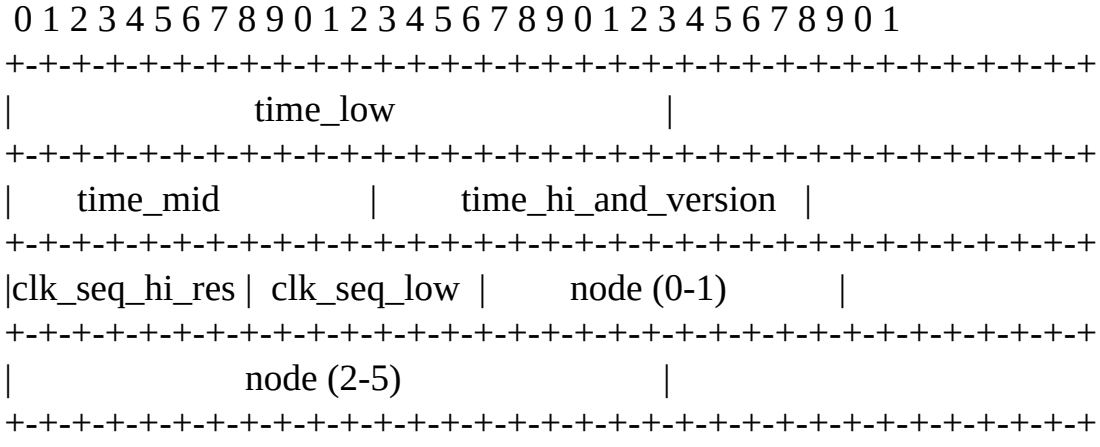
Two UUIDs allocated according to the same variant can also be ordered lexicographically. For the UUID variant herein defined, the first of two UUIDs follows the second if the most significant field in which the UUIDs differ is greater for the first UUID. The first of a pair of UUIDs precedes the second if the most significant field in which the UUIDs differ is greater for the second UUID.

3.8 Byte order of UUIDs

UUIDs may be transmitted in many different forms, some of which may be dependent on the presentation or application protocol where the UUID may be used. In such cases, the order, sizes and byte orders of the UUIDs fields on the wire will depend on the relevant presentation or application protocol. However, it is strongly RECOMMENDED that the order of the fields conform with ordering set out in section 3.1 above. Furthermore, the payload size of each field in the application or presentation protocol MUST be large enough that no information lost in the process of encoding them for transmission.

In the absence of explicit application or presentation protocol specification to the contrary, a UUID is encoded as a 128-bit object,

as follows: the fields are encoded as 16 octets, with the sizes and order of the fields defined in section 3.1, and with each field encoded with the Most Significant Byte first (also known as network byte order).



4. Node IDs when no IEEE 802 network card is available

If a system wants to generate UUIDs but has no IEE 802 compliant network card or other source of IEEE 802 addresses, then this section describes how to generate one.

The ideal solution is to obtain a 47 bit cryptographic quality random number, and use it as the low 47 bits of the node ID, with the most significant bit of the first octet of the node ID set to 1. This bit is the unicast/multicast bit, which will never be set in IEEE 802 addresses obtained from network cards; hence, there can never be a conflict between UUIDs generated by machines with and without network cards.

If a system does not have a primitive to generate cryptographic quality random numbers, then in most systems there are usually a

fairly large number of sources of randomness available from which one can be generated. Such sources are system specific, but often include:

- the percent of memory in use
- the size of main memory in bytes
- the amount of free main memory in bytes
- the size of the paging or swap file in bytes
- free bytes of paging or swap file
- the total size of user virtual address space in bytes
- the total available user address space bytes
- the size of boot disk drive in bytes
- the free disk space on boot drive in bytes
- the current time
- the amount of time since the system booted
- the individual sizes of files in various system directories
- the creation, last read, and modification times of files in various system directories
- the utilization factors of various system resources (heap, etc.)
- current mouse cursor position
- current caret position
- current number of running processes, threads
- handles or IDs of the desktop window and the active window
- the value of stack pointer of the caller
- the process and thread ID of caller
- various processor architecture specific performance counters (instructions executed, cache misses, TLB misses)

(Note that it precisely the above kinds of sources of randomness that are used to seed cryptographic quality random number generators on systems without special hardware for their construction.)

In addition, items such as the computer's name and the name of the operating system, while not strictly speaking random, will help differentiate the results from those obtained by other systems.

The exact algorithm to generate a node ID using these data is system specific, because both the data available and the functions to obtain

them are often very system specific. However, assuming that one can concatenate all the values from the randomness sources into a buffer, and that a cryptographic hash function such as MD5 [3] is available, then any 6 bytes of the MD5 hash of the buffer, with the multicast bit (the high bit of the first byte) set will be an appropriately random node ID.

Other hash functions, such as SHA-1 [4], can also be used. The only requirement is that the result be suitably random _ in the sense that the outputs from a set uniformly distributed inputs are themselves uniformly distributed, and that a single bit change in the input can be expected to cause half of the output bits to change.

5. Obtaining IEEE 802 addresses

At the time of writing, the following URL

http://standards.ieee.org/db/oui/forms/

contains information on how to obtain an IEEE 802 address block. At the time of writing, the cost is $1250 US.

6. Security Considerations

It should not be assumed that UUIDs are hard to guess; they should not be used as capabilities.

7. Acknowledgements

This document draws heavily on the OSF DCE specification for UUIDs. Ted Ts'o provided helpful comments, especially on the byte ordering section which we mostly plagiarized from a proposed wording he supplied (all errors in that section are our responsibility, however).

8. References

[1] Lisa Zahn, et. al., Network Computing Architecture, Prentice Hall, Englewood Cliffs, NJ, 1990

[2] DCE: Remote Procedure Call, Open Group CAE Specification C309 ISBN 1-85912-041-5 28cm. 674p. pbk. 1,655g. 8/94

[3] R. Rivest, RFC 1321, "The MD5 Message-Digest Algorithm", 04/16/1992.

[4] NIST FIPS PUB 180-1, "Secure Hash Standard," National Institute of Standards and Technology, U.S. Department of Commerce, DRAFT, May 31, 1994.

9. Authors' addresses

Paul J. Leach
Microsoft
1 Microsoft Way
Redmond, WA, 98052, U.S.A.
paulle@microsoft.com
Tel. 425 882 8080
Fax. 425 936 7329

Rich Salz
100 Cambridge Park Drive
Cambridge MA 02140
salzr@certco.com
Tel. 617 499 4075
Fax. 617 576 0019

10. Notice

obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

11. Full Copyright Statement

Appendix A _ UUID Sample Implementation

This implementation consists of 5 files: uuid.h, uuid.c, sysdep.h, sysdep.c and utest.c. The uuid.* files are the system independent implementation of the UUID generation algorithms described above, with all the optimizations described above except efficient state sharing across processes included. The code has been tested on Linux (Red Hat 4.0) with GCC (2.7.2), and Windows NT 4.0 with VC++ 5.0. The code assumes 64 bit integer support, which makes it a lot clearer.

All the following source files should be considered to have the following copyright notice included:

copyrt.h

```
/*
** Copyright (c) 1990- 1993, 1996 Open Software Foundation, Inc.
** Copyright (c) 1989 by Hewlett-Packard Company, Palo Alto, Ca. &
** Digital Equipment Corporation, Maynard, Mass.
** Copyright (c) 1998 Microsoft.
** To anyone who acknowledges that this file is provided "AS IS"
** without any express or implied warranty: permission to use, copy,
** modify, and distribute this file for any purpose is hereby
** granted without fee, provided that the above copyright notices and
** this notice appears in all source code copies, and that none of
** the names of Open Software Foundation, Inc., Hewlett-Packard
** Company, or Digital Equipment Corporation be used in advertising
** or publicity pertaining to distribution of the software without
** specific, written prior permission.  Neither Open Software
** Foundation, Inc., Hewlett-Packard Company, Microsoft, nor Digital
Equipment
** Corporation makes any representations about the suitability of
** this software for any purpose.
*/
```

uuid.h

```
#include "copyrt.h"
#undef uuid_t
typedef struct _uuid_t {
    unsigned32          time_low;
    unsigned16          time_mid;
    unsigned16          time_hi_and_version;
    unsigned8           clock_seq_hi_and_reserved;
    unsigned8           clock_seq_low;
    byte                node[6];
} uuid_t;

/* uuid_create -- generate a UUID */
int uuid_create(uuid_t * uuid);

/* uuid_create_from_name -- create a UUID using a "name"
   from a "name space" */
void uuid_create_from_name(
  uuid_t * uuid,        /* resulting UUID */
  uuid_t nsid,          /* UUID to serve as context, so identical
                        names from different name spaces generate
                        different UUIDs */
  void * name,          /* the name from which to generate a UUID */
  int namelen           /* the length of the name */
);

/* uuid_compare --  Compare two UUID's "lexically" and return
        -1   u1 is lexically before u2
         0   u1 is equal to u2
         1   u1 is lexically after u2
   Note:   lexical ordering is not temporal ordering!
*/
int uuid_compare(uuid_t *u1, uuid_t *u2);
```

uuid.c

```
#include "copyrt.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "sysdep.h"
#include "uuid.h"

/* various forward declarations */
```

```
static int read_state(unsigned16 *clockseq, uuid_time_t *timestamp,
uuid_node_t * node);
static void write_state(unsigned16 clockseq, uuid_time_t timestamp,
uuid_node_t node);
static void format_uuid_v1(uuid_t * uuid, unsigned16 clockseq,
uuid_time_t timestamp, uuid_node_t node);
static void format_uuid_v3(uuid_t * uuid, unsigned char hash[16]);
static void get_current_time(uuid_time_t * timestamp);
static unsigned16 true_random(void);
```



```
/* uuid_create -- generator a UUID */
int uuid_create(uuid_t * uuid) {
  uuid_time_t timestamp, last_time;
  unsigned16 clockseq;
  uuid_node_t node;
  uuid_node_t last_node;
  int f;

  /* acquire system wide lock so we're alone */
  LOCK;

  /* get current time */
  get_current_time(&timestamp);

  /* get node ID */
  get_ieee_node_identifier(&node);

  /* get saved state from NV storage */
  f = read_state(&clockseq, &last_time, &last_node);

  /* if no NV state, or if clock went backwards, or node ID changed
     (e.g., net card swap) change clockseq */
  if (!f || memcmp(&node, &last_node, sizeof(uuid_node_t)))
      clockseq = true_random();
  else if (timestamp < last_time)
      clockseq++;

  /* stuff fields into the UUID */
```

```
  format_uuid_v1(uuid, clockseq, timestamp, node);

  /* save the state for next time */
  write_state(clockseq, timestamp, node);

  UNLOCK;
  return(1);
};

/* format_uuid_v1 -- make a UUID from the timestamp, clockseq,
                    and node ID */
void format_uuid_v1(uuid_t * uuid, unsigned16 clock_seq, uuid_time_t
timestamp, uuid_node_t node) {
    /* Construct a version 1 uuid with the information we've gathered
     * plus a few constants. */
  uuid->time_low = (unsigned long)(timestamp & 0xFFFFFFFF);
    uuid->time_mid = (unsigned short)((timestamp >> 32) & 0xFFFF);
    uuid->time_hi_and_version = (unsigned short)((timestamp >> 48) &
      0x0FFF);
    uuid->time_hi_and_version |= (1 << 12);
    uuid->clock_seq_low = clock_seq & 0xFF;
    uuid->clock_seq_hi_and_reserved = (clock_seq & 0x3F00) >> 8;
    uuid->clock_seq_hi_and_reserved |= 0x80;
    memcpy(&uuid->node, &node, sizeof uuid->node);
};
```



```
/* data type for UUID generator persistent state */
typedef struct {
  uuid_time_t ts;       /* saved timestamp */
  uuid_node_t node;     /* saved node ID */
  unsigned16 cs;        /* saved clock sequence */
  } uuid_state;

static uuid_state st;

/* read_state -- read UUID generator state from non-volatile store */
int read_state(unsigned16 *clockseq, uuid_time_t *timestamp,
uuid_node_t *node) {
```

```
  FILE * fd;
  static int inited = 0;

  /* only need to read state once per boot */
  if (!inited) {
     fd = fopen("state", "rb");
     if (!fd)
         return (0);
     fread(&st, sizeof(uuid_state), 1, fd);
     fclose(fd);
     inited = 1;
  };
  *clockseq = st.cs;
  *timestamp = st.ts;
  *node = st.node;
  return(1);
};

/* write_state -- save UUID generator state back to non-volatile
storage */
void write_state(unsigned16 clockseq, uuid_time_t timestamp,
uuid_node_t node) {
  FILE * fd;
  static int inited = 0;
  static uuid_time_t next_save;

  if (!inited) {
     next_save = timestamp;
     inited = 1;
  };
  /* always save state to volatile shared state */
  st.cs = clockseq;
  st.ts = timestamp;
  st.node = node;
  if (timestamp >= next_save) {
     fd = fopen("state", "wb");
     fwrite(&st, sizeof(uuid_state), 1, fd);
     fclose(fd);
     /* schedule next save for 10 seconds from now */
     next_save = timestamp + (10 * 10 * 1000 * 1000);
  };
};
```

```
/* get-current_time -- get time as 60 bit 100ns ticks since whenever.
  Compensate for the fact that real clock resolution is
  less than 100ns. */
void get_current_time(uuid_time_t * timestamp) {
    uuid_time_t               time_now;
    static uuid_time_t  time_last;
    static unsigned16   uuids_this_tick;
  static int                inited = 0;

  if (!inited) {
        get_system_time(&time_now);
      uuids_this_tick = UUIDS_PER_TICK;
      inited = 1;
  };

    while (1) {
        get_system_time(&time_now);

      /* if clock reading changed since last UUID generated... */
        if (time_last != time_now) {
          /* reset count of uuids gen'd with this clock reading */
           uuids_this_tick = 0;
          break;
      };
        if (uuids_this_tick < UUIDS_PER_TICK) {
          uuids_this_tick++;
          break;
      };
      /* going too fast for our clock; spin */
    };
  /* add the count of uuids to low order bits of the clock reading */
  *timestamp = time_now + uuids_this_tick;
};

/* true_random -- generate a crypto-quality random number.
   This sample doesn't do that. */
static unsigned16
true_random(void)
{
  static int inited = 0;
  uuid_time_t time_now;
```

```
  if (!inited) {
     get_system_time(&time_now);
     time_now = time_now/UUIDS_PER_TICK;
     srand((unsigned int)(((time_now >> 32) ^ time_now)&0xffffffff));
     inited = 1;
  };

    return (rand());
}
```



```
/* uuid_create_from_name -- create a UUID using a "name" from a "name
space" */
void uuid_create_from_name(
  uuid_t * uuid,       /* resulting UUID */
  uuid_t nsid,         /* UUID to serve as context, so identical
                       names from different name spaces generate
                       different UUIDs */
  void * name,         /* the name from which to generate a UUID */
  int namelen          /* the length of the name */
) {
  MD5_CTX c;
  unsigned char hash[16];
  uuid_t net_nsid;     /* context UUID in network byte order */

  /* put name space ID in network byte order so it hashes the same
     no matter what endian machine we're on */
  net_nsid = nsid;
  htonl(net_nsid.time_low);
  htons(net_nsid.time_mid);
  htons(net_nsid.time_hi_and_version);

  MD5Init(&c);
  MD5Update(&c, &net_nsid, sizeof(uuid_t));
  MD5Update(&c, name, namelen);
  MD5Final(hash, &c);
```

```
  /* the hash is in network byte order at this point */
  format_uuid_v3(uuid, hash);
};

/* format_uuid_v3 -- make a UUID from a (pseudo)random 128 bit number
*/
void format_uuid_v3(uuid_t * uuid, unsigned char hash[16]) {
    /* Construct a version 3 uuid with the (pseudo-)random number
     * plus a few constants. */

    memcpy(uuid, hash, sizeof(uuid_t));

  /* convert UUID to local byte order */
  ntohl(uuid->time_low);
  ntohs(uuid->time_mid);
  ntohs(uuid->time_hi_and_version);

  /* put in the variant and version bits */
    uuid->time_hi_and_version &= 0x0FFF;
    uuid->time_hi_and_version |= (3 << 12);
    uuid->clock_seq_hi_and_reserved &= 0x3F;
    uuid->clock_seq_hi_and_reserved |= 0x80;
};

/* uuid_compare --  Compare two UUID's "lexically" and return
       -1   u1 is lexically before u2
        0   u1 is equal to u2
        1   u1 is lexically after u2
```



```
    Note:   lexical ordering is not temporal ordering!
*/
int uuid_compare(uuid_t *u1, uuid_t *u2)
{
  int i;

#define CHECK(f1, f2) if (f1 != f2) return f1 < f2 ? -1 : 1;
  CHECK(u1->time_low, u2->time_low);
  CHECK(u1->time_mid, u2->time_mid);
  CHECK(u1->time_hi_and_version, u2->time_hi_and_version);
```

```
  CHECK(u1->clock_seq_hi_and_reserved, u2->clock_seq_hi_and_reserved);
  CHECK(u1->clock_seq_low, u2->clock_seq_low)
  for (i = 0; i < 6; i++) {
     if (u1->node[i] < u2->node[i])
         return -1;
     if (u1->node[i] > u2->node[i])
     return 1;
    }
  return 0;
};
```

sysdep.h

```
#include "copyrt.h"
/* remove the following define if you aren't running WIN32 */
#define WININC 0

#ifdef WININC
#include <windows.h>
#else
#include <sys/types.h>
#include <sys/time.h>
#include <sys/sysinfo.h>
#endif

/* change to point to where MD5 .h's live */
/* get MD5 sample implementation from RFC 1321 */
#include "global.h"
#include "md5.h"

/* set the following to the number of 100ns ticks of the actual
resolution of
your system's clock */
#define UUIDS_PER_TICK 1024

/* Set the following to a call to acquire a system wide global lock
*/
#define LOCK
#define UNLOCK

typedef unsigned long   unsigned32;
typedef unsigned short  unsigned16;
typedef unsigned char   unsigned8;
typedef unsigned char   byte;
```

```
/* Set this to what your compiler uses for 64 bit data type */
#ifdef WININC
#define unsigned64_t unsigned __int64
#define I64(C) C
#else
#define unsigned64_t unsigned long long
#define I64(C) C##LL
#endif


typedef unsigned64_t uuid_time_t;
typedef struct {
  char nodeID[6];
} uuid_node_t;

void get_ieee_node_identifier(uuid_node_t *node);
void get_system_time(uuid_time_t *uuid_time);
void get_random_info(char seed[16]);


sysdep.c

#include "copyrt.h"
#include <stdio.h>
#include "sysdep.h"

/* system dependent call to get IEEE node ID.
   This sample implementation generates a random node ID
   */
void get_ieee_node_identifier(uuid_node_t *node) {
  char seed[16];
  FILE * fd;
  static inited = 0;
  static uuid_node_t saved_node;

  if (!inited) {
     fd = fopen("nodeid", "rb");
     if (fd) {
         fread(&saved_node, sizeof(uuid_node_t), 1, fd);
```

```
        fclose(fd);
    }
    else {
        get_random_info(seed);
        seed[0] |= 0x80;
        memcpy(&saved_node, seed, sizeof(uuid_node_t));
        fd = fopen("nodeid", "wb");
        if (fd) {
            fwrite(&saved_node, sizeof(uuid_node_t), 1, fd);
            fclose(fd);
        };
    };
    inited = 1;
 };
```



```
  *node = saved_node;
};

/* system dependent call to get the current system time.
   Returned as 100ns ticks since Oct 15, 1582, but resolution may be
   less than 100ns.
*/
#ifdef _WINDOWS_

void get_system_time(uuid_time_t *uuid_time) {
  ULARGE_INTEGER time;

  GetSystemTimeAsFileTime((FILETIME *)&time);

    /* NT keeps time in FILETIME format which is 100ns ticks since
     Jan 1, 1601.  UUIDs use time in 100ns ticks since Oct 15, 1582.
     The difference is 17 Days in Oct + 30 (Nov) + 31 (Dec)
     + 18 years and 5 leap days.
  */

    time.QuadPart +=
        (unsigned __int64) (1000*1000*10)       // seconds
      * (unsigned __int64) (60 * 60 * 24)       // days
      * (unsigned __int64) (17+30+31+365*18+5); // # of days
```

```
  *uuid_time = time.QuadPart;

};

void get_random_info(char seed[16]) {
  MD5_CTX c;
  typedef struct {
      MEMORYSTATUS m;
      SYSTEM_INFO s;
      FILETIME t;
      LARGE_INTEGER pc;
      DWORD tc;
      DWORD l;
      char hostname[MAX_COMPUTERNAME_LENGTH + 1];
  } randomness;
  randomness r;

  MD5Init(&c);
  /* memory usage stats */
  GlobalMemoryStatus(&r.m);
  /* random system stats */
  GetSystemInfo(&r.s);
  /* 100ns resolution (nominally) time of day */
  GetSystemTimeAsFileTime(&r.t);
  /* high resolution performance counter */
  QueryPerformanceCounter(&r.pc);
  /* milliseconds since last boot */
  r.tc = GetTickCount();
  r.l = MAX_COMPUTERNAME_LENGTH + 1;
```



```
  GetComputerName(r.hostname, &r.l );
  MD5Update(&c, &r, sizeof(randomness));
  MD5Final(seed, &c);
};
#else

void get_system_time(uuid_time_t *uuid_time)
{
```

```
    struct timeval tp;

    gettimeofday(&tp, (struct timezone *)0);

    /* Offset between UUID formatted times and Unix formatted times.
       UUID UTC base time is October 15, 1582.
       Unix base time is January 1, 1970.
    */
    *uuid_time = (tp.tv_sec * 10000000) + (tp.tv_usec * 10) +
      I64(0x01B21DD213814000);
};

void get_random_info(char seed[16]) {
  MD5_CTX c;
  typedef struct {
      struct sysinfo s;
      struct timeval t;
      char hostname[257];
  } randomness;
  randomness r;

  MD5Init(&c);
  sysinfo(&r.s);
  gettimeofday(&r.t, (struct timezone *)0);
  gethostname(r.hostname, 256);
  MD5Update(&c, &r, sizeof(randomness));
  MD5Final(seed, &c);
};

#endif
```

utest.c

```
#include "copyrt.h"
#include "sysdep.h"
#include <stdio.h>
#include "uuid.h"

uuid_t NameSpace_DNS = { /* 6ba7b810-9dad-11d1-80b4-00c04fd430c8 */
    0x6ba7b810,
    0x9dad,
    0x11d1,
    0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
  };
```

```
/* puid -- print a UUID */
void puid(uuid_t u);

/* Simple driver for UUID generator */
void main(int argc, char **argv) {
  uuid_t u;
  int f;

  uuid_create(&u);
  printf("uuid_create()            -> "); puid(u);

  f = uuid_compare(&u, &u);
  printf("uuid_compare(u,u): %d\n", f);     /* should be 0 */
  f = uuid_compare(&u, &NameSpace_DNS);
  printf("uuid_compare(u, NameSpace_DNS): %d\n", f); /* s.b. 1 */
  f = uuid_compare(&NameSpace_DNS, &u);
  printf("uuid_compare(NameSpace_DNS, u): %d\n", f); /* s.b. -1 */

  uuid_create_from_name(&u, NameSpace_DNS, "www.widgets.com", 15);
  printf("uuid_create_from_name() -> "); puid(u);
};

void puid(uuid_t u) {
  int i;

  printf("%8.8x-%4.4x-%4.4x-%2.2x%2.2x-", u.time_low, u.time_mid,
     u.time_hi_and_version, u.clock_seq_hi_and_reserved,
     u.clock_seq_low);
  for (i = 0; i < 6; i++)
     printf("%2.2x", u.node[i]);
  printf("\n");
};
```

Appendix B _ Sample output of utest

```
uuid_create()            -> 7d444840-9dc0-11d1-b245-5ffdce74fad2
uuid_compare(u,u): 0
uuid_compare(u, NameSpace_DNS): 1
```

```
uuid_compare(NameSpace_DNS, u): -1
uuid_create_from_name()   -> e902893a-9d22-3c7e-a7b8-d6e313b71d9f
```

Appendix C _ Some name space IDs

This appendix lists the name space IDs for some potentially interesting name spaces, as initialized C structures and in the string representation defined in section 3.5

```
uuid_t NameSpace_DNS = { /* 6ba7b810-9dad-11d1-80b4-00c04fd430c8 */
    0x6ba7b810,
    0x9dad,
    0x11d1,
    0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
  };
```



```
uuid_t NameSpace_URL = { /* 6ba7b811-9dad-11d1-80b4-00c04fd430c8 */
    0x6ba7b811,
    0x9dad,
    0x11d1,
    0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
  };

uuid_t NameSpace_OID = { /* 6ba7b812-9dad-11d1-80b4-00c04fd430c8 */
    0x6ba7b812,
    0x9dad,
    0x11d1,
    0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
  };

uuid_t NameSpace_X500 = { /* 6ba7b814-9dad-11d1-80b4-00c04fd430c8 */
    0x6ba7b814,
    0x9dad,
    0x11d1,
    0x80, 0xb4, 0x00, 0xc0, 0x4f, 0xd4, 0x30, 0xc8
  };
```

ISO Catalogue

ICS fields

35
Information technology. Office machines

35.060
Languages used in information technology

View Shopping Basket

Search options

- (•) Text
- ( ) ISO Number

Type in search string

Start Search

Help on using search
Extended Search

- How to use the catalogue
- Maintenance agencies and Registration Authorities
- List of withdrawn standards

The URL you've requested doesn't link to a valid catalogue entry.

Hinweis

Aufgrund der verfügbaren Menge guter einführender (auch deutschsprachiger) Datenbankliteratur verzichtet das vorliegende Scriptum darauf den in der Literatur verfügbaren Stoff nochmals aufzubereiten, sondern versammelt die Definitionen, Beispiele und Anmerkungen der Vorlesungen in einer übersichtlichen Zusammenstellung.
Für die vertiefende ausführliche lehrbuchartige Darstellung des Stoffes sei auf die empfohlene Literatur verwiesen.

1 Motivation und Einführung

1.1 Begriffsbestimmung: Was ist eine Datenbank?

Motivation für die Einführung einer Datenbank anstatt selbsterstellter Verwaltungs- und Zugriffsroutinen:

- Daten-Programm-Unabhängigkeit.
  Die verwalteten Daten sollen unabhängig vom sie verarbeiteten Programm gespeichert und zugreifbar sein.
  Dies wäre zwar in einem ersten Schritt auch durch die Verwendung des Dateisystems möglich, allerdings würde hierfür ein Programm zur Abbildung der Programmdaten auf die Dateistrukturen benötigt, welches selbst wieder eine Abhängigkeitsbeziehung zwischen Daten und Programm --- nun eben dem Abbildungsprogramm --- darstellen würde.
- Flexible Speicherung.
  Datenbankmanagementsysteme speichern die verwalteten Daten deutlich flexibler als selbsterstellte Routinen und sind somit hinsichtlich der Zukunftsfähigkeit effizienter.
- Verwaltungsfunktionen.
  Datenbankmanagementsysteme bieten in der Regel eine Reihe über die reine Datenverwaltung hinausgehende Funktionen wie Backup-Recovery, Integritätssicherung, Synchronisation gleichzeitiger Zugriffe oder Transaktionskontrolle an, die nicht selbständig implementiert werden

müssen.

Grundlegende Begriffe

Definition 1: *Daten*
Daten sind durch die Maschine verarbeitbare Einheiten.

Definition 2: *Information*
Daten die Bedeutung für den Empfänger besitzen.
Nach Shannon ermißt sich der Wert einer Information durch den Zuwachs der durch den Adressaten nach Kenntnis der Information beantwortbaren Ja/Nein-Fragen.

Mehr zum Unterschied zwischen *Daten* und *Information*:

- Homepage des Arbeitskreises *Bildung*, einem Zusammenschluß von Stipendiaten der Friedrich Naumann Stiftung
- Glossar der deutschsprachigen Anleitung zu PGP

Definition 3: *Datenbank*
Eine Datenbank (engl. *data base*) ist ein integrierter, persistenter Datenbestand einschließlich aller relevanten Informationen über die dargestellten Information (sog. Metainformation, d.h. Integritätsbedingungen und Regeln), der einer Gruppe von Benutzern in nur einem Exemplar zur Verfügung steht und durch ein DBMS verwaltetet wird.

Definition 4: *Datenbankmanagementsystem (DBMS)*
Ein Datenbankmanagementsystem (DBMS) ist die Gesamtheit aller Programme zur Erzeugung, Verwaltung und Manipulation einer Datenbank.
Im Deutschen wird auch der Begriff *Datenbankverwaltungssystem* (DBVS) synonym verwendet.

Beispiele verfügbarer DBMS:

- Die DBMS-Produkte des Herstellers Sybase
- IDMS von Computer Associates
- IMS, DB2 und Informix von IBM
- MySQL des gleichnamigen Herstellers
- Oracle 9i des Herstellers Oracle
- SQLServer und Access des Herstellers Microsoft

Beispiel 1: Am Markt verfügbare DBM-Systeme

Definition 5: *Relationales DBMS*
Ein relationales Datenbankmanagementsystem (RDBMS) ist ein DBMS, welches intern gemäß dem relationalen Modell organisiert ist.

Bei den genannten DBMS *MySQL*, *SQLServer*, *Access*, *DB2* und *Oracle* handelt es sich um relationale Systeme, bzw. Weiterentwicklungen davon.

Beispiel 2: Am Markt verfügbare RDBM-Systeme

Definition 6: *Relation*
Eine Relation $R(A_1, A_2 \ldots A_n)$ ist eine benannte Menge von *n*-Tupeln, wobei ein *n*-Tupel eine Anordnung von *n* atomaren, d.h. einfachen (nicht weiter zerlegbaren) Attributen $A_1, A_2 \ldots A_n$ ist.

Die Relation *Person* mit den Attributen *Vorname*, *Nachname* und *Geburtsdatum*.
Werteausprägungen davon:
$Person_1$("Meier", "Schorsch", "1955-10-01")
$Person_2$("Huber", "Franz", "1945-08-03")
...

Die Relation *Student* mit den Attributen *Name*, *Matrikelnummer*, *Semester* und *regelmäßigerMensabesucher*.
Werteausprägungen davon:
$Student_1$("Meier Schorsch", "08154711", "WIB 1", "true")
$Student_2$("Müller Xaver", "73619452", "BCM 4", "false")

...

Beispiel 3: Relationen

Definition 7: *Tabelle*
Eine Tabelle unterscheidet sich von einer Relation darin, daß ein Tupel mehrfach auftreten darf; eine Tabelle ist mathematisch keine Menge.

Die Tabelle *Student* mit den Attributen *Name*, *Matrikelnummer*, *Semester* und *regelmäßigerMensabesucher*.
Werteausprägungen davon:
$Student_1$("Meier Schorsch", "08154711", "WIB 1", "true")
$Student_2$("Müller Xaver", "73619452", "BCM 4", "false")
$Student_3$("Müller Xaver", "73619452", "BCM 4", "false")
...

Man beachte, daß der dritte Eintrag doppelt vorkommt, d.h. in all seinen Wertbelegungen mit dem zweiten übereinstimmt.

Beispiel 4: Tabelle

Definition 8: *Modell*
Ein Modell bildet einen existierenden Sachverhalt deskriptiv nach oder nimmt einen Zukünftigen präskriptiv voraus.
Teilweise wird der Begriff *Schema* synonym gebraucht.

Deskriptive Modelle: Modelleisenbahn, Stadtplan, Photo.
Präskriptive Modelle: Bauplan eines Hauses, Skizze eines Gemäldes, maßstäblich verkleinerte Skulptur als Vorbild.

Beispiel 5: Modelle

Definition 9: *Datenbanksprache*
Eine Sprache die zur Erzeugung oder Interaktion mit den Daten bzw. zu deren Verwaltung eingesetzt wird.
Es werden unterschieden:

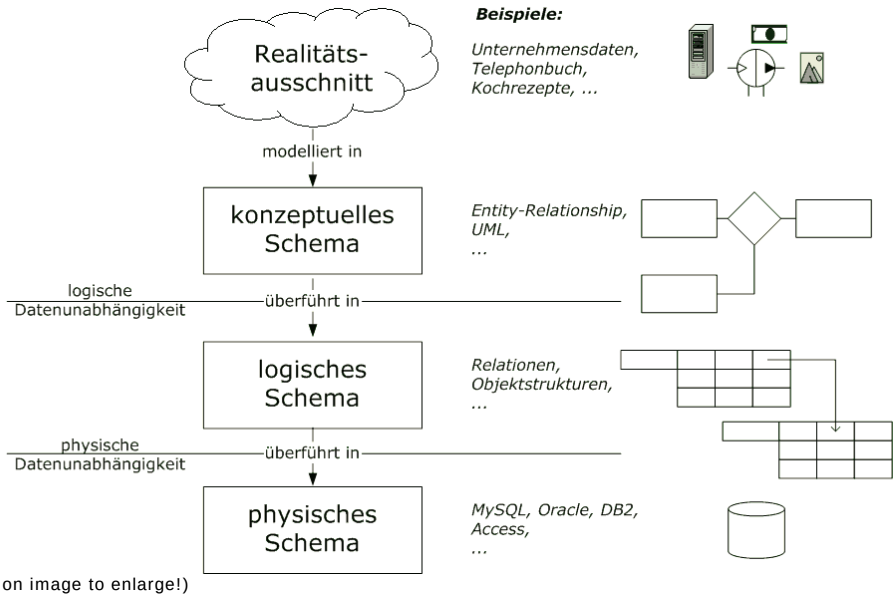
- Data Definition Language (DDL).
  Zur Erzeugung eines Datenmodells.
- Data Manipulation Language (DML).
  Zur Modifikation der verwalteten Daten.
- Data Retrieval Language (DRL).
  Zur Anfrage der in einer Datenbank gespeicherten Daten.
- Data Control Language (DCL).
  Zur Festlegung und Kontrolle von Zugriffsberechtigungen.

Im Verlauf der Vorlesung wird mit *SQL* die bekannteste Sprache im Umfeld relationaler DBMS eingeführt.
Beispiel einer SQL-Anfrage:
```
SELECT FNAME, BDATE FROM EMPLOYEE ORDERED BY BDATE
```

Beispiel 6: Die Datenbanksprache SQL

3-Schema-Architektur

Abbildung 1: *3-Schema-Architektur*



(click on image to enlarge!)

Die Abbildung 1 stellt die 3-Schema-Architektur dar, welche die drei zentralen Modelltypen des Datenbankentwurfsprozesses miteinander in Beziehung setzt.

Definition 10: *Konzeptuelles Schema*
Ein konzeptuelles Schema ist ein Modell, welches den relevanten Realitätsausschnitt (auch *Miniwelt*, *Diskursbereich* oder *Universe of Discourse* genannt) in Struktur und Inhalt beschreibt.

Definition 11: *Logisches Schema*
Ein logisches Schema ist ein Modell, welches paradigmenspezifisch aus einem konzeptuellen Schema abgeleitet wurde.

Die Definition von Relationen als mathematisches Konzept zur Datenstrukturierung stellt ein logisches Schema dar.
Beispielsweise die Festlegung der Struktur der *Person* oder des *Studenten* in Beispiel 3.

Beispiel 7: Relationen sind ein logisches Schema

Definition 12: *Physisches Schema*
Ein physisches Schema ist ein implementierungsspezifisches Modell, welches aus einem logischen Schema abgeleitet wurde.

Definition 13: *Datenunabhängigkeit*
Die Formulierung einer Modellschicht (d.h. eines Datenmodells) ist von den darunter- bzw. darüberliegenden Modellschichten dann datenunabhängig, wenn Änderungen in den „umgebenden“ Modellschichten sich nicht auf die betrachtete Modellschicht auswirken.

Der Vorgang der *Ableitung* zwischen den verschiedenen Modelltypen der 3-Schema-Architektur sollte hierbei idealerweise (aus Gründen der Überprüfbarkeit, Nachvollziehbarkeit, Wiederholbarkeit und Qualitätssicherung) durch einen deterministischen Algorithmus erfolgen.

Die Abbildung 1 zeigt rechts neben den Modelltypen symbolhaft typische graphische Veranschaulichungen der jeweiligen Modellausprägungen.

1.2 Anforderungen an Datenbanksysteme

Allgemein: Speicherung, Verwaltung und Kontrolle der Daten sowie Organisation des u.U. gleichzeitig erfolgenden Zugriffs.
Spezieller:

- Redundanzfreie Datenspeicherung.
  Von dieser Forderung kann bewußt aus Gründen der Geschwindigkeitsoptimierung abgewichen werden.
- Gewährleistung von Integritätsbedingungen und Einhaltung von Regeln.

- Daten-Programm-Unabhängigkeit.

Wünschenswerte Eigenschaften:

- Leistungsfähigkeit
- Skalierbarkeit
- Benutzerfreundlichkeit
- Flexibilität
- ... spezifische Anforderungen, die sich aus der Anwendungssituation ergeben

1.3 Typen von Datenbankmanagementsystemen

Datenbanken werden heute vielfälltig in Wirtschaft, Technik und Wissenschaft eingesetzt.
Für verschiedene Anwendungsgebiete und Strukturen der verwalteten Daten haben sich daher spezifische DBMS-(Unter-)Typen herausgebildet, die diese Anwendungsfelder besonders gut unterstützen:

- **Deduktive Datenbanken**
  Ein um eine Menge von Regeln (Deduktionskomponenten) erweitertes Datenmodell welches logische Schlüsse auf Basis der hinterlegten Fakten ziehen kann.
- **Multimedia Datenbanken**
  Ein System, welches sich besonders zur Verwaltung großer Bild-, Audio- oder Videodaten eignet.
- **Objektdatenbanken**
  Ein System zur Speicherung von Strukturen gemäß dem logischen Objektmodell.
- **Geographische Datenbanken**
  Ein System das sich besonders zur Verwaltung geographischer Daten (z.B. Landkarten) eignet.
- **XML-Datenbanken**
  Ein System zur Speicherung gemäß dem logischen Modell des XML Information Sets.
- **Aktive Datenbanken**
  Ein System zur selbständigen Reaktion auf externe Ereignisse.
- **Temporale Datenbanken**
  Ein System, welches neben den reinen Datenbeständen auch die Zeit des Datenzustandes mitverwaltet.

Exkurs

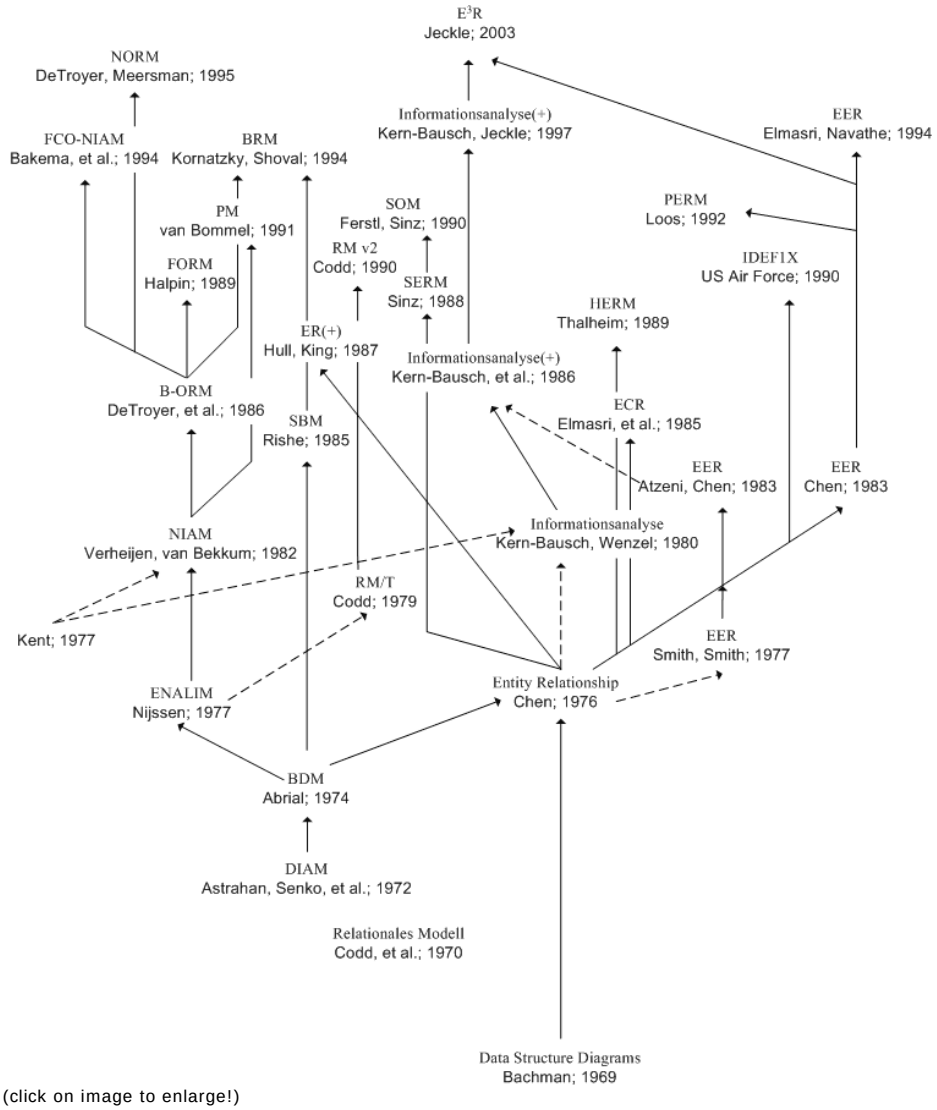
Erste Gehversuche mit dem RDBMS MySQL

2 Entwurf einer Datenbank

2.1 Graphischer Entwurf des konzeptuellen Schemas mit dem Entity-Relationship Modell

Seit der wirkungsmächtigen Erstveröffentlichung des *Entity Relationship Modells* (ERM) durch P. Chen 1976 kommt dieser Modellierungssprache zur Erstellung des konzeptuellen Schemas die uneingeschänkt größte Bedeutung in der Praxis zu.
In der Folgezeit wurden verschiedene Weiterentwicklungen des ursprünglichen ERM vorgeschlagen, die das Originalmodell in verschiedenen Richtungen erweitern. Hierunter fallen die Einführung von Konstrukten zur Abbildung hierarchischer Beziehungen ebenso wie Primitive zur Darstellung von Aggregationsbeziehungen.
Die Graphik der Abbildung 2 zeigt eine Auswahl verschiedener Entwicklungen rund um das initiale ERM sowie einige zentrale Weiterentwicklungen. Innerhalb der Abbildung ist unterhalb des Namens der Modellierungssprache (sofern vorhanden, bei Weiterentwicklungen ohne eigenständige Namensgebung ist zur Unterscheidung vom Vorgängermodell ein geklammertes Pluszeichen angetragen) der Autor sowie das Jahr der Erstveröffentlichung dargestellt.

Abbildung 2: *Entwicklungslinien des ER-Modells*



(click on image to enlarge!)

Im oberen Bereich der Abbildung ist das *Semantically Enriched Extended Entity Relationship Model* (E^3R) dargestellt, welches im Rahmen dieser Vorlesung behandelt wird.
Es stellt eine kompatible Entwicklung dar, die versucht die datenorientierten Aspekte der ER-Nachfolgemodelle mit denen der semantischen Datenmodellierung zu vereinigen.

Die Grundkonzepte des E^3R-Modells sind:

Definition 14: *Entität*
Eine Entität ist ein eindeutig identifizierbares und daher wohlunterscheidbares „Ding".
Im graphischen E^3R-Modell werden Entitäten durch benannte Rechecke dargestellt, die im Zentrum den unterstrichenen Namen der Entität, gefolgt vom -- in Klammern angegebenen -- Namen des Entitätstypen, tragen.

Anmerkung: Der Begriff *Ding* wird hierbei in seiner Bedeutung als Synonym von *Seiendes, Gegenstand* oder *Objekt* gebraucht. Die philosophische Terminologie detailliert den Begriff zusätzlich hinsichtlich seiner Verwendung zur Beschreibung raumzeitlicher Gegenstände mit festgelegten charakteristischen (substantiellen) und zufällig anhaftenden Eigenschaften (Akzidenzien) aus.

- Die Tafel direkt vor ihnen.
- Sie selbst.
- Dieser Hörsaal.

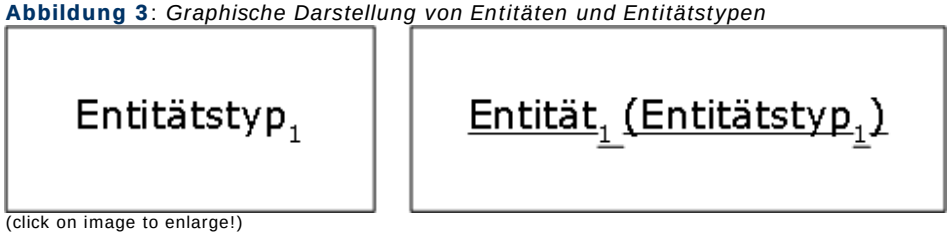
Beispiel 8: Beispiele für Entitäten

Definition 15: *Entitätstyp*
Ein Entitätstyp ist eine ungeordnete und duplikatfreie Sammlung von als logische zusammengehörig betrachteten Entitäten.
Im graphischen E^3R-Modell werden Entitätstypen durch benannte Rechtecke dargestellt. Der Name eines Entitätstyps muß dabei schemaweit eineindeutig sein.

- Tafel.
- Person.
- Student.

Beispiel 9: Beispiele für Entitätstypen

Abbildung 3: *Graphische Darstellung von Entitäten und Entitätstypen*



(click on image to enlarge!)

Soll ein Hinweis auf eine spätere physische Realisierung (d.h. die gewählte Form der Abspeicherung von Entitäten in der Datenbank) gegeben werden, so kann einem Entitätstypen ein Repräsentationstyp zugeordnet werden, bzw. einer Entität eine Repräsentation.

Definition 16: *Repräsentationstyp*
Ein Repräsentationstyp führt einen physischen Typ in das konzeptuelle Schema ein, der zur technischen Implementierung eines durch ihn annotierten Entitätstypen herangezogen werden kann.
Als Repräsentationstypen sind beliebige atomare (d.h. in ihrer Semantik nicht weiter verlustfrei zerlegbare) Datentypen eines logischen oder physischen Modells zugelassen.

- Integer
- Datum
- Money
- String

Beispiel 10: Beispiele für Repräsentationstypen

Definition 17: *Repräsentation*
Eine Repräsentation ist eine Ausprägung genau eines Repräsentationstypen.

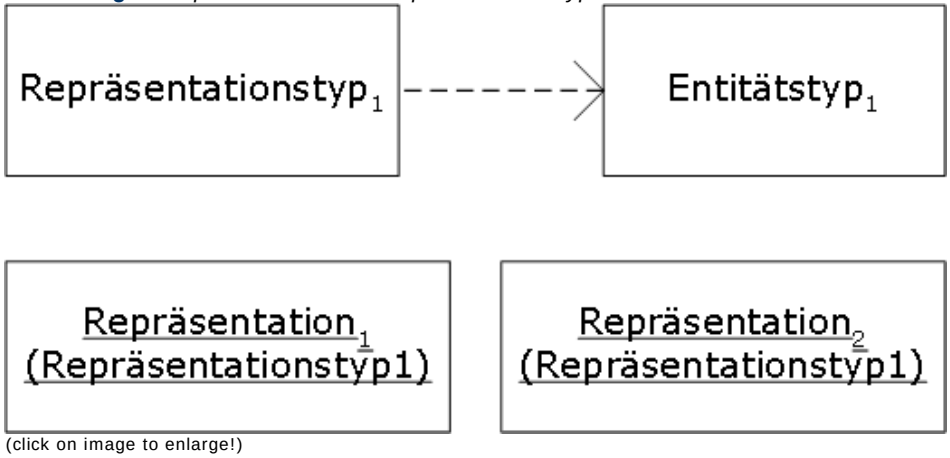
Der zugehörige Repräsentationstyp ist in Klammern angegeben.

- 42 (Integer)
- 2004-05-24 (Datum)
- €99,95 (Money)
- "Hallo Welt!" (String)

Beispiel 11: Beispiele für Repräsentationen

Die graphische Darstellung erfolgt durch benannte Rechtecke. Repräsentationen werden unterstrichen mit der geklammerten nachfolgenden Angabe des Repräsentationstypen dargestellt. Repräsentationstypen werden durch eine gerichtete Kante mit unterbrochener Linienführung mit dem durch sie repräsentierten Entitätstypen verbunden.

Abbildung 4: *Repräsentation und Repräsentationstyp*

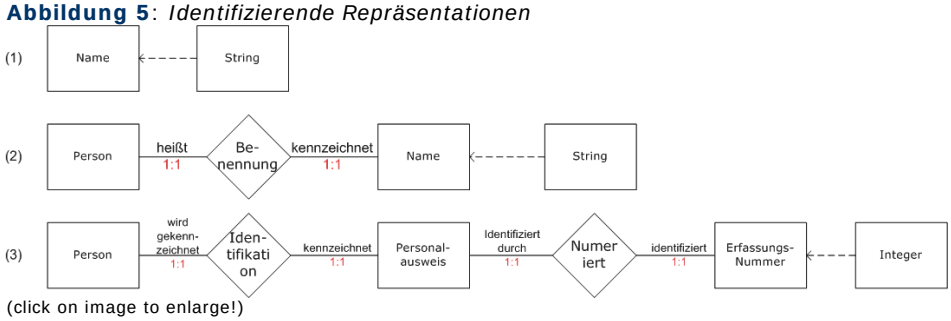


(click on image to enlarge!)

Das Beispiel der Abbildung 5 zeigt verschiedene Beispiele für die Verknüpfung von Entitätstypen

mit ihren zugehörigen Repräsentationstypen.
Im Teilbeispiel (1) wird der Entitätstype `Name` unmittelbar durch den Repräsentationstypen `String` repräsentiert, d.h. die spätere physische Realisierung des Entitätstypen `Name` wird durch den Datentyp `String` erfolgen.
Teilbeispiel (2) zeigt eine transitive Repräsentation (genaugenommen eine transitive Repräsentation erster Ordnung). Hier ist der Entitätstyp `Person`, welcher selbst über keinen Repräsentationstypen verfügt, in eindeutiger Weise (d.h. über einen Assoziationstyp der ausschließlich über Kardinalitsintervalle von `1:1` verfügt (nach Maßgabe der Anmerkung zur Struktur der Kardinalitätsintervalle kann es sich daher nur um einen binären Assoziationstypen handeln)) mit dem Entitätstypen `Name` verknüpft, der über die Repräsentation `String` verfügt.
Abschließend zeigt das Teilbeispiel (3) die transitive eindeutige Assoziierung des Entitätstypen `Person` mit dem Entitätstypen `Personalausweis` durch den Assoziationstypen `Identifikation`, wobei `Personalausweis` seinerseits in eindeutiger Weise mit der durch `Integer` repräsentierten Erfassungsnummer assoziiert ist.

Abbildung 5: *Identifizierende Repräsentationen*



(click on image to enlarge!)

Definition 18: *Assoziation*
Eine Assoziation ist eine benannte n-äre Beziehung (n>1) zwischen Entitäten.
Die Semantik jeder durch eine Assoziation verbundenen Entität wird durch Angabe einer innerhalb einer Assoziation für jede verbundene Entität eindeutigen Rolle konkretisiert.
Im graphischen E^3R-Modell wird eine Assoziation durch eine Raute dargestellt, die durch ungerichtete Kanten mit Entitäten verbunden ist. Im Zentrum wird der Name der Assoziation, gefolgt vom in Klammern geschriebenen Namen des Assoziationstypen plaziert. Zusätzlich sind die beiden Namen unterstrichen dargestellt.

Definition 19: *Assoziationstyp*
Ein Assoziationstyp ist eine duplikatfreie ungeordnete Sammlung von logisch als zusammengehörig betrachteten Assoziationen.
Jede zu einem Assoziationstypen beitragende Rolle wird durch ein Kardinalitätsintervall ergänzt.
Im graphischen E^3R-Modell wird ein Assoziationstyp durch eine Raute dargestellt, die durch ungerichtete Kanten mit Entitätstypen verbunden ist. Im Zentrum des Assoziationstypen wird sein schemaweit eineindeutiger Name plaziert.

- Arbeitsverhältnis.
- Ehe.
- Verwandschaft.

Beispiel 12: Beispiele für Assoziationstypen

Definition 20: *Kardinalitätsintervall*
Ein Kardinalitätsintervall legt die Anzahl derjenigen Entitäten fest, die mit einer die Rolle einnehmenden Entität zu einem Zeitpunkt innerhalb einer Assoziation verbunden sein können.
Das Intervall wird in der Schreibweise „i:j“ angegeben, wobei *i* eine beliebige natürliche Zahl oder die Null ist und *j* eine beliebige natürliche Zahl oder das Symbol *n* ist. Zusätzlich gilt: i<=j.

- 0:1.
- 3:7.
- 0:n.
- 1:n.
- 99:n.

Beispiel 13: Beispiele für Kardinalitätsintervalle

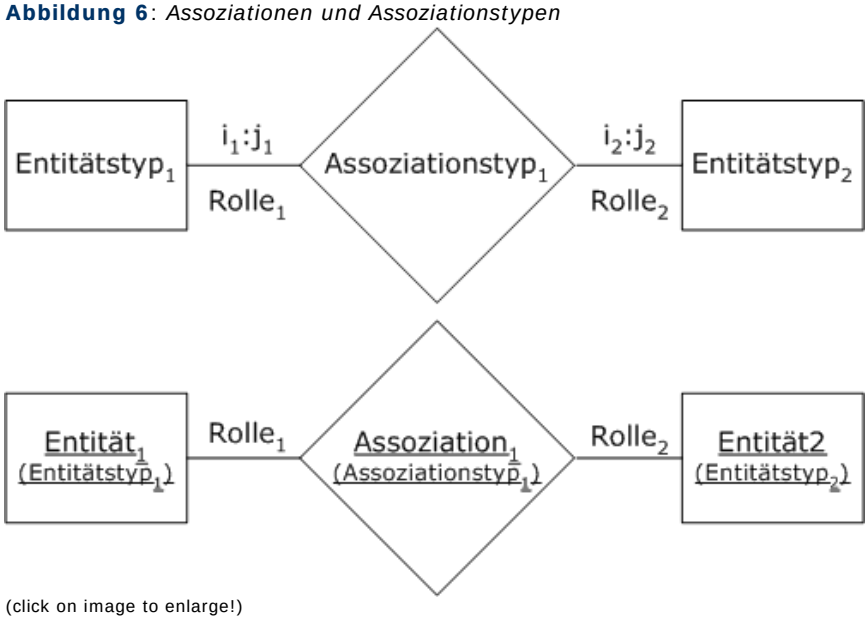
Ungültig hingegen sind:

- 7:0 (Obergrenze kleiner als Untergrenze).
- -5:7 (-5 ist keine natürliche Zahl.)

- n:8 (*n* ist nicht als Untergrenze erlaubt.)

Allgemein gilt: Für *n*-äre Assoziationstypen gilt die Einschränkung, daß die Maximalkardinalität die ein Entitästyp zu einem *n*-ären Assoziationstypen beitragen darf größer gleich *n*-1 ist.

Abbildung 6: *Assoziationen und Assoziationstypen*



(click on image to enlarge!)

Zentrales Konzept des E^{3}R-Modells ist die Idee der Rolle, welche als hauptinformationstragendes Konstrukt fungiert:

Definition 21: *Rolle*
Eine Rolle die durch einen Entitätstypen innerhalb eines Assoziationstypen eingenommen wird charakterisiert die konkrete Verwendung von Entitäten des gegebenen Typs im Kontext der Assoziationen die zum betrachteten Assoziationstyp zusammengefaßt werden.

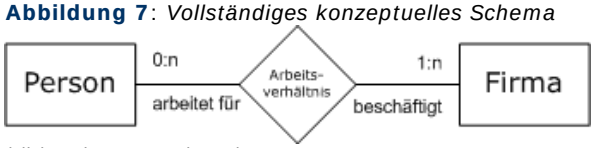
Anmerkung: Für den relationalen Datenbankentwurf ist es notwendig, daß jeder im konzeptuellen Schema modellierte Entitätstyp entweder über einen Repräsentationstyp verfügt oder über eine *Namenskonvention*, d.h. eine binäre Assoziationstyp deren Kardinalitätsintervalle ausschließlich auf 1:1 festgelegt sind, die den Entitätstyp direkt oder transitiv mit einem mit Repräsentation versehenen Entitätstypen verbindet.

Gleichzeitig wird durch die Rolle der Brückenschlag zwischen natürlicher Sprache und formaler graphischer Darstellung im E^{3}R-Modell ermöglicht.
So lassen sich die Sätze

- Jede Person arbeitet optional für mehrere Firmen.
- Jede Firma beschäftigt ein oder mehrere Personen.

in das nachfolgende konzeptuelle Schema überführen:

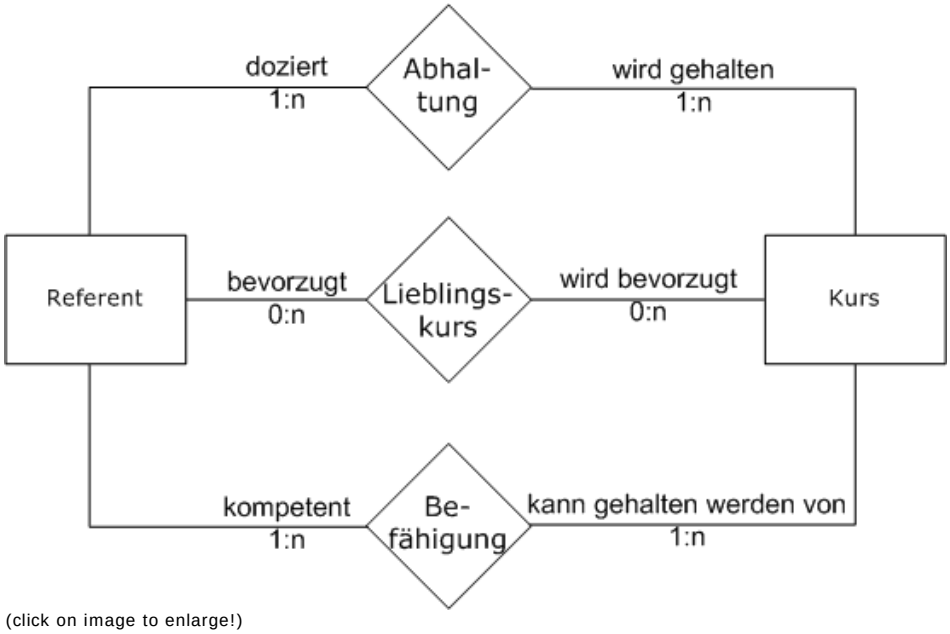
Abbildung 7: *Vollständiges konzeptuelles Schema*



(click on image to enlarge!)

Zusätzlich zeigt das konzeptuelle Schema der Abbildung 8 die Mächtigkeit des Rollenkonzepts zur Darstellung verschiedener Informationszusammenhänge.
So enthält das abgebildete konzeptuelle Schema die drei verschiedenen Assoziationstypen `Abhaltung`, `Lieblingskurs` und `Befähigung` welche ausschließlich Rollen enthalten die durch die beiden dargestellten Entitätstypen `Referent` und `Kurs` gespielt werden.

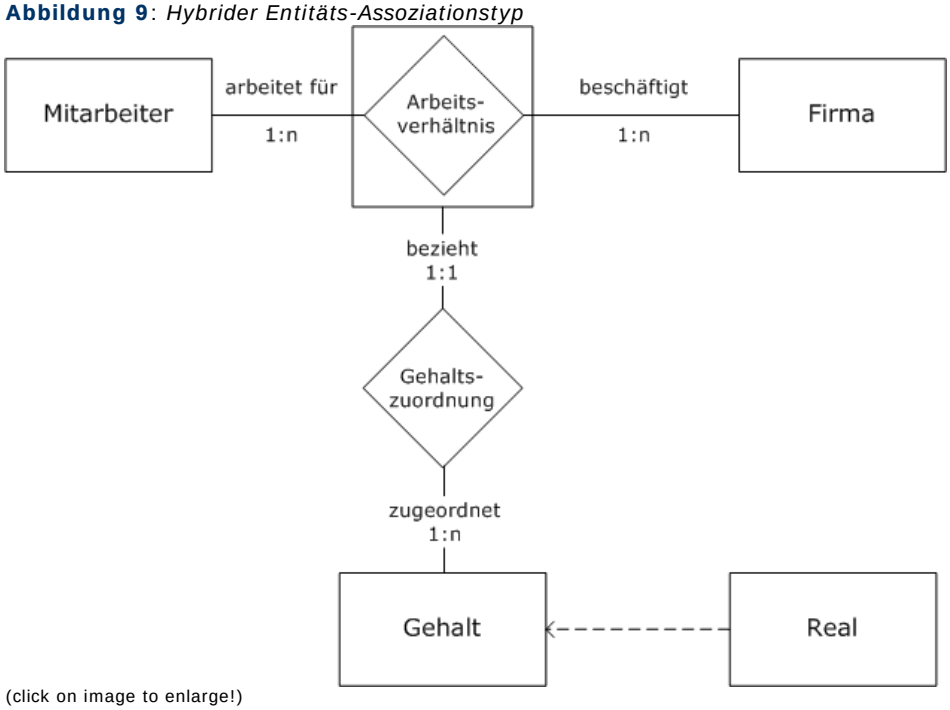
Abbildung 8: *Verschiedene Rollen*



(click on image to enlarge!)

Definition 22: *Hybrider Entitäts-Assoziationstyp*
Ein hybrider Entitäts-Assoziationstyp vereinigt das Sprachelement des Entitätstyps und des Assoziationstyps in sich und bewahrt die Semantik beider Konstrukte.

Abbildung 9: *Hybrider Entitäts-Assoziationstyp*

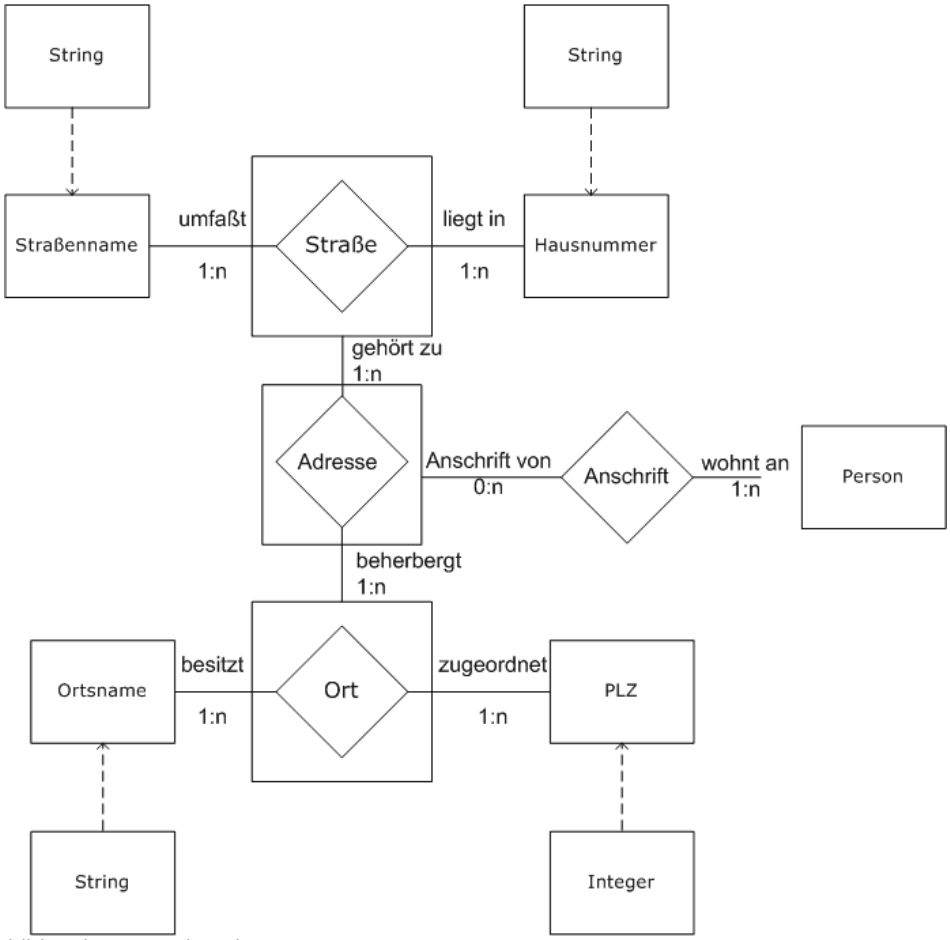


(click on image to enlarge!)

Abbildung Abbildung 10 stellt die Informationsstruktur einer Adresse dar.
Dabei zeigt das konzeptuelle Schema die Verwendung der hybriden Entitäts-Assoziationstypen.
So können jedem `Straßennamen` beliebig viele `Hausnummern` zugeordnet werden und umgekehrt.
Jeweils zwei dieser Angaben zusammen bilden die `Straße`.
Jedem `Ortsnamen` kann über mehrere `Postleitzahlen` verfügen, ebenso kann dieselbe Postleitzahl mehreren gleich benannten Orten zugeordnet werden (Beispiel: Ortsteile).
`Postleitzahl` und `Ortsname` zusammen bilden den `Ort`.
Aus der Kombination von `Straße` und `Ort` wird eine `Adresse` gebildet. Dabei kann jede `Straße` (=Kombination aus `Straßenname` und `Hausnummer`) mehreren Orten (=Kombination aus `Ortsname` und `Postleitzahl`) und umgekehrt zugeordnet sein.

Das Beispiel unterstreicht die alleinige Bildbarkeit hybrider Entitäts-Assoziationstypen beim vorliegen von Kardinalitätsintervallen, die alle über ein Maximum größer 1 verfügen.
Vgl. hierzu Aussagen der Anmerkung zur Bildung von Kardinalitätsintervallen

Abbildung 10: *Informationsstruktur Adresse*

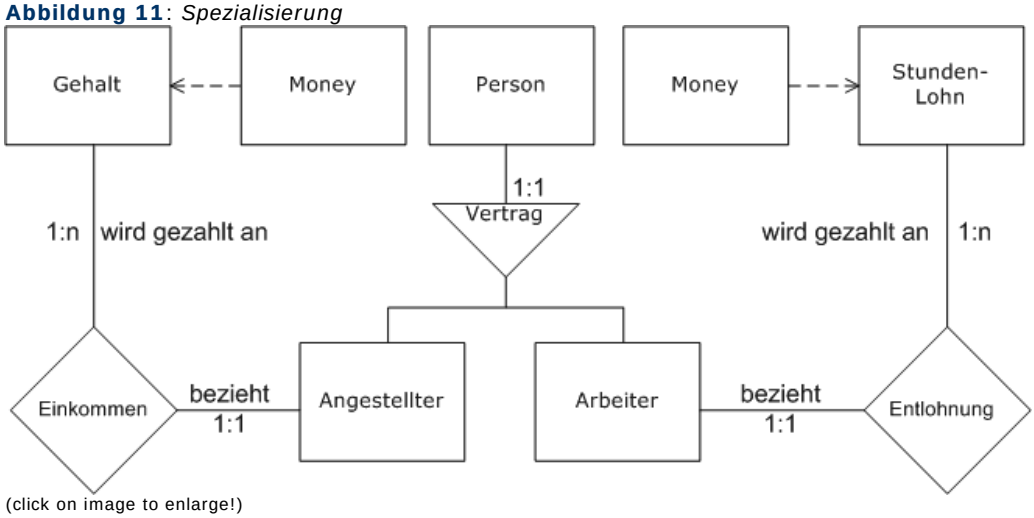


(click on image to enlarge!)

Weiterführende Konzepte

Definition 23: *Spezialisierungsassoziationstyp*
Ein Spezialisierungsassoziationstyp ist eine duplatfreie ungeordnete Sammlung von logisch als zusammengehörig betrachteten Assoziationen.Zu den in der Menge enthaltenen Assoziationen tragen der zu spezialisierende Entitätstyp (der sog. *Super-* oder *Obertyp*) und der spezialisierende (entsprechend als sog. *Sub-* oder *Untertyp* bezeichnet) Rollen bei.
Die Rolle des Supertyps ist hierbei auf *wird spezialisiert zu* fixiert, als Kardinalitätsintervalle sind ausschließlich `0:1`, `0:n`, `1:1` und `1:n` zulässig.
Die Rolle des Subtyps ist auf *ist Spezialisierung von* mit dem Kardinalitätsintervall `1:n`fixiert.
Jeder Spezialisierungsassoziationstyp wird durch ein Distinktionsmerkmal charakterisiert, das expliziert hinsichtlich welchen Merkmals die Spezialisierung gebildet wird.
Die Verknüpfung durch einen Spezialisierungsassoziationstyp bewirkt, daß alle Assoziations- und Repräsentationstypen, die für den Supertyp definiert sind auch automatisch für alle Subtpyen definiert werden.

Abbildung 11: *Spezialisierung*



(click on image to enlarge!)

Die Abbildung 11 zeigt die Spezialisierung des Entitätstypen `Person` hinsichtlich des Distinktionsmerkmals `Vertrag`. Dabei gilt: Jede Person kann nur genau einmal (`1:1`) hinsichtlich

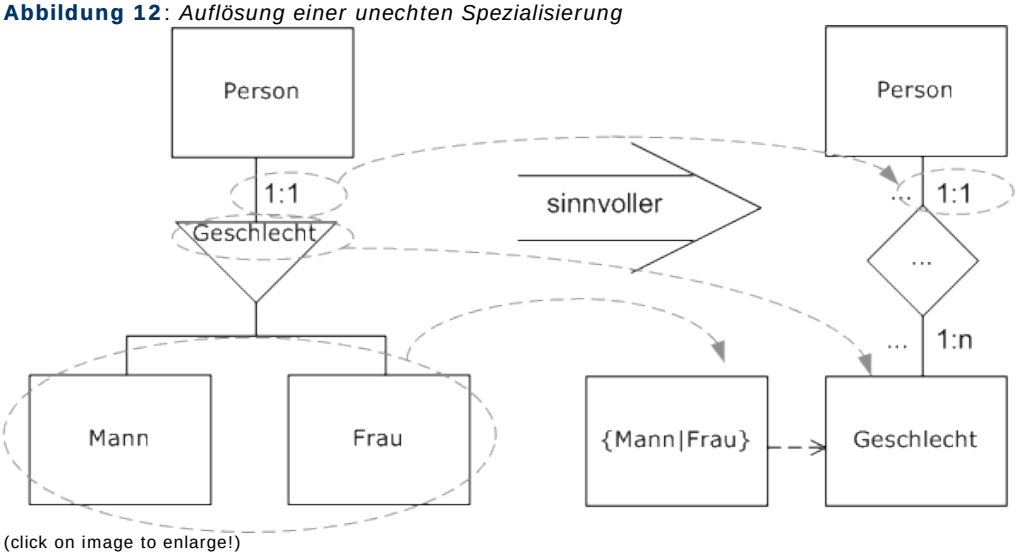
ihres (Arbeits-)Vertrages zu `Angestellter` oder `Arbeiter` spezialisiert werden.

Abbildung 11 zeigt ferner, daß die spezialisierten Entitätstypen `Angestellter` und `Arbeiter` über zusätzliche, d.h. für den Obertyp `Person` nicht definierte, Eigenschaften (`Gehalt` bzw. `Stundenlohn`) verfügen.

Zur Pragmatik des Spezialisierungsassoziationstyps:
Spezialisierungsassoziationstypen sollten ausschließlich dann eingesetzt werden, wenn es sich um „echte“ Spezialisierungen handelt. Eine *echte Spezialisierung* liegt immer dann vor, wenn jeder spezialisierte Entitätstyp über Assoziationstypen verfügt, die der allgemeinere Obertyp nicht besitzt.

Gilt dieses Kriterium nicht, d.h. können für die spezialisierten Entitätstypen keine zusätzlichen Eigenschaften gebildet werden, dann solte die Spezialisierung *nicht vorgenommen* werden.
In diesem Falle bietet sich die Überführung der unnötigen Spezialisierung eine Eigenschaft des (vermeintlichen) Obertypen an. Zusätzlich sollte ein Entitätstyp zur Aufnahme derjenigen Information gebildet werden, für welche die Darstellung als Spezialisierungsassoziationstyp intendiert war.
Dieser neue Entitätstyp kann geeignet mit einem Repräsentationstyp versehen werden, um die unterscheidende (distingierende) Information darzustellen.
Die einzelnen Schritte sind in Abbildung 12 beispielhaft zusammengestellt.

Abbildung 12: *Auflösung einer unechten Spezialisierung*



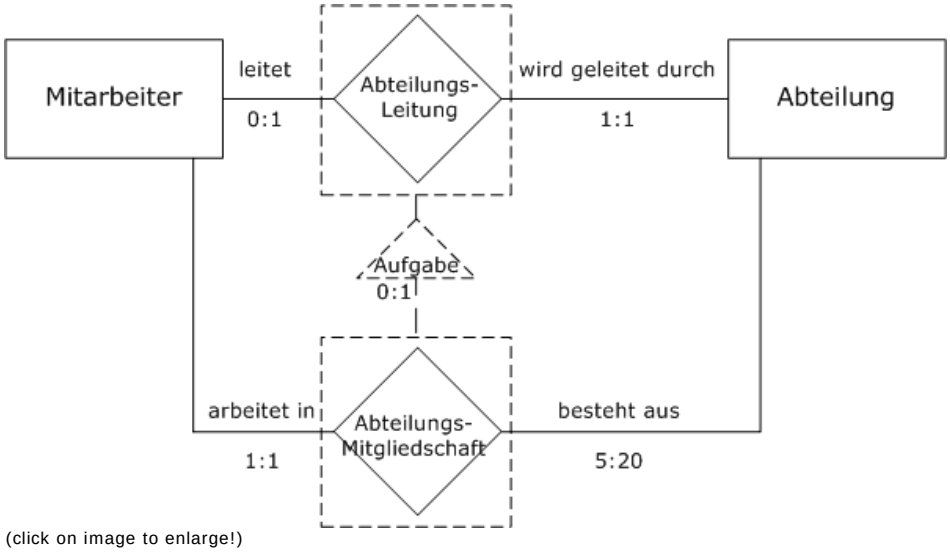
(click on image to enlarge!)

Definition 24: *Metainformation*
Informationsanteile eines Modells, die nicht direkt in das logische Schema übernommen werden, sondern in konsistenzgarantierende Regeln oder Applikationscode abgebildet werden.

Die Abbildung 13 veranschaulicht die Nutzung des Spezialisierungsassoziationstyps zur Formulierung von Metainformation. Im Beispiel wird gefordert, daß jeder `Abteilungsleiter` auch gleichzeitig als Mitarbeiter der durch ihn geleiteten Abteilung erfaßt sein muß.

Hinweis: Metainformation muß nicht zwingend semantisch irreduzibel erfaßt werden, wie die --- eigentlich illegale Bildung der beiden hybriden Entitäts-Assoziationstypen `Abteilungsleitung` und `Abteilungsmitgliedschaft` zeigt.

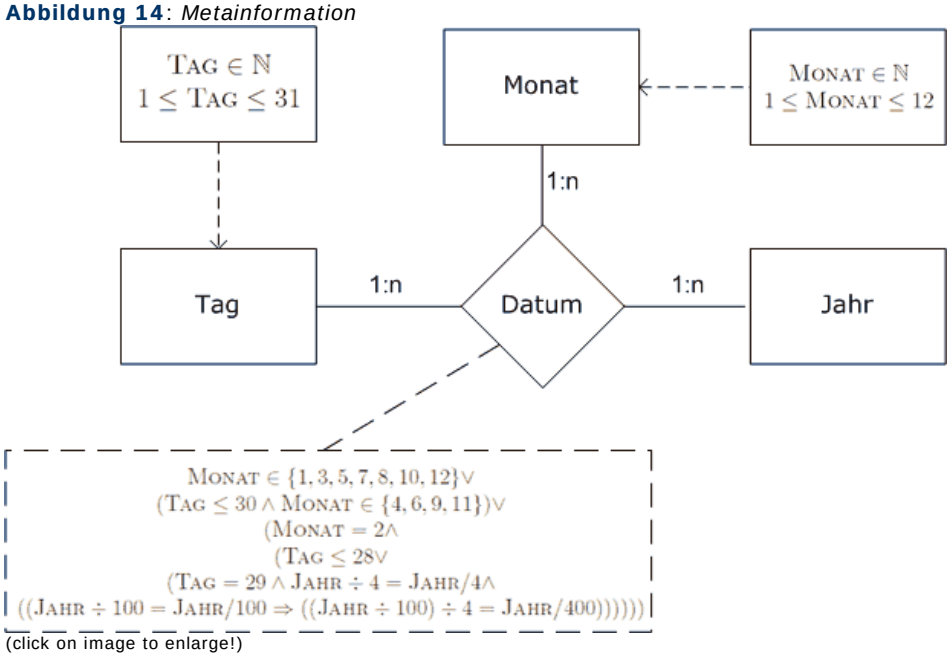
Abbildung 13: *Metainformation*



(click on image to enlarge!)

Das konzeptuelle Schema der Abbildung 14 zeigt ein Beispiel für konsistenzgarantierende Metainformation, die nicht durch E^{3}R-Syntax ausdrückbar ist und daher in textueller Form annotiert wird.

Abbildung 14: *Metainformation*



(click on image to enlarge!)

Phasenmodell der Erstellung eines konzeptuellen Schemas mit E^{3}R

E^{3}R ist eine Notation und Methode zur Entwicklung des konzeptuellen Schemas für jede beliebige Art von Kommunikationssituationen.

Phase 0 ist die Vorbereitungsphase, die von der Idee, ein E^3-Schema für einen Realitätsausschnitt zu erstellen, über die Auswahl der Beteiligten bis zu ihrer Ausbildung in den Techniken zur Darstellung von Information und in der Vorgehensweise der Analyse reicht.

Es folgt die Festlegung des Informationsbereichs (**Phase 1**). Hier werden die für den gewünschten Anwendungsbereich relevanten Entitäten und Assoziationen gesammelt und zu Entitäts- und Assoziationstypen zusammengefaßt. Eine große Hilfe dabei sind verbale Beschreibungen der in der Datenbank zu verwaltenden Information, kommentierte Listen mit Daten des betrachteten Informationsbereichs oder ähnliches. Das Ergebnis ist eine erste, grobe Struktur der relevanten Information.

Diese Struktur wird in **Phase 2** immer weiter verfeinert, wobei man für jeden Entitätstyp entweder direkt oder transitiv eine Repräsentation definiert.
Dann werden alle relevanten Eigenschaften, die eine Entität eines Typs haben kann, in Form von semantisch irreduzibel formulierten Assoziationstypen beschrieben. Dabei treten erfahrungsgemäß neue, zuvor nicht berücksichtigte Entitätstypen auf.

Deshalb wird die Phase 2 solange inkrementell iteriert, bis keine neuen Entitätstypen mehr

identifiziert werden zu denen noch Repräsentation zu definieren oder durch Assoziationstypen anzubinden sind.

Bis zu diesem Punkt standen strukturelle, formale Gesichtspunkte im Vordergrund. In **Phase 3** treten diese zurück; nun stehen semantische Gesetzmäßigkeiten im Vordergrund, soweit diese nicht bereits in den Phasen 1 und 2 erkannt und behandelt worden sind.

Ziel der Phase 3 ist es, die bis dato erstellte Informationsbeschreibung geeignet zu ergänzen um auch alle nicht durch die E^3R-Notation darstellbaren Konsistenzregeln zu erfassen.
Zusätzlich kann die E^3R-Notation zur Formulierung von Metainformation auf einer höheren Modellebene angewendet werden.

Hinweis: Es kann beim Erstellen des konzeptuellen Schemas durchaus vorkommen, daß sich das Ergebnis der vorausgegangenen Phase als unvollständig herausstellt. In diesem Fall ist es unbedingt notwendig, in diese Phase zurückzukehren und dann mit dem korrigierten Ergebnis dieser Phase weiterzuarbeiten. Dies ist kein Wegwerfen der bisher geleisteten Arbeit, denn meist genügen einige wenige Streichungen und Ergänzungen.
Bleibt diese Regel unberücksichtigt, so nimmt begibt man sich der Möglichkeit wichtige Eigenschaften der Information im konzeptuellen Schema eindeutig festzuhalten. Dabei spricht das Verhältnis zwischen der gewonnen Exaktheit und dem zusätzlichen Aufwand sehr zugunsten der exakten und sauberen Lösung.

Resultat der korrekten Anwendung des Phasenmodelles ist ein *vollständiges konzeptuelles Schema* als Voraussetzung der Umsetzbarkeit in beliebige logische Strukturen.

Definition 25: *Vollständiges konzeptuelles Schema*
Ein vollständiges konzeptuelles Schema ist ein E^3R-Schema in dem alle Entitäts- und Assoziationstypen, sowie alle Rollen benannt sind. Darüberhinaus ist jede Rolle mit einem korrekten Kardinalitätsintervall versehen, sowie jedem Entitätstypen direkt oder transitiv ein Repräsentationstyp zugeordnet.
Sofern Metainformation existiert, ist diese auch in adäquater Weise dargestellt.

Fallstudie: Fächerdatenbank

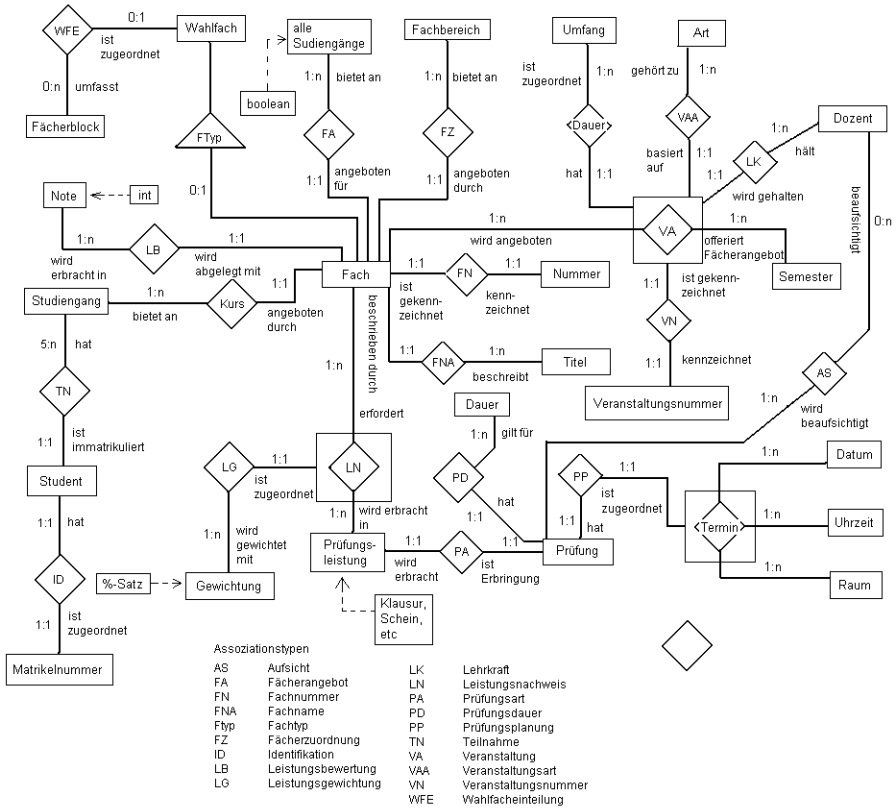
Der Fachbereich möchte die Belegung der Fächer in einer Datenbank abspeichern; hierfür gelten folgende semantische Regeln:

Die vorgesehenen Fächer haben eine feste Nummer, die sich niemals ändert sowie eine einen längeren Titel. Zusätzlich sind sie von einem Fachbereich entweder für einen speziellen Studiengang (z.B. *WIB*) oder allen Studiengängen angeboten, dann gilt die Zuordnung *FH*. Gleichzeitig kann jedes Wahlfach einem Fächerblock (z.B. *Consulting* oder *Informatik*) zugeordnet sein.

Die in einem bestimmten Semester angebotenen Fächer erhalten eine Veranstaltungsnummer, die nur für dieses Semester gilt. Dazu wird der jeweilige Dozent angegeben und die Art der Lehrveranstaltung (*Vorlesung*, *Seminar*, *Praktikum* etc.) sowie ihr Umfang in Semesterwochenstunden.
Die Notenbildung in jedem Fach kann durch eine oder mehrere Prüfungsleistungen erfolgen (z.B. *Leistungsnachweis*, *Schein*, *Prüfung*, etc.), die in unterschiedlichen Prozentsätzen gewichtet werden. Jede Prüfung findet zu einem festgelegten Datum in einem Raum zu einer Uhrzeit statt und wird durch mindestens einen Dozenten beaufsichtigt. Zusätzlich soll die Dauer der Prüfungsleistung vermerkt werden.

Ein Student ist durch eine eindeutige Matrikelnummer gekennzeichnet. Zusätzlich wird sein Studiengang gespeichert.

Abbildung 15: *Konzeptuelles Schema der Fallstudie*



(click on image to enlarge!)

2.2 Ableitung logischer Relationenstrukturen

Erweiterung der Grundbegriffe des Relationenmodells

Definition 26: *Superschlüssel*
Ein Superschlüssel *SK* ist eine nicht-leere Teilmenge von Attributen einer Relation für die gilt, daß zwei verschiedene Tupel t_1 und t_2 dieser Relation keine gleiche Wertbelegung aufweisen.

Der Superschlüssel definiert damit eine Eindeutigkeitkeitseinschränkung, nach der zwei Tupel allein über die Betrachtung der im Superschlüssel zusammengefaßten Attribute unterscheidbar sind.

Gegeben sei die Relation *Mitarbeiter*:

Vorname	Nachname	Geburtsdatum	Persausweisnummer
Xaver	Obermüller	1970-03-04	134975459
Rosi	Hinterhuber	1973-06-02	781367519
Rosi	Obermüller	1963-11-03	783148384
Hans	Hinterhuber	1970-03-04	977554422

Mögliche Superschlüssel dieser Relation sind:

- (Vorname, Nachname, Personalausweisnummer)
- (Nachname, Geburtsdatum, Personalausweisnummer)
- (Geburtsdatum, Personalausweisnummer)
- ...

Beispiel 14: Beispiele für Superschlüssel

Definition 27: *Schüssel*
Ein Schlüssel *K* ist ein Superschlüssel, der sofern man ein Attribut aus ihm entfernt nicht mehr eindeutigkeitseinschränkend wirkt.

Der einzige Schlüssel der Relation Mitarbeiter ist `Personalausweisnummer`.

Beispiel 15: Beispiele für Schlüssel

Anmerkungen:

- Die Menge aller Attribute einer Relation ist immer Superschlüssel.
- Jeder Schlüssel ist auch ein Superschlüssel.
  Der Umkehrschluß gilt nicht, da Schlüssel eine schärfere Forderung darstellt.
- Jeder Schlüssel ist zwingend eine minimal identifizierende Attributkombination.

Häufig tritt es in der Praxis auf, daß sich in einer Relation mehr als ein Schlüssel finden läßt. Jeder dieser möglichen gleichwertigen Schlüssel wird daher als *Schlüsselkandidat* bezeichnet.

Gegeben sei die Relation *Lagerverwaltung*:

Lagerplatz	Produktnummer	Produktname	Menge
7952	7946	Wusch Superfein	3
7412	9854	Blitzbank Extra	5
7894	6542	Maiengrün natur	7
9461	8954	Gelber Gigant	5

Die Relation enthält folgende Schlüsselkandidaten.

- `Lagerplatz`
- `Produktnummer`
- `Produktname`

Beispiel 16: Beispiele für Superschlüssel

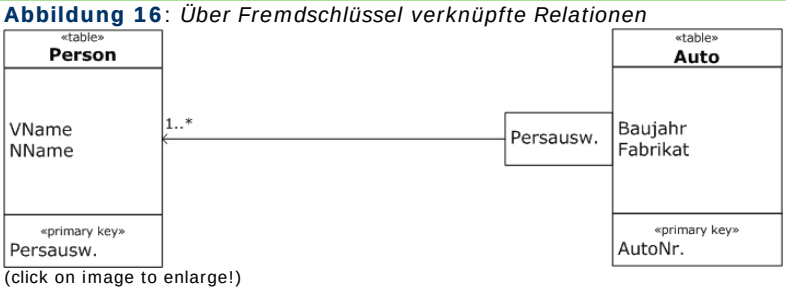
Definition 28: *Primärschüssel*
Ein Primärschlüssel *P* ist ein Schlüssel, der als identifizierendes Merkmal ausgewählt wurde.

In der graphischen Darstellung der Demo-DB sind die Primärschlüssel durch Unterstreichung der beitragenden Attribute hervorgehoben.

Zur Wahrung der Konsistenz innerhalb einer relationalen Datenbank wird üblicherweise u.a. das Mittel der *referentiellen Integrität* eingesetzt, um gleicher Wertinhalte in Attributen (derselben oder verschiedener Relationen) aufeinander abzustimmen.

Definition 29: *Referentielle Integrität*
Attributwerte einer durch referentielle Integrität verknüpften Relation müssen auch in der verknüpften Relation existieren.

Abbildung 16: *Über Fremdschlüssel verknüpfte Relationen*



(click on image to enlarge!)

Das Attribut `Persausw.` der Relation `Auto` verweist auf den Primärschlüssel gleichen Namens der Relation `Person`.
Als Konsequenz dürfen für `Persausw.` in `Auto` nur Werte definiert werden, die sich bereits im Attribut `Persausw.` von `Person` finden.

Beispiel 17: Beispiel für referentielle Integrität

Die Prüfung von Primärschlüsselwerten erfordert u.U. eine Reihe zusätzlicher Datenbankzugriffe. Zu ihrer Beschleunigung können zusätzliche Speicherbereiche, sog. *Indexe* angelegt werden.

Definition 30: *Index*
Ein Index ist ein zusätzlicher Speicherbereich der in der Datenbank verwaltet wird um den lesenden Zugriff auf einzelne Tupel zu beschleunigen.

In der Konsequenz der Beschleunigung der lesenden Zugriffe durch zusätzlichen Speicherplatz verringert sich die Geschwindigkeit der schreibenden Zugriffe (Erzeugung, Aktualisierung und Löschung) etwas.

Die Tabelle `tab` besteht aus zwei Attributen `UUID1` und `UUID2`, wobei ersteres duplikatfrei indexiert wird.

Aktion	Dauer ohne Index [sec]	Dauer mit Index [sec]
Einfügen von 10.000.000 Tupeln `INSERT INTO tab VALUES (...)`	1812	2025
Auswahl aller Tupel `SELECT COUNT(*) FROM tab WHERE UUID1<> "X"` (`UUID1` enthält niemals den Wert X daher werden alle Tupel selektiert)	2100	1800
Auswahl genau eines Tupels `SELECT UUID2 FROM tab WHERE UUID1="..."`	0,422	0,028
Aktualisierung genau eines Tupels `UPDATE tab SET UUID2="Z" WHERE UUID1="..."`	0,415	0,033
Aktualisierung keines Tupels, jedoch vollständige Durchsuchung eines Attributs. `UPDATE tab SET UUID2="Z" WHERE UUID1<>"X"`	0,395	0,014
Löschung eines Tupels `DELETE FROM tab WHERE UUID1="..."`	0,431	0,043
Löschung keines Tupels, jedoch vollständige Durchsuchung eines Attributs. `DELETE FROM tab WHERE UUID1="x"`	0,037	0,008

Beispiel 18: Geschwindigkeitsverhalten mit/ohne Index

Die Erstellung des Index nimmt, bei in der Tabelle gehaltenen 10.000.000 Tupeln 363,063 Sekunden in Anspruch.

Definition 31: *NULL-Wert*
Fehlende Attributwerte in einer Relation werden durch den gesonderten Datenbankeintrag `NULL` dargestellt.

Für die Wertbelegung `NULL` stellt das DBMS sicher, daß sie nicht mit der Ziffer 0 oder dem leeren String kollidiert.

Der Algorithmus

Zentrale Zielsetzung der Erstellung des konzeptuellen Schemas ist die Möglichkeit von ihm ausgehend unterschiedliche logische Modelle, die später in die physische Implementierungssicht abgebildet werden, ableiten zu können.
Dieser Abschnitt stellt einen Algorithmus vor, der es erlaubt aus dem mit E^3R formulierten konzeptuellen Schema logische Strukturen gemäß dem Relationenmodell abzuleiten.

Dabei operiert der Algorithmus ausschließlich auf der graphischen Repräsentation des konzeptuellen Schemas und kann daher auch von entsprechend ausgebildeten Fachexperten manuell durchgeführt werden.
Der durch den Algorithmus abgeleitete logische Datenbankentwurf orientiert sich an festgelegten Gütekriterien um einen redundanzfreien und somit anomalienfreinen Entwurf zu gewährleisten.

Schritt 1

Markiere alle Verbindungslinien (d.h. Rollen), an denen das Kardinalitätsintervall 1:1 steht.

Anmerkung: Eine ausschließliche 1:1-Markierung stellt einen logisch korrekten DB-Entwurf sicher.
Verbindungslinien, die zu Spezialisierungsassoziationstypen führen müssen nicht markiert werden.

Das zusätzliche Markieren aller 0:1-Verbindungen führt zu Performanceverbesserungen.
Bei relationalen DBMS, die fehlende Werte (NULL) zulassen führt das Markieren von 0:1-Verbindungen zu einem optimalen relationalen DB-Entwurf. Bei Implementiereungen die auch optionale Schlüsselkandidaten zulassen führt das fortgesetzte Markieren über 0:1 Verbindungen hinweg zu effizienten DB-Strukturen.

Schritt 2

1. Bilde die Zusammenhangskomponenten (bestehend aus Entitäts- und Assoziationsstypen) bezüglich der markierten Verbindungslinien.
2. Übrigbleibende (d.h. noch außerhalb von Zusammenhangskomponenten plaziert Entitäts- und Assoziationstypen bilden jeweils eine eigene Zusammenhangskomponenten, wenn gilt:
   - Falls ausschließlich alle Mitgliedschaftsintervalle an der Verbindung zwischen einem solchen Entitätstyp und irgendeinem Assoziationstyp den minimalen Wert 0 haben, ist aus diesem Entitätstyp eine eigenständige Zusammenhangskomponente zu bilden, sofern der Entitätstyp kein Repräsentationstyp oder die entsprechenden Entitäten nicht schon in einem anderen Assoziationstyp definiert sind (Kardinalitätsintervall 1:z; z>=1).
   - Repräsentationstypen werden nicht berücksichtigt.
   - Assoziationstypen bilden jeweils (als einziger Inhalt) eine eigene Zusammenhangskomponente.

Schritt 3

Treten innerhalb einer Zusammenhangskomponente Zyklen auf, so sind diese wie folgt zu behandeln:

Anmerkung: Ein Zyklus in einer Zusammenhangskomponente ist eine Folge $\{ET_1, AT_{1,2}, ET_2, AT_{2,3}, ..., ET_n, AT_{n,1}\}$ mit den Eigenschaften:

- In allen Assoziationstypen$_{i,k}$ ($1<=i, k<=n$) wird die eine Rolle von ET_i und die andere Rolle von ET_k gespielt.
  Anmerkung:Zu Untertypen führende Kanten werden behandelt wie gewöhnliche Beziehungen zu Assoziationstypen.
- Alle Verbindungslinien einer Folge sind markiert.

Zur Ableitung von Relationen müssen Zyklen aufgelöst werden:

- Falls innerhalb eines Zyklus 0:1-Markierungen vorhanden sind, werden diese gelöscht;
- falls nur 1:1-Markierungen vorhanden sind, wird eine beliebig festzusetzende Verbindungslinie gelöscht.

Schritt 4

Aus jeder Zusammenhangskomponente wird eine Relation nach folgenden Regeln:

1. Namensgebend für eine Relation ist genau einer der innen liegenden Entitätstypen. Enthält eine Zusammenhangskomponente nur einen Assoziationstyp, so bekommt die abgeleitete Relation dessen Namen.
2. Die Relation enthält je ein Attribut für jeden direkt mit einer Repräsentation versehenen Entitätstypen im Inneren der Zusammenhangskomponente. Ein Entitätstyp wird zusammen mit seinen sämtlichen Untertypen als ein Entitätstyp betrachtet, sofern die Untertypen über keine eigene Repräsentation verfügen.
3. Zusätzlich enthält die Relation für jeden Assoziationstyp im Inneren einer Zusammenhangskomponente noch je ein Attribut für jede Rolle die zu einem innenliegenden Assoziationstyp beiträgt, welche ein Entitätstyp spielt, der außerhalb der Zusammenhangskomponente liegt.
   Für einen im Inneren der Zusammenhangskomponente liegenden Assoziationstyp sind alle

Entitätstypen, zu denen eine nicht markierte Verbindungslinie führt, außerhalb.

4. Zusammenhangskomponenten, die ausschließlich aus genau einem Assoziationstyp bestehen werden in eine eigenständige Relation überführt, die für jede zum Assoziationstyp beitragende Rolle ein Attribut enthält.
   Dieses Attribut wird mit der Repräsentation des rollenspielenden Entitätstypen typisiert.
5. Jedes aus einem Entitätstyp im Inneren einer Zusammenhangskomponente abgeleitete Attribut ist Schlüsselkandidat.
   Außerdem sind alle diejenigen Attribute Schlüsselkandidaten, deren entsprechende Kardinalitätsintervalle das Maximum 1 besitzen.
6. In den restlichen Fällen sind alle Attribute zusammen Schlüsselkandidat.
7. Existiert in einer Relation mehr als genau ein Schlüsselkandidat, so ist einer unter diesen als Primärschlüssel auszuzeichnen.
8. Jeder Untertyp erbt alle Rollen, die sein Obertyp innerhalb von Assoziationstypen spielt. Zusätzlich erbt er auch die vorhandene Repräsentation des Obertyps.
9. Relationen, deren Attribute sich aus jeweils gleichen Rollen ableiten, werden durch eine einzige Relation dargestellt.
   Treten in einer Zusammenhangskomponente gleiche Rollen eines Entitätstyps mehrfach auf, so werden sie in einer entsprechenden Relational als genau ein Attribut übernommen.
10. Manuelles Eingreifen:
    bei `0:1`-Markierung ist u.U. eine Entscheidung, orientiert an der modellierten Semantik, zu treffen:
    Folgende Konstellationen können das Rückgängigmachen von Markierungen innerhalb einer Zusammenhangskomponente notwendig werden lassen:
    mehrere Schlüsselkandidaten und:
    - alle Schlüsselkandidaten sind optional
    - nicht alle verpflichtend und die Notwendigkeit vorhanden, einen bestimmten als Primärschlüssel festzulegen.

Schritt 5

1. Leiten sich aus einem Entitätstyp mehrere sich entsprechende Attribute ab, so sind die folgenden Abhängigkeiten (Fremdschlüsselbeziehungen) zu berücksichtigen:
   - Ist eine Attributkombination Schlüsselkandidat, so sind zu den entsprechenden Attributkombinationen, die als Primärschlüssel ausgewählt wurden, Fremdschlüsselbeziehungen vorzusehen.
     Hinweis: Fremdschlüsselbeziehungen bedeuten zusätzliche Zugriffe und sollten daher in der Datenbank entsprechend durch Indexstrukturen unterstützt werden.
   - Beim Auftreten identischer Schlüsselkandidaten sind ebenfalls Fremdschlüsselbeziehungen vorzusehen.
2. Metainformationen, die nicht die DB-Strukturebene betreffen, sondern Ausprägungen einschränken, werden den entsprechenden DB-Strukturelementen zugeordnet (z.B. Domäneneinschränkungen bei Attributen).

Schritt 6

Ist ein Entitätstyp Spezialisierung (d.h. Untertyp) eines anderen, so wird in die Relation die aus der Zusammenhangskomponente gebildet wurde, welche den Untertypen beinhaltet die Primärschlüsselattribute derjenigen Relation übernommen, die den Obertypen beinhaltet.

Anmerkung: Schlüsselkandidaten dieser Relation werden identisch zu den anderen Relationen ermittelt.

Schritt 7

1. Systemunabhängig
   - Alle Attribute bekommen als Datentyp den Entitätstyp, durch den sie repräsentiert werden.
   - Bei `1:1`-Markierung wird für alle Attribute der Relation ein `NOT NULL` vergeben.
   - Ein Primärschlüssel muß stets mit `NOT NULL` vereinbart werden.
   - Bei markierten `0:1`-Beziehungen: Alle aus über `0:1`-Beziehungen angebundenen Entitätstypen entstehenden

Attribute werden auf NULL gesetzt.
- Schlüsselkandidaten und Zugriffspfade werden als Indexe angelegt.
- Soweit für Metainformation formale Umsetzungsmöglichkeiten existieren, werden die entsprechenden konsistenzgarantierenden Einschränkungen formuliert.

2. Systemabhängig
   Die Syntax für die physische Realisierung der Relationen (Tabellen) und der Indexstrukturen sowie der Datentypen und Einschränkungen (soweit unterstützt) müssen dem jeweiligen DBMS angepaßt werden.
   Evtl. durch das DBMS automatisch angelegte Indexstrukturen müssen nicht mehr explizit formuliert werden.

2.3 Algebraischer Entwurf mit der Normalformentheorie

Neben dem graphischen Entwurf logischer DB-Strukturen genießt der algebraische Entwurf auf Basis der sog. *Normalformentheorie* in Theorie und Praxis große Bedeutung.
Historisch gesehen stellt die Betrachtung von relationalen Strukturen mit Hilfe mathematischer Methoden die älteste Disziplin dar und findet sich heute in allen bedeutenden Lehrbüchern.
Dieser Abschnitt führt in die sechs verschiedenen Normalformen hinsichtlich ihrer Definition sowie ihrer Implikationen auf die Struktur des logischen Modells ein und zieht Parallelen zur Vorgehensweise des eingeführten Algorithmus zur Umsetzung konzeptueller Strukturen des E^3R-Modells.

Ebenso wie der Algorithmus zur Transformation eines E^3R-Schemas führt auch die Normalisierung zu einem anomalienfreien relationalen Datenbankentwurf. Voraussetzung der Anomaliefreiheit ist die konsequente Ermittlung und Eliminierung von Redundanz, d.h. keine Information darf in der Datenbank mehrfach vorhanden sein.

Insgesamt verfolgt der Normalisierungsprozeß folgende Ziele:

- Redundanzvermeidung als Basis der Anomaliefreiheit
- Vermeidung unnötiger Abhängigkeiten, die Performanceeinbußen bei Einfüge-, Lösch- und Änderungsoperationen nach sich ziehen
- Senkung der Anzahl beteiligter Tabellen bei der Modifikation der Datenbank
- Erhöhung des Dokumentationsgrades des entstehenden Datenmodells (Ziel: Verständlichkeit)

Die Anomalienfreiheit ist die zentrale Basisforderung und Zielsetzung des Normalisierungsprozesses. Im Detail werden drei Ausprägungen unterschiedlicher Anomalien unterschieden:

- **Einfügeanomalie: Durch das Hinzufügen eines korrekten neuen Tupels werden konsistent vorliegende Daten in einen inkonsistenten Zustand überführt.**
- **Löschanomalie: Durch die Entfernung eines Tupels entsteht ein inkonsistenter Datenbestand.**
- **Aktualisierungsanomalie: Durch die Änderung eines vorhandenen Tupels entsteht ein inkonsistenter Datenbestand.**

Alle Arten von Anomalien gehen auf das Vorhandensein von Redundanz, mithin einem Verstoß gegen die Grundregel jede im konzeptuellen Schema modellierte Information an nur genau einer Stelle abzuspeichern, zurück.

Ausgangssituation des Normalisierungsvorganges ist die *Urrelation* die alle Attribute in genau einer Relation zusammenfaßt.

Erste Normalform

Definition 32: *Erste Normalform (1NF)*
Eine Relation ist dann in erster Normalform, wenn ihre Domänen (=Wertausprägungen der Attribute) nur einfache (atomare) Werte besitzen.

Atomarer Wert bedeutet hierbei, daß kein Attributinhalt strukturiert sein darf, d.h. durch mögliche Zerlegungsoperationen in kleine eigenständige Informationseinheiten zerlegt werden kann.

Beispiel einer Relation die **nicht in 1NF** ist:

FNAME	LNAME	ADDRESS	BDATE
John	Smith	731 Fondren, Houston, TX	1965-01-09
Franklin	Wong	638 Voss, Houston, TX	1955-12-08
Joyce			1972-07-31
Polly Esther	Wallace	291 Berry, Bellaire, TX	1941-06-20, 1952-09-04

Beispiel 19: Relation, die nicht in 1NF ist

Ziel der Überführung in 1NF ist es, Relationen zu erhalten, die in gängigen RDBMS abspeicherbar sind. Diese bieten zwar heute technische Mechanismen (wie Array- und Referenztypen) an, die Strukturen ähnlich den dargestellten verwaltbar werden lassen. Voraussetzung ihrer konzeptionellen Beherrschung ist jedoch die vorherige Normalisierung.

Im Beispiel befinden sich die Zeilen von `Franklin` und `Joyce Wong` nicht in 1NF, da sie nicht für jedes Attribut einen Wert besitzen, sondern sich eine Wertausprägung (`Wong` und `638 Voss, Houston, TX`) teilen.
Ebenso befindet sich der Eintrag von `Polly Esther Wallace` nicht in 1NF, da hier für das Geburtsdatum unerlaubterweise zwei Einträge auftreten.
Die beiden Vornamen sind im Rahmen der Semantik des Attributes `FNAME` zugelassen und daher im Normalisierungsprozeß nicht zu beanstanden.

Die Relation aus Beispiel 19 in erster Normalform:

FNAME	LNAME	ADDRESS	BDATE
John	Smith	731 Fondren, Houston, TX	1965-01-09
Franklin	Wong	638 Voss, Houston, TX	1955-12-08
Joyce	Wong	638 Voss, Houston, TX	1972-07-31
Polly Esther	Wallace	291 Berry, Bellaire, TX	1941-06-20
Polly Esther	Wallace	291 Berry, Bellaire, TX	1952-09-04

Beispiel 20: Relation, die in 1NF ist

Zur Umformung der Relation in eine Relation in erster Normalform wurden die „gemeinsamen“ Attribute aufgelöst, so das nunmehr jedes Attribut genau einem Tabelleintrag (Tupel) zugeordnet ist.
Zusätzlich wurde für jedes Attribut die atomare Belegung sichergestellt.

Die Einführung der ersten Normalform verhindert damit die Bildung *geschachtelter Relationen*, die entstünden, jedes Attribut eine Menge anderer Attribute, mithin wiederum eine vollständige Relation, enthalten könnte.
Anmerkung: Diese Forderung wird durch die in SQL:1999 definierten `ARRAY`-Typen aufgeweicht und für postrelationale und objektorientierte Datenbanken vollständig aufgegeben, weshalb diese Strukturen auch als *Non-First-Normal-Form* (kurz: NFNF, NF2 oder NF^2) bezeichnet werden.

Test auf Einhaltung der ersten Normalform:
Die Relation sollte keine nicht-atomaren Attribute oder verschachtelte Relationen enthalten.

Der algorithmische Ableitungsprozeß aus dem konzeptuellen Schema stellt durch die Organisation der Repräsentationstypen sicher, daß Attribute ausschließlich durch atomare Werte repräsentiert werden.

Zweite Normalform

Grundlegende Voraussetzung zum Verständnis der zweiten Normalform ist das Konzept der *vollen funktionalen Abhängigkeit*.

Definition 33: *Volle funktionale Abhängigkeit*
Ein Attribut *y* einer Relation ist vollfunktional abhängig von einem Attribut *x* wenn gilt, daß jede Ausprägung von *x* genau eine Ausprägung von *y* bestimmt und *y* nicht abhängig von Teilattributen von *x* ist.
Im Zeichen: *x->y*.
Die vollfunktionale Abhängigkeit wird häufig als *FD* (*functional dependency*) abgekürzt.

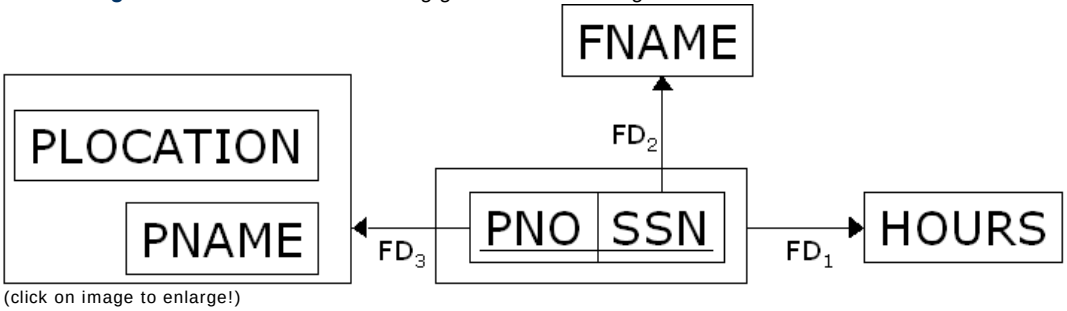
Bemerkung: Man beachte, daß der Begriff *Attribut* in Definition 33 eine nichtleere Menge von Attributen bezeichnet.

Beispiel aus der Demodatenbank:

SSN	PNO	HOURS	FNAME	DNAME	PNAME	PLOCATION
123456789	1	32.5	John	Research	ProductX	Bellaire
123456789	2	7.5	John	Research	ProductY	Sugarland
333445555	2	10.0	Franklin	Research	ProductY	Sugarland
333445555	3	10.0	Franklin	Research	ProductZ	Houston
333445555	10	10.0	Franklin	Research	Computerization	Stafford
333445555	20	10.0	Franklin	Research	Reorganization	Houston
453453453	1	20.0	Joyce	Research	ProductX	Bellaire
453453453	2	20.0	Joyce	Research	ProductY	Sugarland
666884444	3	40.0	Ramesh	Research	ProductZ	Houston
888665555	20	NULL	James	Headquarters	Reorganization	Houston
987654321	20	15.0	Jennifer	Administration	Reorganization	Houston
987654321	30	20.0	Jennifer	Administration	Newbenefits	Stafford
987987987	10	35.0	Ahmad	Administration	Computerization	Stafford
987987987	30	5.0	Ahmad	Administration	Newbenefits	Stafford
999887777	10	10.0	Alicia	Administration	Computerization	Stafford
999887777	30	30.0	Alicia	Administration	Newbenefits	Stafford

In der Relation existieren folgende voll funktionale Abhängigkeiten:

Abbildung 17: *Voll funktionale Abhängigkeiten in der dargestellten Relation*



(click on image to enlarge!)

Es ist offensichtlich, daß zwar HOURS vom vollständigen Primärschlüssel (gebildet aus PNO gemeinsam mit SSN) abhängen (FD_1), aber der Name (FNAME) und die Kombination aus PLOCATION und PNAME nur von SSN bzw. PNO und damit Teilen des Primärschlüssels abhängen (FD_2 bzw. FD_3).

Inhaltlich manifestieren sich diese Probleme im Zwang verschiedene Daten (etwa SSN und FNAME) wiederholt (redundant) abspeichern zu müssen. Ändert sich eine dieser Angaben, so muß potentiell eine große Anzahl Tupel in der Datenbank aktualisiert werden. Werden hierbei nicht

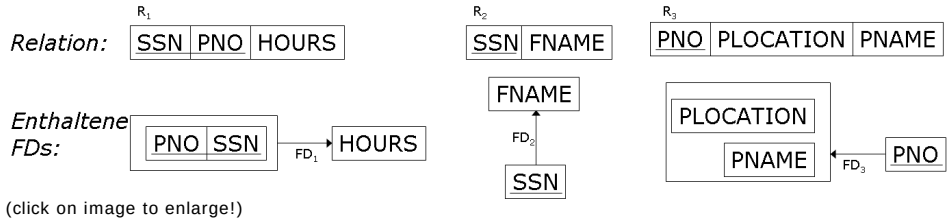
alle Datensätze aktualisiert so entsteht ein inkonsistenter Datenbestand.
Zusätzlich ist es nicht möglich bestimmte Informationszusammenhänge abzubilden. Hierunter fällt beispielsweise der Wunsch Projekte (etwa: PLOCATION und PNAME) zur verwalten, denen noch keinen Mitarbeiter zugeordnet ist.

Um eine Relation in 2NF zu überführen muß sie so zerlegt werden, daß alle Attribute der neu entstehenden Relation vom selben Schlüsselkandidaten abhängen.

Definition 34: *Zweite Normalform (2NF)*
Eine Relation ist in 2NF genau dann, wenn sie in 1NF ist und jedes Nichtschlüsselattribut voll funktional abhängig von einem Schlüsselkandidaten ist.

Entstehende Relationen:

Abbildung 18: *Relationen in 2NF*

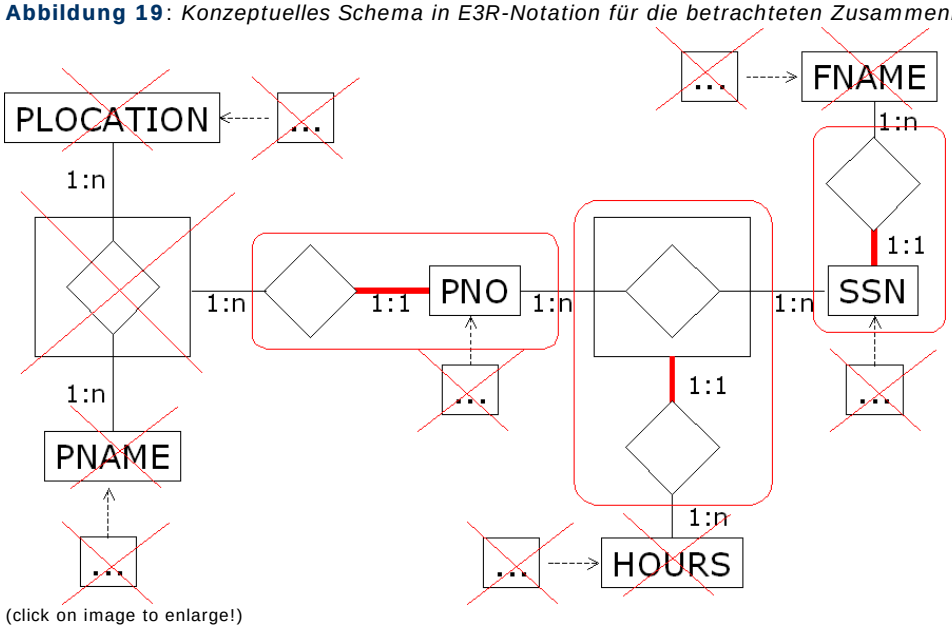


(click on image to enlarge!)

Test auf Einhaltung der zweiten Normalform:
In Relationen deren Primärschlüssel mehrere Attribute enthalten, sollte kein Nichtschlüsselattribut voll funktional von einem Teil des Primärschlüssels abhängen.

Der Ableitungsprozeß aus dem konzeptuellen Schema in E^{3}R-Notation gewährleistet automatisch die Erzeugung von Relationen in 2NF:

Abbildung 19: *Konzeptuelles Schema in E3R-Notation für die betrachteten Zusammenhänge*



(click on image to enlarge!)

Dritte Normalform (3NF)

Die dritte Normalform erweitert die für die Zweite getroffenen Aussagen dahingehend, daß zusätzlich zur voll funktionalen Abhängigkeit die *transitive Abhängigkeit* eingeführt und betrachtet wird.

Definition 35: *Transitive Abhängigkeit*
In einer Relation *R* ist ein Attribut *z* transitiv von einem Attribut *x* abhängig dann und nur dann, wenn *z* voll funktional von *y* und *y* voll funktional von *x* abhängig ist.
Im Zeichen: x->->z

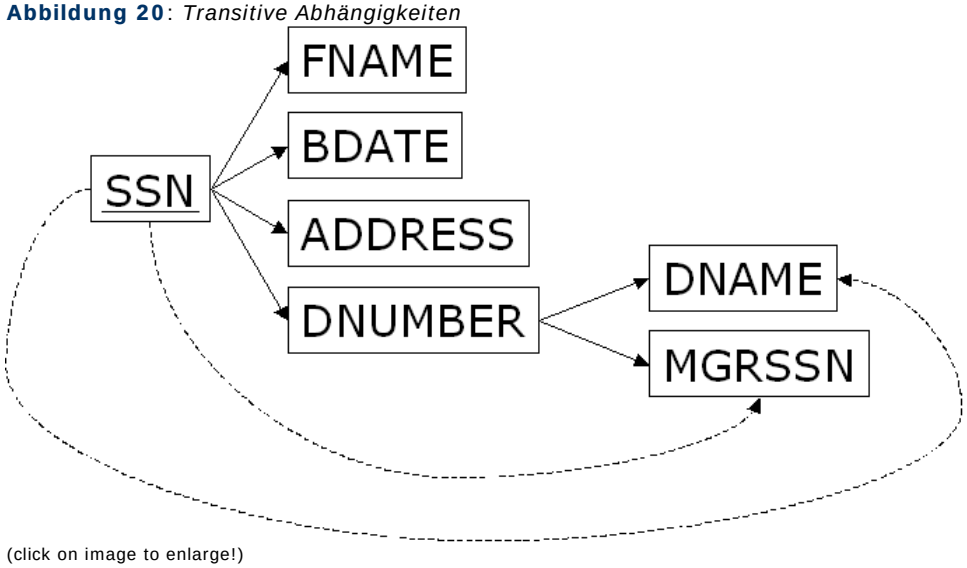
Bemerkung: Man beachte, daß der Begriff *Attribut* in Definition 35 eine nichtleere Menge von Attributen bezeichnet.

Im Beispiel der Demodatenbank:

```
+----------+-----------+------------+--------------------------+---------
+---------------+-----------+
| FNAME    | SSN       | BDATE      | ADDRESS                  | DNUMBER |
DNAME          | MGRSSN    |
+----------+-----------+------------+--------------------------+---------
+---------------+-----------+
| James    | 888665555 | 1937-11-10 | 450 Stone, Houston, TX   |       1 |
Headquarters   | 888665555 |
| Jennifer | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX  |       4 |
Administation | 987654321 |
| Ahmad    | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX  |       4 |
Administation | 987654321 |
| Alicia   | 999887777 | 1968-07-19 | 3321 Castle, Spring, TX  |       4 |
Administation | 987654321 |
| John     | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX |       5 |
Research       | 333445555 |
| Franklin | 333445555 | 1955-12-08 | 638 Voss, Houston, TX    |       5 |
Research       | 333445555 |
| Joyce    | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX   |       5 |
Research       | 333445555 |
| Ramesh   | 666884444 | 1962-09-15 | 975 Fire Oak, Humble, TX |       5 |
Research       | 333445555 |
+----------+-----------+------------+--------------------------+---------
+---------------+-----------+
```

In der Relation existieren folgende direkten und transitive Abhängigkeiten (die direkten funktionalen Abhängigkeiten sind durch gerichtete Kanten mit durchgezogener Linienführung, die Transitiven durch unterbroche Linienführung dargestellt):

Abbildung 20: *Transitive Abhängigkeiten*



(click on image to enlarge!)

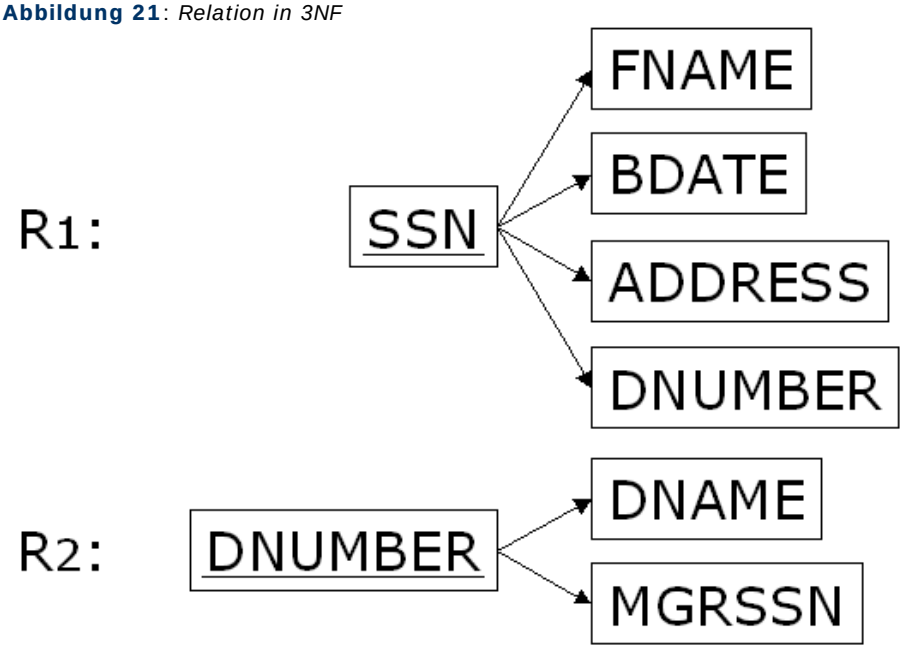
In dieser Organisationsform tritt eine Einfügeanomalie auf, wenn ein Mitarbeiter neu eingefügt wird und dabei „falsche“ (d.h. inkonsistente) Werte für die Abteilung angelegt werden. Zusätzlich fällt das Einfügen neuer Abteilungen, zu denen (noch) kein Mitarbeiter abgespeichert wird, schwer, da SSN zwingend anzugeben ist (NULL-Wert ist wegen der Definition als Primärschlüssel nicht zugelassen!).
Daneben existiert eine Löschanomalie dahingehend, daß wenn der letzte Mitarbeiter einer Abteilung aus der Datenbank entfernt wird auch alle Informationen über diese Abteilung verloren gehen.
Werden Abteilungsdaten verändert, so kann es zu einer Modifikationsanomalie kommen, da nicht in jedem Falle eindeutig klärbar ist ob nur die Abteilungsdaten dieses Mitarbeiters verändert werden sollen (beispielsweise im Falle eines Abteilungswechsels) oder die Daten aller abgespeicherten Abteilungen (beispielsweise im Falle der Umbenennung einer Abteilung).

Die dritte Normalform setzt sich zum Ziel die Ursachen dieser Anomalien zu beseitigen:

Definition 36: *Dritte Normalform (3NF)*
Eine Relation ist dann und nur dann in 3NF, wenn sie in 2NF ist und jedes Nicht-Schlüsselattribut nicht-transitiv abhängig ist von einem Schlüsselkandidaten; sie ist auch in 3NF, wenn sich eine transitive Abhängigkeit ausschließlich über Bestandteile des Schlüsselkandidaten herleiten läßt.

Gemäß dieser Definition ist die Beispielrelation zu zerlegen in:
R_1(SSN, FNAME, BDATE, ADDRESS, DNUMBER) und
R_2(DNUMBER, DNAME, MGRSSN).

Abbildung 21: *Relation in 3NF*

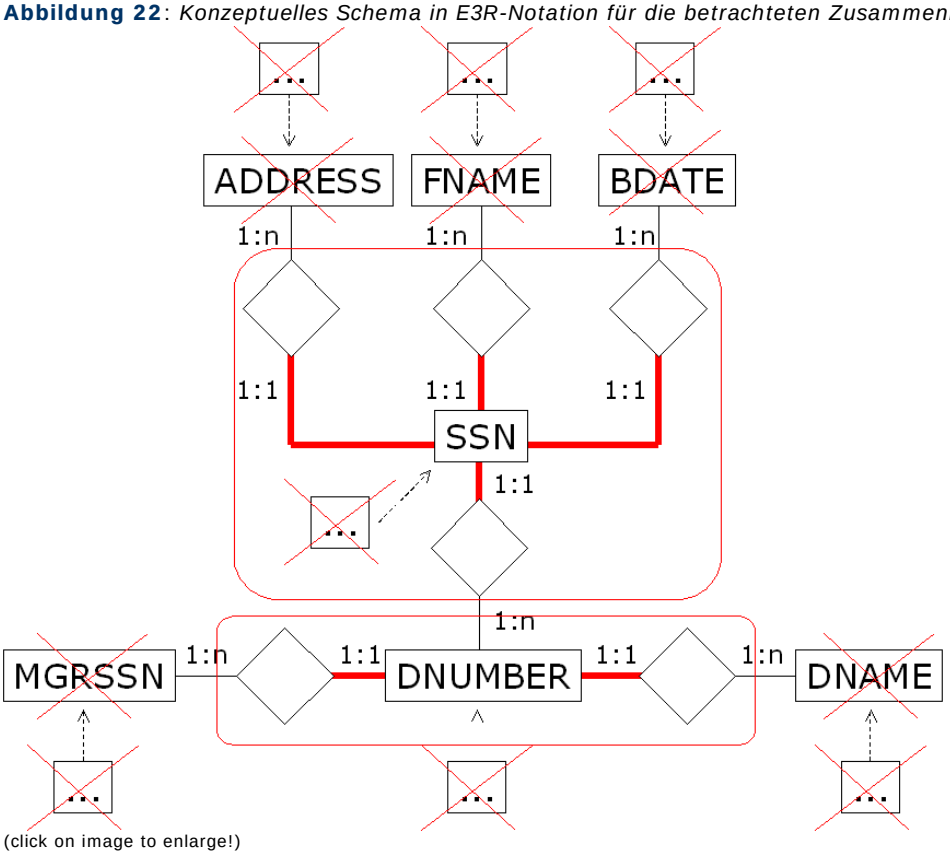


(click on image to enlarge!)

Test auf Einhaltung der dritten Normalform:
Eine Relation sollte kein Nicht-Schlüsselattribut enthalten, das voll funktional von einem anderen Nicht-Schlüsselattribut (oder von einer Menge von Nichtschlüsselattributen) abhängig ist. Das heißt, es sollte keine transitive Abhängigkeit eines Nichtschlüsselattributs vom Primärschlüssel bestehen.

Der Ableitungsprozeß aus dem konzeptuellen Schema in E^{3}R-Notation gewährleistet automatisch die Erzeugung von Relationen in 3NF:

Abbildung 22: *Konzeptuelles Schema in E3R-Notation für die betrachteten Zusammenhänge*



(click on image to enlarge!)

Das Rissanen-Theorem klärt die sich prinzipiell ergebene Fragestellung warum die in Abbildung 21 dargestellte Zerlegung gewählt wurde, da sich prinzipiell unter Ausnutzung der transitiven Abhängigkeiten auch eine (Alternativ-)Zerlegung in R_1(SSN, FNAME, BDATE, ADDRESS, DNUMBER) und R_2(SSN, DNAME, MGRSSN) angeboten hätte.

Nach dem Theorem von Heath und Rissanen ist jedoch die in Abbildung 21 gewählte Zerlegung die einzig korrekt mögliche, da sie die tatsächlich vorhandenen funktionalen Abhängigkeiten erhält, was bei obiger Alternative nicht der Fall wäre.

Angewendet auf unser Beispiel bedeutet dies, daß für obige (nach Heath/Rissenan fehlerhafte) Zerlegung beispielsweise der Anwendungsfall nach Anlage einer Abteilung (DNUMER gemeinsam mit ihrem Namen (DNAME) und der SSN ihres Abteilungsleiters (MGRSSN) nicht möglich wäre, da DNAME und MGRSSN nur verwaltet werden können, wenn gleichzeitig die als Primärschlüssel zwingend anzugebende SSN eines Mitarbeiters dieser Abteilung existiert. Mithin wäre die Speicherung einer Abteilung die nur über ihren Leider aber (noch) nicht über Mitarbeiter verfügt nicht möglich.
Die gewählte Zerlegung vermeidet jedoch diese Einschränkung und liefert, ebenso wie der Abteilungsalgorithmus aus dem konzeptuellen Schema, das korrekte Resultat.

Boyce/Codd-Normalform (BCNF)

Ursprünglich wurde die *Boyce/Codd Normalform* (BCNF) als Vereinfachung der dritten Normalform vorgeschlagen. Jedoch faßt sie diese schärfer und führt so zu einem neuen Typ von Normalform, der nach ihren Schöpfern Boyce und Codd benannt wurde.
Inhaltlich räumt sie mögliche Anomalien aus, die in Relationen, welche sich bereits in 3NF befinden, noch auftreten können.

Definition 37: *Boyce/Codd-Normalform*
In einer Relation ist dann und nur dann in BCNF, wenn sie in 3NF ist und gleichzeitig jede Determinante Schlüsselkandidat ist.

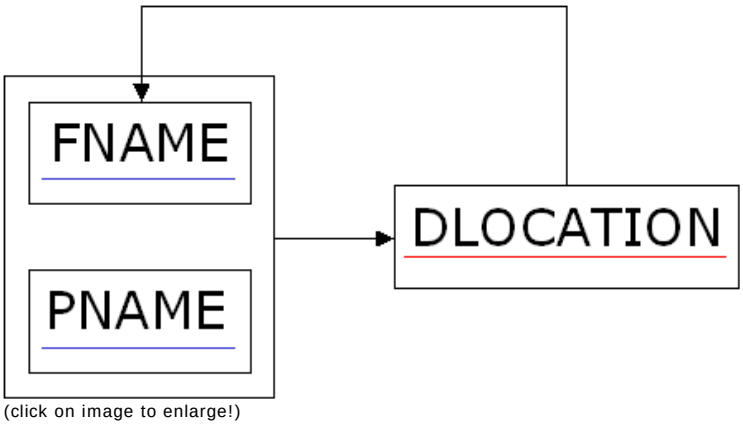
Hierbei definiert die BCNF den Begriff der *Determinante* als den Ausgangspunkt einer funktionalen Abhängigkeit.

Im Beispiel der Demodatenbank:

FNAME	PNAME	DLOCATION
Franklin	ProductY	Sugarland
Franklin	ProductZ	Houston
Jennifer	Newbenefits	Stafford
...	...	...

In der Relation existieren folgende funktionale Abhängigkeiten:

Abbildung 23: *Funktionale Abhängigkeiten*



(click on image to enlarge!)

Die Schlüsselkandidaten sind durch farbliche Unterstreichung hervorgehoben. Hierbei bestimmt FNAME gemeinsam mit PNAME eindeutig einen Wert für DLOCATION (blau) und DLOCATION (rot) bestimmt eindeutig einen Wert für FNAME.

Da sich die transitive Abhängigkeit PNAME->DLOCATION->FNAME ausschließlich über Schlüsselkandidaten herleitet ist die Relation in 3NF.
Dennoch ist sie nicht anomalienfrei, da sie beispielsweise die Ablage von Abteilungsstandorten (DLOCATION) und den Namen der Abteilungsleiter (FNAME) verbietet, wenn kein Projektname (PNAME) zusätzlich abgespeichert wird.

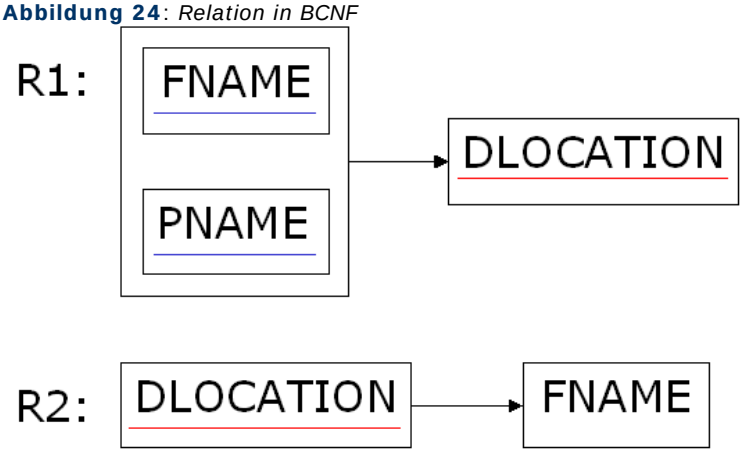
Ebenso ist die Ablage von Projekten und den zuständigen Abteilungen erst dann möglich, wenn zusätzlich der Name des Abteilungsleiters bekannt ist.

Dieses Manko wird durch die Forderung der BCNF behoben. Sie erzwingt die Zerlegung der abgebildeten Ausgangsrelation in zwei Eigenständige. Hierbei wird jede Determinante der Ausgangsrelation zum Schlüsselkandidaten in den neu gebildeten Relationen.

Gemäß Definition 37 ist die Beispielrelation zu zerlegen in:

R_1(FNAME, PNAME, DLOCATION)
R_2(DLOCATION, FNAME).

Abbildung 24: *Relation in BCNF*

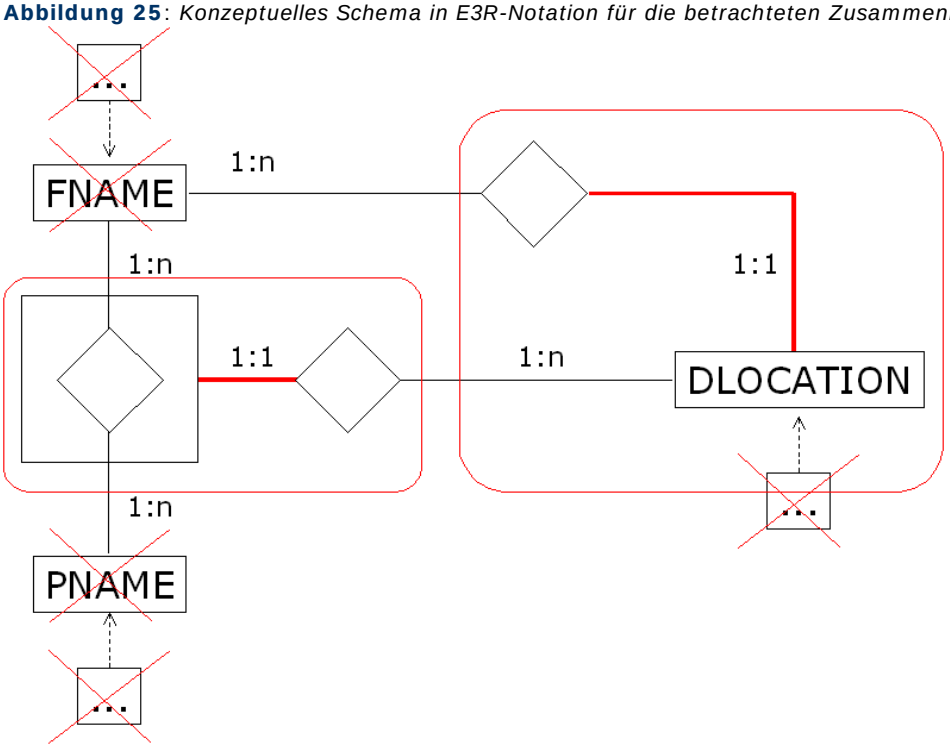


(click on image to enlarge!)

Test auf Einhaltung der Boyce/Codd-Normalform:
Eine Relation sollte lediglich direkt vom Schlüssel abhängende Attribute enthalten.

Der Ableitungsprozeß aus dem konzeptuellen Schema in E^{3}R-Notation gewährleistet automatisch die Erzeugung von Relationen in BCNF:

Abbildung 25: *Konzeptuelles Schema in E3R-Notation für die betrachteten Zusammenhänge*



(click on image to enlarge!)

Vierte Normalform (4NF)

Die bisher betrachteten Normalformen nutzen alle das Vorhandensein funktionaler Abhängigkeiten aus. Jedoch existieren neben diesem Beziehungstyp auch noch andere, im vorhergehenden nicht berücksichtige, Abhängigkeit.

Definition 38: *Mehrwertige Abhängigkeit*
Eine mehrwertige Abhängigkeit (*multivalue dependency*, MVD) ist eine Abhängigkeit, die einem

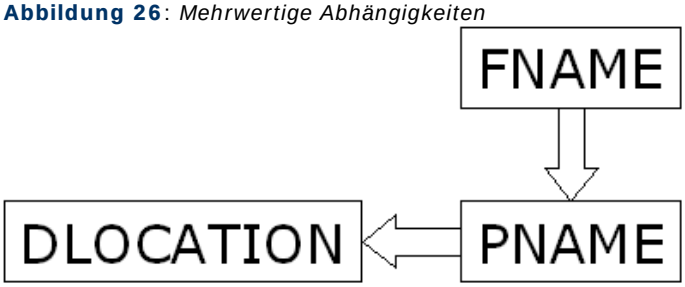
einem Attribut eine Menge verschiedener Werte zuordnet.

Das Vorhandensein mehrwertiger Abhängigkeiten in einer Relation kann zu Aktualisierungsanomalien führen, wie die Betrachtung des bekannten Beispiels aus der Demodatenbank:

FNAME	PNAME	DLOCATION
Franklin	ProductY	Sugarland
Franklin	ProductZ	Houston
Jennifer	Newbenefits	Stafford
...	...	...

In der Relation existieren folgende mehrwertige Abhängigkeiten:

Abbildung 26: *Mehrwertige Abhängigkeiten*



(click on image to enlarge!)

Die Existenz der dargestellten mehrwertigen Abhängigkeiten in der Datenbank führt dazu, daß dieselbe Information mehrfach abgespeichert werden muß. So enthält die Beispieldatenbank mehrfach denselben FNAME sowie wiederholt denselben Produktnamen (PNAME).
Als Konsequenz dieser Wiederholung müssen bei jedem Aktualisierungsvorgang, der die Zuordnung des Produktes zum FNAME betrifft alle FNAME-Einträge geändert werden, jedoch bei der Zuständigkeitsänderung eines Produktverantwortlichen nur der betroffene FNAME.

Ziel der vierten Normalform ist die Elimination von Redundanzen, die aus mehrwertigen Abhängigkeiten herrühren. Allerdings können mehrwertige Abhängigkeiten nicht vollständig entfernt werden, daher ist für die Normalisierung gemäß 4NF die Eliminierung aller *nicht trivialen* mehrwertigen Abhängigkeiten als Zielsetzung definiert.

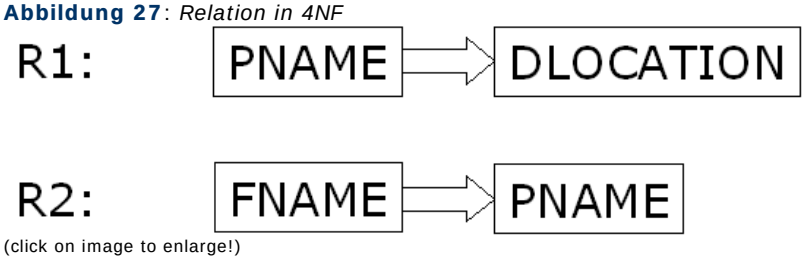
Eine *triviale mehrwertige Abhängigkeit* ist hierbei festgelegt als:

Definition 39: *Triviale mehrwertige Abhängigkeit*
Eine triviale mehrwertige Abhängigkeit ist eine mehrwertige Abhängigkeit zwischen Attributmengen X und Y der Relation R für die gilt: Y ist eine Teilmenge von X oder die Vereinigung von X und Y bildet R.

Definition 40: *Vierte Normalform*
Eine Relation ist dann in vierter Normalform, wenn sie nur noch triviale mehrwertige Abhängigkeiten enthält.

Gemäß dieser Definition ist die Beispielrelation zu zerlegen in:
R_1(DLOCATION, PNAME) und
R_2(FNAME, PNAME).

Abbildung 27: *Relation in 4NF*

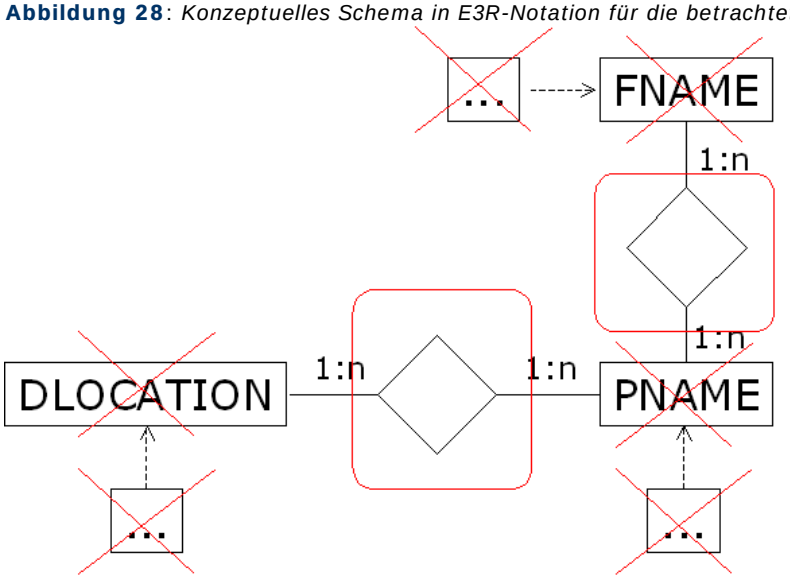


(click on image to enlarge!)

Test auf Einhaltung der vierten Normalform:
Eine Relation sollte keine nicht trivialen mehrwertigen Abhängigkeiten enthalten.

Der Ableitungsprozeß aus dem konzeptuellen Schema in E^{3}R-Notation gewährleistet automatisch die Erzeugung von Relationen in 4NF:

Abbildung 28: *Konzeptuelles Schema in E3R-Notation für die betrachteten Zusammenhänge*



(click on image to enlarge!)

Fünfte Normalform (5NF)

Die fünfte Normalform greift anders als alle vorhergehenden nicht auf die Zusammenhänge der Typebene zurück, sondern benötigt zu ihrer Untersuchung die Betrachtung von tatsächlichen Datenbankinhalten.
Ausgehend von diesen definiert sie die fünfte Normalform als:

Definition 41: *Fünfte Normalform*
Eine Relation ist dann in fünfter Normalform (5NF), bei ihrer Zerlegung durch Projektionen und deren anschließender Kombination durch Verbundoperationen keine Tupel gebildet werden, die nicht Bestandteil der Ausgangsrelation waren.

Aufgrund ihrer Abstützung auf die Projektions- und Verbundoperation (engl. *join*) wird die 5NF auch als *Project Join Normalform* (PJNF) bezeichnet.

Im Beispiel der Demodatenbank:

Relation `Ur`:

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

Durch Projektion ergeben sich die drei möglichen Relationen:

```
SELECT Ur.FNAME, Ur.DNUMBER INTO R11 FROM Ur
```

R_{11}:

FNAME	DNUMBER
John	5
John	5
James	1

```
SELECT Ur.FNAME, Ur.DLOCATION INTO R12 FROM Ur
```

R_{12}:

FNAME	DLOCATION
John	Bellaire
John	Houston

James	Houston

```
SELECT Ur.DNUMBER, Ur.DLOCATION INTO R13 FROM Ur
```

R_{13}:

DNUMBER	DLOCATION
5	Bellaire
5	Houston
1	Houston

Durch Verbundoperationen entstehen die Relationen:

```
SELECT R11.FNAME, R11.DNUMBER, R12.DLOCATION INTO R21 FROM R11 INNER JOIN R12 ON
R11.FNAME=R12.FNAME
```

R_{21}:

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

```
SELECT R11.FNAME, R11.DNUMBER, R13.DLOCATION INTO R22 FROM R11 INNER JOIN R13 ON
R11.DNUMBER = R13.DNUMBER
```

R_{22}:

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

```
SELECT R12.FNAME, R12.DLOCATION, R13.DNUMBER INTO R23 FROM R12 INNER JOIN R13 ON
R12.DLOCATION=R13.DLOCATION
```

R_{23}:

FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston
James	5	Houston
John	1	Houston

Trotz der ausschließlich durch Rekombination der zuvor aus der Urrelation gebildeten Projektionen (d.h. ohne Hinzunahme neuer) Daten „entstehten“ in Relation R_{23} (rot hervorgehobene) Tupel (sog. *spurious tupel*), die in dieser Form nicht in der Ausgangsrelation präsent waren.

Erst das Zusammenfügen aller aus Verbundoperationen erzeugten Relationen durch einen erneuten Verbund eliminiert diesen *unechten Tupel*:

```
SELECT R21.FNAME, R22.DNUMBER, R23.DLOCATION INTO RESULT FROM (R21 INNER JOIN R22
ON (R21.FNAME = R22.FNAME) AND (R21.DNUMBER = R22.DNUMBER) AND (R21.DLOCATION =
R22.DLOCATION)) INNER JOIN R23 ON (R22.FNAME = R23.FNAME) AND (R22.DNUMBER = R23.
DNUMBER) AND (R22.DLOCATION = R23.DLOCATION) AND (R21.FNAME = R23.FNAME) AND (R21.
DNUMBER = R23.DNUMBER) AND (R21.DLOCATION = R23.DLOCATION)
```

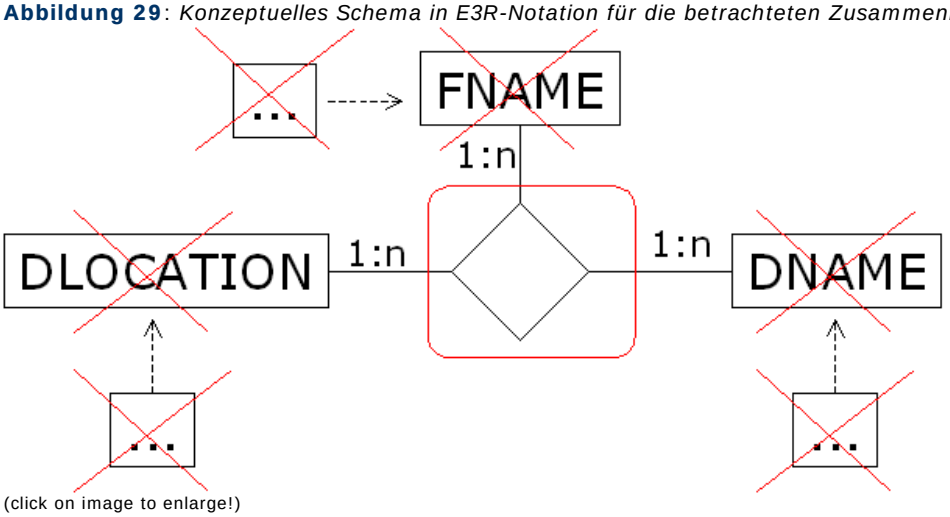
FNAME	DNUMBER	DLOCATION
John	5	Bellaire
John	5	Houston
James	1	Houston

Das Auftreten unechter Tupel ist ein Indiz für eine Relation in der Verbundabhängigkeiten existieren. Eine solche Relation darf nicht weiter zerlegt werden, da durch die Entfernung von Attributen Information verloren ginge.

Test auf Einhaltung der fünften Normalform:
Wenn durch Projektions- und Verbundoperationen keine unechten Tupel entstehen, dann ist die Relation in 5NF.

Der Ableitungsprozeß aus dem konzeptuellen Schema in E^3R-Notation gewährleistet automatisch die Erzeugung von Relationen in 5NF:

Abbildung 29: *Konzeptuelles Schema in E3R-Notation für die betrachteten Zusammenhänge*



(click on image to enlarge!)

Die 5NF ist die höchstmögliche über den Normalisierungsprozeß erreichbare Normalform.
Dies bedeutet jedoch nicht, daß jede Relation bis in 5NF gebracht werden kann. Der Normalisierungsprozeß endet generell mit der höchstmöglichen Normalform, dies muß jedoch nicht immer 5NF sein, sondern orientiert sich an den modellierten Informationszusammenhängen.

Weitere Normalisierungsaspekte

Über die im vorhergehedenden betrachteten formalisierten Normalformen hinaus existieren noch weitere Abhängigkeiten und Normalformen-ähnliche Güteaussagen für Datenbanken.

Inklusionsabhängigkeit:
Die Inklusionsabhängigkeit (engl. *inclusion dependence*, ID) beschreibt die Beziehungen zwischen Super- und Subtypen, insbesondere die *Attributentsprechung*, d.h. die Tatsache, daß der Subtype mindestens alle Attribute des Supertypen aufweist, sowie die *Kompatibilitätsrelation* worunter die Austauschbarkeit von Ausprägungen der beiden Typen verstanden wird für Anwendungsbereiche die lediglich auf die gemeinsam vorhandenen Attribute zugreifen.
aus der Inklusionsabhängigkeit folgen u.a. die drei Inferenzregeln:
$IDIR_1$: (Reflexivität) Die Inklusionsabhängigkeit ist reflexiv.
$IDIR_2$: (Attributentsprechung) Verfügen zwei Typen über dieselben Attribute, dann entsprechen sie sich.
$IDIR_3$: (Transitivität) Ist *B* Untertyp von *A* und *C* Untertyp von *B*, dann ist auch *C* Untertyp von *A*.
Trotz dieser formalisierbaren Aussagen und der breiten Verwendung von Modellierungskonstrukten zur Darstellung von Spezialisierungsbeziehungen wurden auf Basis der Inklusionsabhängigkeit bisher noch keine Normalformen vorgeschlagen.

Template-Abhängigkeiten:
Die Idee der Template-Abhängigkeit fußt auf der Definition einer Reihe von *Hypothesetupeln* und daraus abgeleiteten *Konklusionstupeln*, welche gültige Ausprägungen der Datenbank abstrahiert beispielhaft aufzeigen.
Template-Abhängigkeiten gestatten die einfach Formulierung von Intrarelationsabhängigkeiten die sich auf konkrete Wertausprägungen einzelner Attribute beziehen.
Das Beispiel zeigt die Abhängigkeit, daß kein Angestellter mit mehr Einkommen ausgestattet sein darf als sein Vorgesetzter:

```
EMPLOYEE(FNAME, SSN ..., SALARY, SUPERSSN)
             a     b ...    c     d
Hypothese    e     f ...    g     h
-------------------------------------
Konklusion       c < g
```

Domain-Key-Normalform (DKNF):
Grundannahme der Domain-Key-Normalform (DKNF) ist es, daß wenn für jedes Attribut einer Relation eine Domäne (d.h. die Menge der zugelassenen Wertbelegungen) angegeben wird, alle Änderungsanomalien verschwinden.
Gleichzeitig fordert die DKNF die eineindeutige Identifikation jedes Attributs einer Relation durch einen Schlüssel.
In der Praxis kann jedoch die Angabe einer allgemein formulierten eindeutigen Domäne mit unter zu Schwierigkeiten führen, weshalb die Prüfung auf Einhaltung dieser Normalform mit erheblichen technsichen Umsetzungsschwierigkeiten verbunden ist.

3 Arbeiten mit einer Datenbank

3.1 Codd'sche Regeln und Eigenschaften relationaler Systeme

Trotz des in dieser Hinsicht sehr eindeutigen grundlegenden Papiers von E. F. Codd über die Relationenstruktur (*A Relational Model of Data for Large Shared Data Banks*) existierte lange Zeit keine Übereinkunft darüber welche Eigenschaften ein relationales DBMS mindestens aufweisen muß um dieser Systemklasse zugerechnet werden zu können.
Daher definiert Codd 1986 in einem zweiteiligen Artikeln für die Zeitschrift *Computer World* 12 strenge Regeln die ein RDBMS aus seiner Sicht zwingend erfüllen muß um als solches eingestuft werden zu können.
Diese hierin erhobenen Forderungen sind jedoch so streng, daß sie bis heute kein System vollständig erfüllt.
Die Regeln sind nachfolgend mit ihren englischsprachigen Originalbezeichnungen wiedergegeben, da sich für sie bisher keine eindeutige und allgemeinverständliche deutsche Übersetzung etablieren konnte.

Regel 1: **The Information Rule**:
Alle Daten, die in einer Datenbank gespeichert werden sind auf dieselbe Art dargestellt, nämlich durch Werte in Tabellen.
Anmerkung: In dieser Definition wurde bewußt der Begriff der Tabelle gegenüber dem der Relation bevorzugt.

Regel 2: **Guaranteed Access Rule**:
Jeder gespeicherte Wert muß über Tabellenname, Spaltenname und Wert des Primärschlüssels zugreifbar sein, wenn der zugreifende Anwender über hinreichende Zugriffsrechte verfügt.

Regel 3: **Systematic Treatment of Null Values**:
Nullwerte müssen datentypunabhängig zur Darstellung fehlender Werte unterstützt werden.
Systematisch drückt hierbei aus, daß Nullwerte unabhängig von denjenigem Datentyp für den sie auftreten gleich behandelt werden.

Regel 4: **Dynamic On-line Catalog Based on the Relational Model**: Forderung nach einem Online-Datenkatalog (*data dictionary*) in Form von Tabellen.
Dieser Katalog beschreibt die in der Datenbank abgelegten Tabellen hinsichtlich ihrer Struktur und zugelassenen Inhaltsbelegungen.

Regel 5: **Comprehensive Data Sublanguage Rule**:
Für das DBMS muß mindestens eine Sprache existieren durch die sich die verschiedenen Inhaltstypen (Tabelleninhalte, Sichten, Integritätsstrukturen (Schlüsselbeziehungen, Wertebereichseinschränkungen, Aufzählungstypen) sowie Zugriffsrechte) definieren lassen.

Regel 6: **View Updating Rule**:
Sofern theoretisch möglich, müssen Inhalte von Basistabellen auch über deren Sichten änderbar sein.

Regel 7: **High-level Insert, Update, and Delete**:
Innerhalb einer Operation können beliebig viele Tupel bearbeitet werden, d.h. die Operationen werden grundsätzlich mengenorientiert ausgeführt. Hierfür ist eine so abstrahierte Sicht dieser Operationen notwendig, daß keinerlei Information über die syteminterne Darstellung der Tupel notwendig ist.

Regel 8: **Physical Data Independence**:
Änderungen an der internen Ebene dürfen keine Auswirkungen auf die auf den abgespeicherten Daten operierenden Anwendungsprogramme besitzen.
Werden Daten demnach reorganisiert oder beispielsweise durch Indexe zugriffsbeschleunigt, so darf eine solche Änderung die auf die Datenbank zugreifenden Anwendungsprogramme nicht beeinträchtigen.

Regel 9: **Logical Data Independence**:
Änderungen des konzeptuellen Schemas dürfen keine Auswirkung auf die Anwedungsprogramme besitzen, solange diese nicht direkt von der Änderung betroffen sind.

Regel 10: **Integrity Independence**:
In Verfeinerung der fünften Regel wird gefordert, daß alle Integritätsbedingungen ausschließlich durch die Sprache des DBMS definieren lassen können müssen. Definierte Integritätsbedingungen müssen in Tabellen abgespeichert werden und durch das DBMS zur Laufzeit abgeprüft werden.

Im Mindesten werden folgende Forderungen durch verfügbare Systeme unterstützt:

- Kein Attribut welches Teil eines Primärschlüssels ist darf NULL sein.
- Ein Fremdschlüsselattribut muß als Wert des zugehörigen Primärschlüssels existieren.

Regel 11: **Distribution Independence**:
Die Anfragesprache muß so ausgelegt sein, daß Zugriffe auf lokal gehaltene Daten identisch denen auf verteilt gespeicherte Daten formuliert werden können.
Hieraus läßt sich auch die Ausdehnung der Forderungen nach logischer und physischer Datenunabhängigkeit für verteilte Datenbanken ableiten.

Regel 12: **Nonsubversion Rule**:
Definiert ein DBMS neben der High-level Zugriffssprache auch eine Schnittstelle mit niedrigerem Abstraktionsniveau, dann darf durch diese keinesfalls eine Umgehung der definierten Integritätsregeln möglich sein.

Zusätzlich faßt Codd in **Regel 0** nochmals die Anforderungen dahingehend zusammen, daß er postuliert, alle Operationen für Zugriff, Verwaltung und Wartung der Daten ausschließlich mittels relationaler Fähigkeiten abzuwickeln.

Derzeit existiert kein am Markt verfügbares kommerzielles System welches alle zwölf Regeln vollständig umsetzt. Insbesondere sind die Regeln 6, 9, 10, 11 und 12 in der Praxis schwer umzusetzen.

Darüber hinaus greifen die Coddschen Regeln nicht alle Gesichtspunkte des praktischen Datenbankeinsatzes auf. So bleiben Fragestellungen des Betriebs (wie Sicherungs-, Wiederherstellungs- und Sicherheitsaspekte) eines DBMS völlig ausgeklammert.

3.2 Implementierung des logischen Modells mit SQL-DDL

Die SQL-DDL dient allgemein der Definition und Verwaltung von Tabellen- und Indexdefinitionen innerhalb einer relationalen Datenbank entlang ihres gesamten Lebenszyklus, d.h. von ihrer Erstellung über alle Wartungsstadien bis hin zur Entfernung.

Nicht normiert durch den SQL-Standard sind die notwendigen Schritte zur Erzeugung einer Datenbank innerhalb eines Datenbankmanagementsystems. Überdies variieren die hierfür abzusetzenden Kommandos von Hersteller zu Hersteller und müssen der spezifischen Dokumentation entnommen werden.

Hinweis: Zwar läßt SQL inzwischen die beliebige Schreibung der Schlüsselworte (groß, klein oder gemischt) zu, zur bessern Hervorhebung und Kompatibilität mit existierender Literatur werden sie jedoch in den nachfolgenden Syntaxübersichten und Beispielen durchgehend in Großschreibung wiedergegeben.

Innerhalb der Syntaxbeschreibungen gelten folgende Konventionen:

- Schlüsselworte, die direkt wie abgedruckt eingegeben werden müssen sind großgeschrieben.
- Optionale Bestandteile, die weggelassen werden können sind in eckigen Klammern („[]“) dargestellt.

Die Klammern selbst sind nicht Bestandteil der Syntax und müssen nicht eingegeben werden.

- Dargestellte runde Klammern („()“) sind Syntaxbestandteil und müssen unverändert eingegeben werden.
- Senkrechte Striche („|“) trennen Alternativen von denen jeweils eine ausgewählt werden kann, nicht jedoch mehrere.
- Kommentare, die auf die Ausführung keinen Einfluß haben werden durch zwei Minuszeichen („--“) eingeleitet und enden mit dem Zeilenende.
- Alle SQL-Befehle werden generell durch ein Semikolon abgeschlossen. Dieses ist aus Übersichtlichkeitsgründen in den Syntaxübersichten weggelassen.

Erzeugen von Tabellen

Die SQL-Anweisung `CREATE TABLE` dient der Erzeugung neuer Tabellen innerhalb einer bestehenden Datenbank. Sie legt die Struktur und die zugelassenen Typausprägungen, sowie Einschränkungen hinsichtlich der erlaubten Werte fest.

Die vereinfachte Syntax der `CREATE TABLE`-Anweisung lautet:

```
CREATE [TEMPORARY] TABLE tbl_name [(create_definition,...)]
[table_options] [select_statement]

create_definition:
  col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
            [PRIMARY KEY] [reference_definition]
  or    PRIMARY KEY (index_col_name,...)
  or    KEY [index_name] (index_col_name,...)
  or    INDEX [index_name] (index_col_name,...)
  or    UNIQUE [INDEX] [index_name] (index_col_name,...)
  or    [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name,...)
            [reference_definition]

type:
        TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  or    SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  or    MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  or    INT[(length)] [UNSIGNED] [ZEROFILL]
  or    INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  or    BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  or    REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
  or    NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
  or    CHAR(length) [BINARY]
  or    VARCHAR(length) [BINARY]
  or    DATE
  or    TIME
  or    TIMESTAMP
  or    DATETIME
  or    TINYBLOB
  or    BLOB
  or    MEDIUMBLOB
  or    LONGBLOB
  or    TINYTEXT
  or    TEXT
  or    MEDIUMTEXT
  or    LONGTEXT
  or    ENUM(value1,value2,value3,...)
  or    SET(value1,value2,value3,...)

index_col_name:
        col_name [(length)]

reference_definition:
        REFERENCES tbl_name [(index_col_name,...)]
                   [MATCH FULL | MATCH PARTIAL]
                   [ON DELETE reference_option]
                   [ON UPDATE reference_option]
```

```
reference_option:
        RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

```
mysql> CREATE TABLE Person(
        Name VARCHAR(25)
);
```

Beispiel 21: Erzeugung einer Tabelle

Die Anweisung aus Beispiel 21 stellt den einfachsten Fall ein Tabellenerzeugungsanweisung dar. Es wird die Tabelle `Person`, die mit `Name` nur über eine Spalte verfügt erzeugt.
Eine „kleinere“ Fassung ist nicht möglich, da spaltenlose Tabellen nicht erstellt werden können.

Informationen über eine angelegte Tabelle können durch den Befehl `DESCRIBE` gefolgt vom Namen der abzufragenden Tabelle erlangt werden:

```
mysql> DESCRIBE Person;
+-------+-------------+-------------------+------+-----+---------+-------+
| Field | Type        | Collation         | Null | Key | Default | Extra |
+-------+-------------+-------------------+------+-----+---------+-------+
| Name  | varchar(25) | latin1_swedish_ci | YES  |     | NULL    |       |
+-------+-------------+-------------------+------+-----+---------+-------+
```

Beispiel 22: Ermittlung von Tabelleninformation

Im Beispiel werden die zuvor festgelegten Daten wie Spaltenname (`Name`) und Datentyp (`VARCHAR (25)`) ermittelt, sowie die Belegung einiger Vorgabewerte (u.a. `NULL` und `Default`).

Durch Angabe des Schlüsselwortes `TEMPORARY` können Tabellen erstellt werden, die während der Arbeit mit der Datenbank den herkömmlichen gleichgestellt behandelt werden, jedoch dem Ende der Datenbankverbindung automatisch inclusive aller darin abgelegten Daten aus der Datenbank entfernt werden.

```
mysql> CREATE TEMPORARY TABLE Person2(
        Name VARCHAR(25)
);

--Verbindungsende
--Aufbau einer neuen Verbindung

mysql> DESCRIBE Person2;
ERROR 1146: Table 'SQLTest.Person2' doesn't exist
```

Beispiel 23: Erzeugung einer temporären Tabelle

Wird die Datenbankverbindung getrennt und neu aufgebaut, so ist die zuvor temporär erstellte Tabelle Peson2 nicht mehr vorhanden und zugreifbar. Das `DESCRIBE`-Kommando liefert daher einen Fehler.

Wie bereits in Beispiel 21 gezeigt muß jede Spalte einer Tabelle einen Datentyp besitzen. Dieser definiert die zugelassenen Wertbelegungen und wird durch das DBMS bei jeder schreibenden Operation (d.h. Einfügen, Ändern und Leeren) geprüft.
Wird versucht ein ungültiger Wert zu setzen, so erfolgt eine Fehlermeldung und die Ablehnung des Eintragungs- oder Änderungswunsches. Im Falle eines zeichenkettenwertigen Datentyps (z. B. `VARCHAR`) erfolgt keine Fehlermeldung, sondern die Werte werden nur abgeschnitten eingefügt.

```
mysql> CREATE TABLE Person(
        Name VARCHAR(25)
);

mysql> INSERT INTO Person values("Max Mustermann");
--ok

mysql> INSERT INTO Person values("Franz Obermüller-Hinterhuber-Niedermayer");
--keine Fehlermeldung
--Wert wird jedoch nur abgeschnitten eingefügt:

mysql> SELECT * FROM Person;
+---------------------------+
| Name                      |
+---------------------------+
| Max Mustermann            |
| Franz Obermüller-Hinterhu |
+---------------------------+
```

Beispiel 24: Auswirkung von Datentypen I

Beispiel 24 zeigt die Auswirkung eines gesetzten Datentypen VARCHAR. Werden, wie im Beispiel Werte eingefügt, die die zulässige maximale Zeichenkettenlänge überschreiten, so werden die überzähligen Zeichen ohne Fehlermeldung abgeschnitten.

```
mysql> CREATE TABLE Person(GebDat date);

mysql> INSERT INTO Person values ('1970-12-12');
--ok

mysql> insert into Person values ('1970-19-42');
--offensichtlich falsch
--Übernommener Wert: 0000-00-00
```

Beispiel 25: Auswirkung von Datentypen II

Im Beispiel 25 wird ein offensichtlich ungültiges Datum eingefügt. Die Datenbank übernimmt jedoch nicht diesen falschen Wert sondern setzt den Vorgabewert von `0000-00-00`.
Dasselbe geschieht auch bei numerischen Datentypen (etwa: `INTEGER`) wenn versucht wird sie mit einer Zeichenkette zu belegen.

MySQL unterstützt drei Datentypklassen:

- Numerischedatentypen zur Darstellung von Zahlen.
- Zeichenkettendatentypen zur Darstellung von Texten und Binärdaten.
- Datumsdatentypen zur Darstellung von Uhrzeit- und Datumsformaten.

Die derzeit durch MySQL angebotenen Datentypen sind in der nachfolgenden Tabelle zusammengestellt:

Typname	Beispiel	Bemerkung
Numerische Datentypen		
`TINYINT [UNSIGNED]`	(Vorzeichenbehaftete) Ganzzahl der Breite acht Bit.	(2^7, ..., -1) 0, 1, ..., 2^7-1), (2^7, ... 2^8-1) (-128, ..., -1,) 0, 1, ..., 127, (128, ..., 255)
`BOOL`	Boole'scher Wahrheitswert.	Zugelassene Belegungen `0` und `1`. Synonym für `TINYINT(1)`
`SMALLINT [UNSIGNED]`	(Vorzeichenbehaftete) Ganzzahl der Breite 16 Bit.	(2^{15}, ..., -1) 0, 1, ..., 2^{15}-1), (2^{15}, ... 2^{16}-1) (-32.768, ..., -1,) 0, 1, ..., 32.767, (32.768, ..., 65.535)
`MEDIUMINT [UNSIGNED]`	(Vorzeichenbehaftete) Ganzzahl der Breite 24 Bit.	(2^{23}, ..., -1) 0, 1, ..., 2^{23}-1), (2^{23}, ... 2^{24}-1) (-8.388.608, ..., -1) 0, 1, ..., 8.388.607, (8.388.608, ..., 16.777.215)
`INT [UNSIGNED]`	(Vorzeichenbehaftete) Ganzzahl der Breite 32 Bit.	(2^{31}, ..., -1) 0, 1, ..., 2^{31}-1), (2^{31}, ... 2^{32}-1) (-2.147.483.648, ..., -1) 0, 1, ..., 2.147.483.647, (2.147.483.648, ..., 4.294.967.295)

INTEGER [UNSIGNED]		Synonym für INT
BIGINT [UNSIGNED]	(Vorzeichenbehaftete) Ganzzahl der Breite 64 Bit.	(2^{63}, ..., -1) 0, 1, ..., 2^{63}-1), (2^{63}, ... 2^{64}-1) (-9.223.372.036.854.774.808, ..., -1) 0, 1, ..., 9.223.372.036.854.774.807, (9.223.372.036.854.774.808, ..., 18.446.744.073.709.551.615)
FLOAT [UNSIGNED]	(Vorzeichenbehaftete) Fließkommazahl der Breite 32 Bit, gemäß dem Standard IEEE-754	$(-(2-2^{-23})^{127}$, ..., $(2-2^{-23})^{127})$ $-3{,}402...*10^{38}$, ..., $-1{,}175...*10^{-38}$ und $1{,}175...*10^{-38}$ ... $3{,}402...*10^{38}$
DOUBLE [UNSIGNED]	(Vorzeichenbehaftete) Fließkommazahl der Breite 64 Bit, gemäß dem Standard IEEE-754	$(-(2-2^{-52})^{1023}$, ... $(2-2^{-52})^{1023}$ $-1{,}797...*10^{-308}$, ..., $-2{,}225...*10^{308}$ und $1{,}797...^{308}$, ..., $2{,}225...^{-308}$
REAL [UNSIGNED]		Synonym für DOUBLE
DECIMAL [(Genauigkeit, [Nachkommastellen])]	Festkommazahl beliebiger Genauigkeit	DECIMAL(9,2) liefert eine Dezimalzahl mit neun Gesamtziffern, davon zwei Nachkommastellen.
DEC		Synonym für DECIMAL
NUMERIC		Synonym für DECIMAL
Zeichenkettendatentypen		
CHAR [Länge]	Textfeld konstanter Länge bis zu 255 Zeichen, das immer die angegebene Anzahl Speicherstellen benötigt.	Dies ist ein Test0x20;0x20;0x20;
CHARACTER		Synonym zu CHAR
NCHAR		Synonym zu CHAR
NATIONAL CHARACTER		Synonym zu CHAR
CHARACTER VARYING		Synonym zu VARCHAR
NATIONAL VARCHAR		Synonym zu VARCHAR
VARCHAR [Länge]	Textfeld variabler Länge. Überzählige Leerzeichen am Ende einer Zeichenkette werden vor dem Abspeichern entfernt.	Dies ist ein Test
TINYTEXT	Textfeld variabler Länge, bis zu 2^8-1 (=255) Zeichen.	... etwas mehr Text
TEXT	Textfeld variabler Länge, bis zu 2^{16}-1 (=65.535) Zeichen.	Sehr viel ... Text hier ...
MEDIUMTEXT	Textfeld variabler Länge, bis zu 2^{24}-1 (=16.777.215) Zeichen.	... etwas mehr Text hier ...
LONGTEXT	Textfeld variabler Länge, bis zu 2^{32}-1 (=2.294.967.295) Zeichen.	... noch mehr Text hier ...
TINYBLOB	Binäre Form von TINYTEXT.	
MEDIUMBLOB	Binäre Form von MEDIUMTEXT	
	BLOB	Binäre Form von TEXT
Datumstypen		
DATE	Datum in ISO-8601-Schreibweise (JJJJ-MM-TT)	2004-05-24
TIME	Uhrzeit gemäß ISO 8601 (hh:mm:ss)	07:45:00
DATETIME	Datum und Uhrzeit gemäß ISO 8601 (JJJJ-MM-TT hh:mm:ss)	2005-05-27 07:45:00
TIMESTAMP	Sekundengenauer Zeitpunkt zwischen 1970-01-01 und 2037-12-31	Anwenderdarstellung: 2004-05-24 13:36:14
YEAR	Vierstellige Jahreszahl	2004
Komplexe Datentypen		
ENUM	Aufzählungstyp mit bis zu 65.535 Elementen	
SET	Menge mit bis zu 64 Elementen	

Jede Spalte einer Relation muß mit genau einem Datentyp der oben dargestellten Liste versehen werden. Nachfolgend sind einige Definitionen und Besonderheiten zusammengestellt:

```
mysql> CREATE TABLE test(wenigText CHAR(300));
--zulässige Grenze für CHAR überschritten ...

mysql> DESCRIBE test;
+-----------+------+-------------------+------+-----+---------+-------+
| Field     | Type | Collation         | Null | Key | Default | Extra |
+-----------+------+-------------------+------+-----+---------+-------+
| wenigText | text | latin1_swedish_ci | YES  |     | NULL    |       |
+-----------+------+-------------------+------+-----+---------+-------+
```

Beispiel 26: Auswirkung von Datentypen III

Beispiel 26 zeigt die automatische Konversion des Datentypen CHAR in der Datenbank in TEXT sobald bereits bei der Anlage die zulässige Größenbegrenzung für CHAR überschritten wird. Werden zur Laufzeit Texte größer als die im CREATE TABLE-Ausdruck angegebene Maximalkapazität abgespeichert, so werden alle überzähligen Zeichen abgeschnitten.

```
mysql> CREATE TABLE test(ts timestamp, x VARCHAR(10));
Query OK, 0 rows affected (0.35 sec)

mysql> INSERT INTO test values(null, "abc");
Query OK, 1 row affected (0.06 sec)

mysql> INSERT INTO test values(null, "def");
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO test values(null, "abc");
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM test;
+---------------------+------+
| ts                  | x    |
+---------------------+------+
| 2003-05-26 23:25:51 | abc  |
| 2003-05-26 23:26:13 | def  |
| 2003-05-26 23:26:28 | abc  |
+---------------------+------+
3 rows in set (0.00 sec)

mysql> UPDATE test SET x="xyz" WHERE x="abc";
Query OK, 2 rows affected (0.05 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> SELECT * FROM test;
+---------------------+------+
| ts                  | x    |
+---------------------+------+
| 2003-05-26 23:27:10 | xyz  |
| 2003-05-26 23:26:13 | def  |
| 2003-05-26 23:27:10 | xyz  |
+---------------------+------+
3 rows in set (0.00 sec)
```

Beispiel 27: Auswirkung von Datentypen IV

Das Beispiel zeigt die Nutzung des Typs TIMESTAMP. Spalten dieses Typs werden automatisch durch das DMBS mit Werten versorgt werden. Daher ist die Belegung mit NULL bei Einfügung der drei Zeichenketten wirkungslos.
Überdies wird der Wert jeder TIMESTAMP-typisierten Spalte (im Beispiel: ts) bei jedem Schreibvorgang aktualisiert. Dies zeigt die nochmalige Ausgabe der Tabelleninhalte nach Aktualisierung der beiden Tupel, die für das Attribut x den Wert abc aufweisen.

```
mysql> create table Ampel(farbe enum('rot','gelb','gruen'));
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO Ampel VALUES('rot');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT into Ampel VALUES('blau');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM Ampel;
+-------+
| farbe |
+-------+
| rot   |
|       |
+-------+

mysql> INSERT INTO Ampel Values(2);
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM Ampel;
+-------+
| farbe |
+-------+
| rot   |
|       |
| gelb  |
+-------+
3 rows in set (0.00 sec)
```

Beispiel 28: Auswirkung von Datentypen V

Das Beispiel 28 zeigt die Nutzung eines Aufzählungstypen.
Er erlaubt ausschließlich das Einfügen der vordefinierten Werte und legt für alle ungültigen Belegungen (im Beispiel: `blau`) die leeren Zeichenkette ab.
Die Werte können dabei wie in der Aufzählung definiert oder durch ihre Indexposition (beginnend ab 1) gespeichert werden.

```
mysql> CREATE TABLE test(x SET("a","b","c","d"));
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO test VALUES("a");
Query OK, 1 row affected (0.07 sec)

mysql> INSERT INTO test values("a,b");
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO test values("a,c");
Query OK, 1 row affected (0.07 sec)

mysql> INSERT INTO test values("b,c");
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM test;
+------+
| x    |
+------+
| a    |
| a,b  |
| a,c  |
| b,c  |
+------+
4 rows in set (0.00 sec)

mysql> select * FROM test WHERE x & 1;
+------+
| x    |
+------+
| a    |
```

```
| a,b  |
| a,c  |
+------+
3 rows in set (0.00 sec)
--liefert alle Tupel, die das zweite Mengenelement (="a") enthalten
```

Beispiel 29: Auswirkung von Datentypen VI

Das Beispiel zeigt die Definition eines mengenwertigen Datentyps, d.h. einer Tabellenspalte, die mehr als einen Wert aufnehmen kann sowie die Abfragemöglichkeiten dafür.
Hinweis: Dieser Datentyp führt bereits in die NF^2-Datenstrukturen über und wird daher nicht im Rahmen des Entwurfsprozesses im konzeptuellen Schema verwendet.

Ergänzend zur Datentypangabe können für jede Spalte weitere einschränkende Angaben zur Spezifikation der erlaubten Werte getroffen werden.

`NOT NULL` legt hierbei fest, daß ein Tupel für eine Spalte zwingend einen von `NULL` verschiedenen Wert besitzen muß.

```
mysql> CREATE TABLE Person(Name VARCHAR(20), PersAuswNr INT NOT NULL);
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO Person VALUES('Max Obermüller', '123456789');
Query OK, 1 row affected (0.11 sec)

mysql> INSERT INTO Person VALUES('Xaver Hinterhuber', NULL);
ERROR 1048: Column 'PersAuswNr' cannot be null

mysql> select * from Person;
+----------------+------------+
| Name           | PersAuswNr |
+----------------+------------+
| Max Obermüller |  123456789 |
+----------------+------------+
1 row in set (0.00 sec)
```

Beispiel 30: Definition einer Spalte als NOT NULL

Das Beispiel zeigt eine Tabelle, bei der das Attribut `PersAuswNr` als `NOT NULL` definiert wurde. Einfüge- oder Aktualisierungsversuche, die zu Nullwerten dieses Attributs führen würden, werden durch das DMBS unterbunden.

Alternativ dazu gestattet die Angabe von `NULL` die Existenz von Nullwerten in der Tabelle. Diese Definition ist optional und wird bei fehlender Angabe automatisch als Vorgabe gesetzt.
Das Beispiel zeigt die Definition der Spalte `Autofahrer` als `NULL`, die neben den beiden vorgegebenen Werten auch keinen Wert enthalten darf.

```
mysql> CREATE TABLE Person(
    -> Name VARCHAR(20),
    -> PersAuswNr INT NOT NULL,
    -> Autofahrer ENUM('J','N') NULL);
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO Person VALUES('Max Obermüller', '123456789', NULL);
Query OK, 1 row affected (0.12 sec)

mysql> INSERT INTO Person VALUES('Xaver Hinterhuber', '234567891', 'J');
Query OK, 1 row affected (0.04 sec)

mysql> select * from Person;
+-------------------+------------+------------+
| Name              | PersAuswNr | Autofahrer |
+-------------------+------------+------------+
| Max Obermüller    |  123456789 | NULL       |
| Xaver Hinterhuber |  234567891 | J          |
+-------------------+------------+------------+
3 rows in set (0.01 sec)
```

Beispiel 31: Definition einer Spalte als NULL

Die Angabe der Klausel `DEFAULT VALUE` gestattet es einen Vorgabewert zu definieren, der gesetzt wird wenn kein Wert für eine Spalte angegeben wird.
Dies ersetzt jedoch nicht automatisch die Möglichkeit des Auftretens von Nullwerten innerhalb einer Spalte. Diese können auch weiterhin auftreten, sofern sie explizit eingefügt werden.
Die unterschiedlichen Wirkungsweisen zeigt Beispiel 32. Dort findet sich die Spalte `Autofahrer` (vorgabegemäß, da keine andere Angabe erfolgte) als nullwertfähig mit Vorgabewert `J`, sowie die Spalte `Hundebesitzer`, die mit Vorgabe `N` und als nicht nullwertfähig deklariert wurde.

```
mysql> CREATE TABLE Person(
    -> Name VARCHAR(20) NOT NULL,
    -> Autofahrer ENUM('J','N') DEFAULT 'J',
    -> Hundebesitzer ENUM('J','N') NOT NULL DEFAULT 'N'
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO Person VALUES('Xaver Obermüller', 'J', 'J');
Query OK, 1 row affected (0.10 sec)

mysql> INSERT INTO Person VALUES('Max Hinterhuber', NULL, 'N');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Person (Name) VALUES('Schorsch Huber');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM Person;
+------------------+------------+---------------+
| Name             | Autofahrer | Hundebesitzer |
+------------------+------------+---------------+
| Xaver Obermüller | J          | J             |
| Max Hinterhuber  | NULL       | N             |
| Schorsch Huber   | J          | N             |
+------------------+------------+---------------+
3 rows in set (0.00 sec)
```

Beispiel 32: Definition einer Spalte mit Vorgabewerten

Ist die Auszeichnung einer einzelnen Spalte als Primärschlüssel gewünscht, so kann der Definition `(PRIMARY) KEY` nachgestellt werden um dies zu erreichen.
Die Definition eines Primärschlüssels impliziert den Zwang in jedem Tupel einen von `NULL` verschiedenen eineindeutigen Wert dafür abspeichern zu müssen. Primärschlüsselspalten sind damit immer auch `NOT NULL`.
Zusätzlich kann für jede Tabelle höchstens ein Primärschlüsselattribut angegeben werden.
Hinweis: Aus mehr als einer Spalte zusammengesetzte Primärschlüssel können nicht durch diese Syntax gebildet werden.

```
mysql> CREATE TABLE Person(Name VARCHAR(20) PRIMARY KEY, Adresse VARCHAR(50));
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO Person VALUES('Xaver Obermüller', 'Dorfstr. 12');
Query OK, 1 row affected (0.10 sec)

mysql> INSERT INTO Person VALUES('Hans Hintermeier', 'Dorfstr. 13');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Person (Name) VALUES ('Schorsch Huber');
Query OK, 1 row affected (0.06 sec)

mysql> select * from Person;
+------------------+--------------+
| Name             | Adresse      |
+------------------+--------------+
| Hans Hintermeier | Dorfstr. 13  |
| Schorsch Huber   | NULL         |
| Xaver Obermüller | Dorfstr. 12  |
+------------------+--------------+
```

```
4 rows in set (0.00 sec)

mysql> INSERT INTO Person VALUES(NULL, 'Hauptstr. 11');
ERROR 1048: Column 'Name' cannot be null
```

Beispiel 33: Definition eines Primärschlüssels

Zur Erstellung eines zusammengesetzten Primärschlüssels kann nicht das nachgestellte Schlüsselwort `PRIMARY KEY` verwendet werden, da seine wiederholte Angabe mehrdeutig wäre. Für diesen Fall muß eine gesondert `PRIMARY KEY`-Definition in den `CREATE TABLE`-Ausdruck aufgenommen werden:

```
CREATE TABLE DEPT_LOCATIONS(
   DNUMBER INTEGER(1) NOT NULL,
   DLOCATION VARCHAR(20) NOT NULL,
   PRIMARY KEY (DNUMBER, DLOCATION));
```

Beispiel 34: Definition eines zusammengesetzten Primärschlüssels

Das Beispiel zeigt verschiedene Einfügeoperationen. Bemerkenswert ist die letzte, die das Primärschlüsselattribut leer läßt. Die hierbei erfolgende Belegung mit der leeren Zeichenkette (nicht `NULL`!) ist ein gültiger Eintrag im Sinne des angegebenen Datentypen `VARCHAR` und der Restriktion keine Belegung mit `NULL` vorzunehmen.

Soll ein nicht-sprechender Schlüssel (z.B. eine einfache Zählnummer) zur Identifikation genutzt werden, so kann diese durch das DBMS automatisiert bereitgestellt werden.

```
mysql> CREATE TABLE Person(
    -> LfdNr Int AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(20) NOT NULL);
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO Person VALUES(1,'Max Obermüller');
Query OK, 1 row affected (0.09 sec)

mysql> INSERT INTO Person VALUES(3,'Schorsch Hinterhuber');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Person VALUES(3,'Xaver Mayer');
ERROR 1062: Duplicate entry '3' for key 1

mysql> SELECT * FROM Person;
+-------+----------------------+
| LfdNr | Name                 |
+-------+----------------------+
|     1 | Max Obermüller       |
|     3 | Schorsch Hinterhuber |
+-------+----------------------+
2 rows in set (0.00 sec)

mysql> INSERT INTO Person (Name) VALUES('Xaver Mayer');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Person (Name) VALUES('Hans Huber');
Query OK, 1 row affected (0.04 sec)

mysql> select * from Person;
+-------+----------------------+
| LfdNr | Name                 |
+-------+----------------------+
|     1 | Max Obermüller       |
|     2 | Xaver Mayer          |
|     3 | Schorsch Hinterhuber |
|     4 | Hans Huber           |
+-------+----------------------+
4 rows in set (0.00 sec)
```

Beispiel 35: Definition eines automatisch befüllten Primärschlüssels

Im Beispiel 35 wird der Wert der Spalte LfdNr, sofern nicht durch den Anwender explizit angegeben, automatisch ermittelt und eingefügt.

Zur Zugriffsbeschleunigung dienende Indexstrukturen können bereits zum Tabellenerstellungszeitpunkt durch den Anwender angegeben werden. Diese werden jedoch nicht der Spaltendefinition als nachgestellt, sondern bilden einen eigenen Eintrag innerhalb der Tabellendefinition.
Ein Index kann gleichzeitig eine oder mehrere Spalten umfassen. Beispiel Beispiel 36 zeigt dies:

```
mysql> CREATE TABLE Person(
    -> Name VARCHAR(20) PRIMARY KEY,
    -> GebDat DATE, Str_HsNr VARCHAR(20),
    -> PLZ CHAR(5),
    -> Ort VARCHAR(50),
    -> INDEX GebDatIdx (GebDat),
    -> INDEX AdresseIndex (Str_HsNr, PLZ, Ort));
Query OK, 0 rows affected (0.07 sec)
```

Beispiel 36: Definition von Indexen

Als beschränkende Verschärfung, die sich auch positiv auf die Zugriffsgeschwindigkeit auswirkt, kann ein Index als UNIQUE INDEX definiert werden. Er darf dann ausschließlich eindeutige Werte oder Wertkombinationen aufnehmen.

Erzeugung von Fremdschlüsselbeziehungen

Die Erzeugung von Fremdschlüsselbeziehungen ist das integrale Element zur Wahrung der referentiellen Integrität. Fremdschlüsselbeziehungen können bereits zum Erstellungszeitpunkt einer Tabelle angegeben werden wie Beispiel 37 zeigt oder nachträglich durch einen ALTER TABLE-Ausdruck hinzugefügt werden wie durch Beispiel 38 gezeigt.

```
CREATE TABLE DEPARTMENT(
   DNAME VARCHAR(20) NOT NULL,
   DNUMBER INTEGER(1) PRIMARY KEY,
   MGRSSN INTEGER(9),
   MGRSTARTDATE DATE);

CREATE TABLE EMPLOYEE(
   FNAME VARCHAR(10) NOT NULL,
   MINIT VARCHAR(1),
   LNAME VARCHAR(10) NOT NULL,
   SSN INTEGER(9) PRIMARY KEY,
   BDATE DATE,
   ADDRESS VARCHAR(30),
   SEX ENUM('M','F'),
   SALARY REAL(7,2) UNSIGNED,
   SUPERSSN INTEGER(9),
   DNO INTEGER(1) REFERENCES DEPARTMENT(DNUMBER),
   INDEX DNO_IDX (DNO));
```

Beispiel 37: Erzeugung von Fremdschlüsselbeziehungen zum Tabellenersellungszeitpunkt

Das Beispiel erzeugt neben der angestrebten Fremdschlüsselbeziehungen zwischen dem Attribut DNO der Tabelle EMPLOYEE und dem Primärschlüssel DNUMBER in DEPARTMENT einen Index auf den Fremdschlüssel innerhalb der Tabelle EMPLOYEE.
Dies ist für einige Datenbankmanagementsysteme (darunter MySQL) notwendig, um die Zugriffe auf den Fremdschlüssel zu beschleunigen.

Das nachfolgende Beispiel liefert dasselbe Ergebnis, jedoch unter nachträglicher (d.h. nach dem Erstellungszeitpunkt der Tabellen) Fremdschlüsselerzeugung:

```
CREATE TABLE DEPARTMENT(
   DNAME VARCHAR(20) NOT NULL,
   DNUMBER INTEGER(1) PRIMARY KEY,
   MGRSSN INTEGER(9),
   MGRSTARTDATE DATE);

CREATE TABLE EMPLOYEE(
   FNAME VARCHAR(10) NOT NULL,
   MINIT VARCHAR(1),
   LNAME VARCHAR(10) NOT NULL,
   SSN INTEGER(9) PRIMARY KEY,
   BDATE DATE,
   ADDRESS VARCHAR(30),
   SEX ENUM('M','F'),
   SALARY REAL(7,2) UNSIGNED,
   SUPERSSN INTEGER(9),
   DNO INTEGER(1));

ALTER TABLE EMPLOYEE ADD INDEX DNO_IDX (DNO);
ALTER TABLE EMPLOYEE ADD CONSTRAINT DNO_FK FOREIGN KEY (DNO) REFERENCES DEPARTMENT
(DNUMBER);
```

Beispiel 38: Nachträgliche Erzeugung von Fremdschlüsselbeziehungen

3.3 Der Anfrageteil von SQL

Anfragen zur Ermittlung von Datenbankinhalten stellen den eigentlichen Sprachkern von SQL und zweifellos den in der Praxis bedeutsamsten Anteil der Sprache dar.
Die gesamte Mächtigkeit des Anfrageteils von SQL erschließt sich durch das Schlüsselwort SELECT. Es gestattet Anfragen theoretisch unbegrenzter Komplexität in einer uniformen und leicht zu behaltenden Syntax zu formulieren, deren Mächtigkeit von einfachsten Anfragen bis zu aufwendigen Auswertungen reicht.

Anfragen von Datenbankinhalten

Die SQL-Anweisung SELECT dient der Abfrage von in einer Datenbank abgelegten Inhalten. Sie benötigt Wissen über die angelegten Tabellen sowie deren Struktur hinsichtlich Spalten und deren Typen.
Alle nachfolgenden Beispielanfragen beziehen sich, sofern nicht anders angegeben auf die Demodatenbank.

Die vereinfachte Syntax der SELECT-Anweisung lautet:

```
SELECT [ALL|DISTINCT] select_item,...
FROM table_specification,...
[WHERE search_condition]
[GROUP BY grouping_column,...]
[HAVING search_condition]
[ORDER BY sort_specification,...]
```

```
SELECT FNAME
FROM EMPLOYEE;
```

Beispiel 39: Einfache Anfrage

Die Anfrage liefert die Inhalte der Spalte FNAME aller in der Tabelle EMPLOYEE abgelegten Tupel. Die Werte werden in keiner vorgegebenen Reihenfolge ausgegeben, d.h. eine etwaige Sortierung ist zufallsbedingt und kann durch DBMS interne Reorganisationsprozesse zerstört werden.

Durch diese Anfrage wird als Resultat eine nicht in der Datenbank abgelegte Tabelle erzeugt, welche nur die im SELECT-Ausdruck angegebenen Spalten enthält. Die Ergebnistabelle stimmt zwar in Tupelanzahl mit der Ursprungstabelle überein, blendet jedoch einzelne Attribute aus. Diese Vorgang wird als *Projektion* bezeichnet.

Definition 42: *Projektion*
Die Projektion blendet einzelne Spalten aus.

```
SELECT FNAME, MINIT, LNAME, SSN, BDATE, ADDRESS, SEX, SALARY, SUPERSSN, DNO
FROM EMPLOYEE;
```

Beispiel 40: Anfrage aller Spalten einer Tabelle

Die Abfrage aus Beispiel 40 liefert die Wertinhalte aller Spalten (sie sind explizit nach dem Schlüsselwort SELECT angegeben) der Relation EMPLOYEE.
Als Besonderheit wird für das Attribut SUPERSSN des Mitarbeiters James E. Borg der Wert NULL ausgegeben. Diese Zeichenkette gibt an, daß für dieses Attribut der Relation kein Wert abgespeichert wurde.

Die schreibaufwendige und damit fehlerträchtige Explizierung aller Spalten einer Relation ist kaum praktikabel und überdies äußerst änderungssensitiv im Falle der Aufnahme neuer Spalten oder der Lösung Existierender.
Aus diesem Grunde kann statt des Spaltennamens ein Stern „*“ als Jokerzeichen stellvertretend für alle Spalten einer Relation angegeben werden.
Beispiel 41 zeigt dies als Umschreibung der Anfrage aus Beispiel 40:

```
SELECT *
FROM EMPLOYEE;
```

Beispiel 41: Anfrage aller Spalten einer Tabelle mit Jokerzeichen

Enthält die Ausgabe nicht den Primärschlüssel einer Tabelle, so kann es vorkommen, das mehrfach dieselben Werte ausgegeben werden. Dies kann durch Angabe des Schlüsselwortes DISTINCT in der SELECT-Klausel vermieden werden.
DISTINCT überschreibt das vorgegebene Verhalten (ALL) alle Einträge auszugeben.

```
SELECT DISTINCT SALARY
FROM EMPLOYEE;
```

Beispiel 42: Duplikatfreie Ausgabe aller verschiedenen Werte

Beispiel 42 liefert alle verschiedenen Werteinträge der Spalte SALARY duplikatfrei.

Durch Angabe mehrerer Einträge in der FROM-Klausel können Inhalte aus verschiedenen Tabellen innerhalb einer Anfrage extrahiert werden:

```
SELECT DNAME, PNUMBER
FROM DEPARTMENT, PROJECT;
```

Beispiel 43: Anfrage auf zwei Tabellen

Die Anfrage bildet das kartesische Produkt der beiden angefragten Tabellen.

Aliasbildung

Bei Anfragen über mehrere Tabellen kann es zu Problemen hinsichtlich der Eindeutigkeit der Spaltenbezeichner kommen. So würde die Anfrage aus Beispiel 44 nicht die für ESSN abgelegten Werte liefern, sondern den Fehler ERROR 1052: Column: 'ESSN' in field list is ambiguous, da in jeder der beiden in der Anfrage berücksichtigten Tabellen eine Spalte mit ESSN benannt ist.

```
SELECT ESSN, ESSN
FROM WORKS_ON, DEPENDENT;
```

Beispiel 44: Fehlerhafte Anfrage auf zwei Tabellen

Als Lösung bietet SQL die Möglichkeit den Spaltennamen zusätzlich durch Voranstellung des Namens der die Spalte beherbergenden Tabelle zu qualifizieren um die erforderliche Eindeutigkeit herzustellen:

```
SELECT WORKS_ON.ESSN, DEPENDENT.ESSN
FROM WORKS_ON, DEPENDENT;
```

Beispiel 45: Lösung des Mehrdeutigkeitsproblems bei Anfrage auf zwei Tabellen

Unter Nutzung der Möglichkeit Alternativnamen für Tabellen, sog. *Aliasnamen*, anzugeben ergibt sich eine in der Schreibung kompaktere Umsetzung:

```
SELECT w.ESSN, d.ESSN
FROM WORKS_ON AS w, DEPENDENT AS d;
```

Beispiel 46: Lösung des Mehrdeutigkeitsproblems bei Anfrage auf zwei Tabellen

Gleichzeitig kann die Aliasbildung eingesetzt werden, um die Benennung der Spalten bei der Ausgabe zu modifizieren. Auf diesem Wege können wenig sprechende Namen oder Doppelbenennungen umgangen werden.

```
SELECT w.ESSN AS "Mitarbeiter Sozialversicherungsnummer",
d.ESSN AS "Sozialversicherungsnummer des Verwandten"
FROM WORKS_ON AS w, DEPENDENT AS d;
```

Beispiel 47: Umbenennung von Ausgabespalten

Berechnete Ausgaben

Durch die Angabe einfacher arithmetischer Formeln in der SELECT-Klausel können vor ihrer Ausgabe Berechnungen auf den Werten aus der Datenbank angestellt werden.

```
SELECT FNAME, SALARY*12 as Jahreseinkommen
FROM EMPLOYEE;
```

Beispiel 48: Berechnungen I

Beschränkung der Ergebnismenge

Die bisher betrachteten Anfrageformen lieferten immer die gesamten Inhalte der betrachteten Tabellen. Durch Angabe einer einschränkenden Bedingung innerhalb der WHERE-Klausel einer SELECT-Anweisung können die Tabelleninhalte hinsichtlich einer Bedingung gefiltert werden.

Beispiel 49 liefert nur die abgespeicherten Werte für Geburtsdatum (BDATE) und Adresse (ADDRESS) derjenigen Mitarbeiter, deren Vornamen (FNAME) *John* ist.

```
SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE FNAME="John";
```

Beispiel 49: Einschränkung der Anfrage

Durch die Filterung der Ergebnistupel werden alle diejenigen Datenbankeinträge, welche die getroffene Bedingung nicht erfüllen ausgeblendet. Das Ergebnis kann (sofern SELECT * ... gewählt wurde) zwar in der Anzahl Spalten mit der Ursprungsrelation übereinstimmen, wird dies jedoch typischerweise (d.h. außer im Falle, daß alle Tupel der Relation die formulierte Bedingung erfüllen) nicht tun.
Dieser Vorgang wird als *Selektion* bezeichnet.

Definition 43: *Selektion*
Die Selektion blendet einzelne Werteinträge aus..

Als relationale Operatoren für Vergleichstests zwischen zwei (möglicherweise einelementigen) Mengen stehen zur Verfügung:

Operator	Funktion	Bemerkung
=	Gleichheitstest	
<>	Test auf Ungleichheit	Teilweise (auch in MySQL!) ist der Standardoperator durch != ersetzt
<	Test auf kleiner	Liefert bei Zeichenkettendatentypen alle lexikalisch „kleineren“, d.h. diejenigen die in der alphabetischen Sortierung früher auftreten. Ebenso liefert der Operator bei der Anwendung auf Datumsdatentypen die kalendarisch früheren.
<=	Kleiner oder gleich	
>	Größer	
>=	Größer oder gleich	
IS NULL	Testet ob eine Spalte NULL enthält	
IS NOT NULL	Testet ob eine Spalte nicht NULL enthält	
BETWEEN	Testet ob ein Wert in vorgegebenen Grenzen liegt	
IN	Testet ob ein Wert innerhalb einer vorgegebenen Menge liegt	

Neben den einfachen Vergleichsoperationen können durch den LIKE-Operator unscharfe musterbasierte Suchen ausgedrückt werden.
Die Musterausdrücke werden dabei aus den tatsächlich in der Ergebnisemenge erwarteten Zeichen ergänzt um Metazeichen mit besonderer Bedeutung zusammengesetzt. Hierbei stehen „%“ zur Stellvertretung einer (möglicherweise leeren) Menge beliebiger Zeichen und „_“ zur Stellvertretung genau eines Zeichens zur Verfügung.

```
SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE FNAME LIKE "J%";
```

Beispiel 50: Musterbasierte Anfrage I

Die Anfrage aus Beispiel 50 liefert die Werte der Spalten BDATE und ADDRESS aller Mitarbeiter deren Name (FNAME) mit einem „J“ beginnt.
Die Anfrage aus Beispiel 51 beschränkt die Suche zusätzlich auf diejenigen Namen, deren vorletztes Zeichen ein „e“ ist, d.h. diejenigen Einträge für die nach dem „e“ nur noch genau ein beliebiges Zeichen auftritt.

```
SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE FNAME LIKE "J%e_";
```

Beispiel 51: Musterbasierte Anfrage II

Die Variante aus Beispiel 52 extrahiert alle Spalteninhalte der Mitarbeitertabelle deren Name aus genau fünf Zeichen besteht.

```
SELECT *
FROM EMPLOYEE
WHERE FNAME LIKE "____";
```

Beispiel 52: Musterbasierte Anfrage III

Kombination von Einzelbedingungen

Zur Selektion nach mehreren Bedingungen können diese mit den logischen Operationen AND, OR und NOT kombiniert werden.

Die Anfrage aus Beispiel 53 liefert die Namen aller Mitarbeiter, die in „Houston“ wohnen und weniger als 50000 verdienen.

```
SELECT FNAME
FROM EMPLOYEE
WHERE ADDRESS LIKE "%Houston%" AND SALARY < 50000;
```

Beispiel 53: Kombination von Bedingungen

Mittels der Verknüpfungsoperatoren können auch einige der zuvor gezeigten Vergleichsoperatoren ausgedrückt werden.

Vergleichsoperator	Alternative Schreibweise mit Bedingungsverknüpfung
a BETWEEN b and c	(a >= b) AND (a <= c)
x IN (a, b, c)	(x = a) OR (x = b) OR (x = c)

Für die Kombinationsoperatoren gilt, aufgrund der Möglichkeit des Auftretens von NULL-Werten, die dreiwertige Logik:

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Kombination von Anfrageergebnissen

In manchen Fällen ist es gewünscht das Ergebnis eigenständiger Anfragen zu einem Ergebnis zu kombinieren. Hierzu kann das UNION-Schlüsselwort zur Verbindung der Einzelanfragen.

```
CREATE TABLE Artikel(
        ArtNo           VARCHAR(4) PRIMARY KEY,
        Bezeichnung VARCHAR(10) NOT NULL,
        Preis                   DECIMAL(5,2) NOT NULL);
CREATE TABLE Sonderpreise(
        ArtikelNo       VARCHAR(4) PRIMARY KEY,
        Bezeichnung     VARCHAR(10) NOT NULL,
        Sonderverkaufsgrund VARCHAR(20));

INSERT INTO Artikel VALUES("2222", "Blitz Superrein", 19.99);
INSERT INTO Artikel VALUES("1111", "Wusch Superfein", 99.95);

INSERT INTO Sonderpreise VALUES("4444", "Kratzweich", "Wasserschaden");
INSERT INTO Sonderpreise VALUES("3333", "Glanz Extraweiss", NULL);

SELECT ArtNo
FROM Artikel
UNION
SELECT ArtikelNo
FROM Sonderpreise;
```

Beispiel 54: Kombination mittels UNION

Die Anfrage aus Beispiel 54 erstellt zunächst zwei Tabellen und fügt einige Daten ein. Anschließend werden die Artikelnummern (Spalte ArtNo bzw. ArtikelNo) beider Tabellen angefragt und das Ergebnis mittels UNION zu einer Resultattabelle kombiniert.

Voraussetzung der Kombinierbarkeit ist die Typgleichheit der selektierten und zu vereinigenden

Attribute (im Beispiel beide vom Typ VARCHAR(4)).

Implizit führt die Verwendung von UNION sowohl die Sortierung der Ergebnissmenge als auch die Duplikatentfernung daraus herbei.

Verbünde

Häufig besteht der Wunsch Werte aus verschiedenen Tabellen nicht nur gemeinsam abzufragen und anzuzeigen, sondern auch inhaltlich in Beziehung zu setzen.

Die Anfrage aus Beispiel 55 versucht durch Abfrage der Tabellen EMPLOYEE und DEPARTMENT die Namen der Mitarbeiter und die (in der anderen Tabelle abgelegte) Bezeichnung Abteilung zu ermitteln die sie beschäftigen.
Aufgrund der Bildung des kartesischen Produkts werden jedoch alle theoretisch möglichen Kombinationen geliefert und nicht die Untermenge der tatsächlich existierenden Paarungen.

```
SELECT FNAME, DNAME
FROM DEPARTMENT, EMPLOYEE;
```

Beispiel 55: Fehlerhafte Verbundbildung

Durch (geschickte) Nutzung der WHERE-Bedingung, die Werte aus beiden Tabellen miteinander in Beziehung setzt, gelingt jedoch die gewünschte Ermittlung:

```
SELECT FNAME, DNAME
FROM DEPARTMENT AS d, EMPLOYEE as e
WHERE d.DNUMBER = e.DNO;
```

Beispiel 56: Innerer Verbund

Das Beispiel 56 liefert lediglich diejenigen Tupel, für die das in EMPLOYEE abgespeicherte DNO-Attribut einen Wert enthält, der auch in der Spalte DNUMBER der Tabelle DEPARTMENT auftritt.

Definition 44: *Innerer Verbund*
Ein Innerer Verbund enthält die selektierten Daten aller beteiligten Tabellen, welche die formulierte Einschränkungsbedingung erfüllen.

Der SQL-Standard gibt für diese besondere Anfrageform eine eigene Syntax vor:

```
SELECT FNAME, DNAME
FROM DEPARTMENT AS d INNER JOIN EMPLOYEE AS e
ON d.DNUMBER = e.DNO;
```

Beispiel 57: Innerer Verbund in Standardnotation

Die Bildung von Verbünden ist nicht auf die Angabe verschiedener Tabellen beschränkt, sondern kann auch durch mehrfache Bezugnahme auf dieselbe Tabelle geschehen:

```
SELECT e1.FNAME as Chef, e2.FNAME as Mitarbeiter
FROM EMPLOYEE AS e1, EMPLOYEE AS e2
WHERE e1.SSN = e2.SUPERSSN;
```

Beispiel 58: Innerer Verbund unter mehrfacher Nutzung derselben Tabelle

Beispiel 59 zeigt ein Beispiel der Bildung eines inneren Verbundes unter Zugriff auf drei Tabellen. Die Anfrage liefert die Familiennamen (Tabelle EMPLOYEE) sowie die Abteilungen denen der Mitarbeiter zugeordnet ist (aus Tabelle DEPARTMENT) sowie die durch den Mitarbeiter bearbeiteten Projekte (Tabelle PROJECT).
Die Tabelle PROJECT kann jedoch nicht direkt in den Verbund einbezogen werden, da sie über keine geeigneten Attribute (d.h. Attribute die mit derselben Semantik in einer der beiden anderen Tabellen auftreten) verfügt. Daher wird zusätzlich die Tabelle WORKS_ON in die Anfrage miteinbezogen, weil sie mit dem Attribut ESSN ein Attribut bietet, welches die in EMPLOYEE enthaltene Attribut SSN als Fremdschlüssel beinhaltet. Ausgehend hiervon kann eine Bedingung

unter Einbezug von PROJECT formuliert werden.

```
SELECT FNAME, DNAME, PNAME
FROM EMPLOYEE AS e, DEPARTMENT AS d, PROJECT AS p, WORKS_ON AS w
WHERE d.DNUMBER = e.DNO AND e.SSN = w.ESSN AND w.PNO = p.PNUMBER;
```

Beispiel 59: Innerer Verbund dreier Tabellen

Für Verbünde ist die Bildung durch ausschließliche Nutzung des Gleichheitsoperators innerhalb der WHERE-Klausel keineswegs zwingend, wenngleich diese sog. *Equi Joins* eine häufige Anwendungsform darstellen.
Beispiel 1 zeigt einen durch Nutzung des kleiner-Operators gebildeten Verbund, der alle Abteilungen enthält, in denen ein Mitarbeiter (noch) nicht arbeitet und deren Abteilungsnummer größer ist als die Nummer der Abteilung welcher der Mitarbeiter gegenwärtig zugeordnet ist. (Mögliche semantische Deutung: Liste möglicher Beförderungen, sofern größere Abteilungsnummern einen Aufstieg codieren.

```
SELECT FNAME, DNAME
FROM EMPLOYEE JOIN DEPARTMENT
ON DEPARTMENT.DNUMBER < EMPLOYEE.DNO;
```

Beispiel 60: Non-Equi-Join

Äußere Verbunde

Neben der Möglichkeit durch innere Verbünde Tupel die über Attribute mit übereinstimmenden Wertbelegungen zu selektieren besteht durch *äußere Verbünde* die Möglichkeit neben den Tupeln mit übereinstimmenden Werten alle Tupel einer am Verbund beteiligten Tabelle vollständig zu selektieren.

Definition 45: *Äußerer Verbund*
Ein Äußerer Verbund enthält die selektierten Daten aller beteiligten Tabellen, welche die formulierte Einschränkungsbedingung erfüllen, sowie alle Daten der „äußeren“ Tabelle.
Die nicht mit Werten belegbaren Felder werden durch NULL aufgefüllt.

Konzeptionell wird zwischen *linken* und *rechten Äußeren Verbünden* unterschieden. Die „Seite“ des Verbundes gibt diejenige beteiligte Tabelle an, die im Rahmen der Verbundbildung vollständig ausgegeben wird.
Beispiel 61 zeigt ein Beispiel eines linken Äußeren Verbundes, Beispiel 62 illustriert einen rechten äußeren Verbund.

```
INSERT INTO EMPLOYEE VALUES("John", "X", "Doe", "999999999", "1965-03-04", "42 XYZ
Street", "M", 50000, NULL, NULL);

SELECT FNAME, DNAME
FROM EMPLOYEE LEFT OUTER JOIN DEPARTMENT
ON DEPARTMENT.DNUMBER = EMPLOYEE.DNO;
```

Beispiel 61: Linker Äußerer Verbund

Das Beispiel fügt zunächst einen Tupel zur Tabelle EMPLOYEE hinzu, der keiner Abteilung zugeordnet ist. In einem Inneren Verbund erscheint dieser Tupel daher nicht. Der linke Äußere Verbund des Beispiels hingegen umfaßt alle Tupel aus *EMPLOYEE* sowie die Werte der hinsichtlich der Bedingung DEPARTMENT.DNUMBER = EMPLOYEE.DNO ermittelten Übereinstimmungen in DEPARTMENT.
Für die nicht ermittelbaren Übereinstimmungen werden NULL-Werte erzeugt.

```
INSERT INTO DEPARTMENT VALUES("New Dept.", 0, 888665555, NULL);

SELECT FNAME, DNAME
FROM EMPLOYEE RIGHT OUTER JOIN DEPARTMENT
ON DEPARTMENT.DNUMBER = EMPLOYEE.DNO;
```

Beispiel 62: Rechter Äußerer Verbund

Auch das Beispiel 62 zum rechten Äußeren Verbund fügt zunächst einen Datensatz ein; diesmal in die Tabelle DEPARTMENT, der zu keinem Tupel in EMPLOYEE in Beziehung steht. Analog dem linken Äußeren Verbund liefert der rechte Äußere Verbund alle Tupel der rechtsstehenden Tabelle (DEPARTMENT) sowie die mit *EMPLOYEE* übereinstimmenden.

Kreuzverbund

Der Kreuzverbund liefert alle gemäß den Gesetzen des kartesischen Produkts bildbaren Kombinationen aus Tupeln der beitragenden Relationen:

```
SELECT DNAME, PNUMBER
FROM DEPARTMENT CROSS JOIN PROJECT;
```

Beispiel 63: Kreuzverbund

Das Beispiel entspricht damit im Ergebnis der Anfrage aus Beispiel 43.

Wird beim Kreuzverbund eine Bedingung angegeben, so entspricht er dem Inneren Verbund. Das Ergebnis der Anfrage aus Beispiel 64 ist daher identisch zum inneren Verbund aus Beispiel 56.

```
SELECT FNAME, DNAME
FROM DEPARTMENT CROSS JOIN EMPLOYEE
WHERE DEPARTMENT.DNUMBER = EMPLOYEE.DNO;
```

Beispiel 64: Kreuzverbund mit Bedingung

Artikel von Satya Komatineni: The Effective Use of Joins in Select Statements

Sortierungen

Zur Sortierung hinsichtlich einer oder mehrerer Spalten der als Anfrageergebnis ermittelten Tabelle steht die ORDER BY-Klausel zur Verfügung.
Beispiel 65 zeigt die Anwendung zur lexikalischen Sortierung:

```
CREATE TABLE Person(
        Vorname VARCHAR(10),
        Nachname VARCHAR(10));

INSERT INTO Person VALUES("Adam", "C-Mann");
INSERT INTO Person VALUES("Cesar", "C-Mann");
INSERT INTO Person VALUES("Berta", "C-Mann");
INSERT INTO Person VALUES("Adam", "A-Mann");
INSERT INTO Person VALUES("Cesar", "A-Mann");
INSERT INTO Person VALUES("Berta", "A-Mann");
INSERT INTO Person VALUES("Adam", "B-Mann");
INSERT INTO Person VALUES("Cesar", "B-Mann");
INSERT INTO Person VALUES("Berta", "B-Mann");

SELECT *
FROM Person
ORDER BY Nachname;
```

Beispiel 65: Sortierung

Ist die Sortierung bezüglich mehrerer Attribute, d.h. Sortierung innerhalb eines gleicher Attributwerte hinsichtlich eines anderen Attributs, gewünscht, so können auch mehrere Sortierattribute in der ORDER BY-Klausel versammelt werden.
Zusätzlich zeigt das Beispiel die Kurzschreibweise, welche die zu sortierenden Attribute nicht namentlich benennt, sondern nur hinsichtlich ihrer Position innerhalb der SELECT-Klausel referenziert.

```
CREATE TABLE Person(
        Vorname VARCHAR(10),
        Nachname VARCHAR(10));

INSERT INTO Person VALUES("Adam", "C-Mann");
INSERT INTO Person VALUES("Cesar", "C-Mann");
INSERT INTO Person VALUES("Berta", "C-Mann");
INSERT INTO Person VALUES("Adam", "A-Mann");
INSERT INTO Person VALUES("Cesar", "A-Mann");
INSERT INTO Person VALUES("Berta", "A-Mann");
INSERT INTO Person VALUES("Adam", "B-Mann");
INSERT INTO Person VALUES("Cesar", "B-Mann");
INSERT INTO Person VALUES("Berta", "B-Mann");

SELECT *
FROM Person
ORDER BY 2,1;
```

Beispiel 66: Sortierung bezüglich mehrerer Attribute

Vorgabegemäß erfolgt die Sortierung aufsteigend (*ascending*). Die Umkehrung der Sortierreihenfolge kann durch nachstellen der Zeichenfolge DESC (für *descending*) nach dem Namen des Sortierattributes erreicht werden.
Die aufsteigende Vorgabesortierung (ASC) wird üblicherweise nicht ausgeschrieben, ist aber im Beispiel 67 zur besseren Verdeutlichung expliziert.

```
CREATE TABLE Person(
        Vorname VARCHAR(10),
        Nachname VARCHAR(10));

INSERT INTO Person VALUES("Adam", "C-Mann");
INSERT INTO Person VALUES("Cesar", "C-Mann");
INSERT INTO Person VALUES("Berta", "C-Mann");
INSERT INTO Person VALUES("Adam", "A-Mann");
INSERT INTO Person VALUES("Cesar", "A-Mann");
INSERT INTO Person VALUES("Berta", "A-Mann");
INSERT INTO Person VALUES("Adam", "B-Mann");
INSERT INTO Person VALUES("Cesar", "B-Mann");
INSERT INTO Person VALUES("Berta", "B-Mann");

SELECT *
FROM Person
ORDER BY 2 ASC,1 DESC;
```

Beispiel 67: Auf- und Absteigende Sortierung bezüglich mehrerer Attribute

Unteranfragen

Bisher wurden Anfragen lediglich auf Tabellen in ihrer Rolle als in der Datenbank abgelegte Eingabemengen betrachtet. Die relationale Sichtweise erfordert jedoch keineswegs, daß die Eingangswerte einer Anfrage direkt aus der Datenbank gelesen werden müssen. Sie können auch Ergebnis einer weiteren Anfrage sein.

Anfragen die vor einer anderen Anfrage ausgeführt werden müssen um für diese Eingangswerte zu liefern werden daher als *Unterabfragen* (*subqueries* oder *nested queries*) bezeichnet.

Das Beispiel 68 zeigt eine solche Unteranfrage die alle Projektnummern liefert welche Projekten zugeordnet sind die in der durch *Smith* gleiteten Abteilung bearbeitet werden. Eine zweite Unterabfrage des Beispiels liefert alle Nummern von Projekten an denen dieser Mitarbeiter selbst arbeitet.
Die durch diese Abfrage gelieferten Daten (Projektnummern) sind Eingangsdaten in die Ermittlung der Projektnamen.

```
SELECT DISTINCT PNAME
FROM PROJECT
WHERE PNUMBER IN (
   SELECT PNUMBER
   FROM PROJECT AS p, DEPARTMENT AS d, EMPLOYEE AS e
   WHERE e.SSN = d.MGRSSN AND
   d.DNUMBER = p.DNUM AND
   e.LNAME="Smith")
      OR
         PNUMBER IN (SELECT PNO
            FROM WORKS_ON AS w, EMPLOYEE AS e
            WHERE w.ESSN = e.SSN AND
            e.LNAME="Smith");
```

Beispiel 68: Unterabfrage I

Beispiel 69 zeigt den Vergleich eines Einzelwertes (SALARY) mit einer Menge gelieferter Werte. Die Anfrage ermittelt diejenigen Mitarbeiter, deren Einkommen höher liegt als das Einkommen aller Mitarbeit in Abteilung Nummer 5. (Hinweis es wird nicht ermittelt ob das Einkommen größer ist als die Summe aller Einkommen der Mitarbeiter aus Abteilung 5, sondern nur ob das Einkommen größer ist als jedes Einzeleinkommen eines Mitarbeiters aus Abteilung 5.)

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SALARY > ALL (SELECT SALARY FROM EMPLOYEE WHERE DNO=5);
```

Beispiel 69: Unterabfrage II

Korrelierte Unteranfragen

Eine besondere Form der Unteranfragen stellen solche dar, die sich in ihrer WHERE-Klausel auf die äußere Anfrage beziehen.
Diese Form der Anfrageschachtelung wird auch als *korrelierte Unteranfrage* bezeichnet.

Das Beispiel 70 zeigt eine solche Anfrage, die alle Verwandten (DEPENDENT) ermittelt, die das selbe Geschlecht haben wie der in der Tabelle EMPLOYEE erfaßte Mitarbeiter.

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE AS e
WHERE e.SSN IN (SELECT ESSN
                                        FROM DEPENDENT
                                        WHERE e.SEX = SEX);
```

Beispiel 70: Korrelierte Unterabfrage

Jede korrelierte Unterabfrage kann durch Umschreibung in eine nicht-korrelierte Fassung überführt werden. So lautet die Formulierung des aus Beispiel 70 ohne geschachtelte Unterabfrage:

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE AS e, DEPENDENT AS d
WHERE e.SSN = d.ESSN AND
e.SEX = d.SEX;
```

Beispiel 71: Auflösung der korrelierten Unterabfrage

Die Formulierung als geschachtelte Unterabfrage ist damit nicht zwingend notwendig, kann jedoch aus Gründen der Übersichtlichkeit gewünscht sein.

Die nähere Betrachtung der Anfragen aus Beispiel 70 und Beispiel 71 zeigen, daß die aus der Tabelle DEPENDENT angefragten Daten lediglich zur Formulierung der Bedingung, nicht jedoch zur Ausgabe herangezogen werden. Daher läßt sich die Bedingung unter Verwendung des EXISTS-Operators umschreiben zu:

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE AS e
WHERE EXISTS (  SELECT *
                FROM DEPENDENT
                WHERE e.SSN = ESSN AND
                e.SEX = SEX);
```

Beispiel 72: Korrelierte Unterabfrage mit EXISTS

EXISTS liefert den Boole'schen Wahrheitswert immer dann, wenn die (Unter-)Abfrage eine nichtleere Menge ist, d.h. Daten enthält.

Anfragen die EXISTS oder IN beinhalten können auch durch linke Äußere Verbünde ausgedrückt werden, wie Beispiel 73 zeigt:

```
SELECT e.FNAME, e.LNAME
FROM EMPLOYEE AS e LEFT JOIN DEPENDENT AS d
ON e.SSN = d.ESSN AND e.SEX = d.SEX
WHERE d.SEX IS NOT NULL;
```

Beispiel 73: Korrelierte Unterabfrage ausgedrückt als linker äußerer Verbund

Eine ähnliche Funktion wie die EXISTS-Operation stellt ANY bereit, jedoch liefert diese die durch die Unterabfrage angefragten Tupel zurück um sie an eine Bedingung zu knüpfen.
Beispiel 74 zeigt die Ermittlung der Namen derjenigen Mitarbeiter, die mehr als irgendein beliebiger Manager verdienen.

```
SELECT FNAME
FROM EMPLOYEE
WHERE SALARY > ANY (SELECT SALARY
   FROM EMPLOYEE
   WHERE SSN IN (SELECT SUPERSSN
      FROM EMPLOYEE));
```

Beispiel 74: Unterabfrage unter Verwendung von ANY

Aggregatfunktionen und Gruppierung

Über die Sortierung hinausgehend ist oftmals ein bestimmte Anordnung der durch eine Anfrage ermittelten Ergebnistupel gewünscht, etwa als inhaltliche Gruppierung.
Gleichzeitig sind oft quantiative Aussagen über Eigenschaften der Resultatmenge --- wie größter oder kleinster Wert sowie Summen- oder Durchschnittsbildung --- gewünscht.

Beispiel 1 zeigt die Ermittlung der Summe aller Gehälter (SQL-Funktion SUM) sowie des Maximal- (MAX), Minimal- (MIN) und Durchschnittsgehalts (AVG) für die Mitarbeiter der *Research*-Abteilung. Die genannten SQL-Funktionen werden als *Aggregierungsfunktionen* bezeichnet, da sie die durch die Abfrage ermittelten Einzelwerte (d.h. die Einträge der Spalte SALARY) jeweils zu genau einer Aussage verdichten.

```
SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO = DNUMBER AND DNAME="Research";
```

Beispiel 75: Aggregierungsfunktionen

Mit der Funktion COUNT steht eine Möglichkeit zur Ermittlung der Mächtigkeit einer Tupelmenge zur Verfügung. Beispiel Beispiel 76 zeigt ihre Verwendung zur Ermittlung der Anzahl der Mitarbeiter der mit *Research* bezeichneten Abteilung.

```
SELECT COUNT(*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO = DNUMBER AND DNAME = "Research";
```

Beispiel 76: Zählfunktion I

Als Argument der COUNT-Funktion kann mit DISTINCT ein Schlüsselwort angegeben werden, welches die ausschließliche Zählung verschiedener Werte erwirkt.
Die Anfrage aus Beispiel Beispiel 77 ermittelt durch Nutzung dieses Schlüsselwortes die Anzahl der verschiedenen Werte in der Spalte SALARY.

```
SELECT COUNT(DISTINCT SALARY)
FROM EMPLOYEE;
```

Beispiel 77: Zählfunktion II

Häufig wird, wie in Beispiel 78 gezeigt, eine Anfrage zur Ermittlung der Anzahl als Unterabfrage formuliert und in der umgebenden Hauptabfrage mit einer Bedingung versehen.

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE (SELECT COUNT(*)
                FROM DEPENDENT
                WHERE SSN=ESSN) >= 2;
```

Beispiel 78: Eingebettete Zählfunktion

Neben den bisher gezeigten aggregierten Aussagen über eine Gesamtmenge besteht oftmals der Wunsch nach von Ermittlung Aussagen dieses Stils über bestimmte Werteklassen innerhalb der betrachteten Gesamtmenge. Hierzu dienen Gruppierungen der Ausgangsmenge, auf welche dann die verschiedenen Aggregierungsfunktionen separat angewandt werden können.
Beispiel Beispiel 79 zeigt dies für die Ermittlung der Mitarbeiteranzahl pro Abteilung sowie der Berechnung des abteilungsinternen Durchschnittsgehalts.

```
SELECT d.DNAME AS "Abteilung", COUNT(*) AS "Anzahl Mitarbeiter", AVG(SALARY) AS
"Durchschnittsgehalt"
FROM EMPLOYEE AS e, DEPARTMENT AS d
WHERE e.DNO = d.DNUMBER
GROUP BY DNO;
```

Beispiel 79: Gruppierung

Zur Realisierung wird die GROUP BY-Klausel verwendet, welche die Angabe eines oder mehrerer Attribute zuläßt anhand der die selektierte Menge partitioniert werden soll.

Die Anfrage des Beispiels Beispiel 80 zeigt die Nutzung einer Verbundbedingung innerhalb einer Gruppierungsanfrage, die Projektnummer und -name sowie vermöge der COUNT-Funktion die Anzahl der das Projekt bearbeitenden Mitarbeiter ermittelt.

```
SELECT PNUMBER, PNAME, COUNT(*) AS "Anzahl Mitarbeiter"
FROM PROJECT, WORKS_ON
WHERE PNUMBER = PNO
GROUP BY PNUMBER, PNAME;
```

Beispiel 80: Gruppierung mit Verbundbedingung

Durch zusätzliche Angabe der HAVING-Klausel kann die Menge der Gruppierungsresultate mittels einer Bedingung beschränkt werden.
So ermittelt die Anfrage aus Beispiel 81 dieselben Resultat wie die in Beispiel 80 gezeigte, jedoch nur für Projekte deren Mitarbeiteranzahl größer 2 ist.

```
SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER = PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT(*) > 2;
```

Beispiel 81: Bedingte Gruppierung

Die formulierte Beschränkung wirkt sich nicht auf die zur Berechnung herangezogene Grundgesamtheit, sondern lediglich auf die Ausgabe der Gruppierungsergebnisse aus, die vor der Auswertung der in der HAVING-Klausel formulierten Bedingung berechnet werden müssen.
Zur Beschränkung der zur Berechnung heranzuziehenden Grundgesamtheit steht auch unter Nutzung der GROUP BY-Klausel der durch WHERE formulierte Bedingungsteil der SELECT-Anfrage zur Verfügung.

```
SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE PNUMBER = PNO AND SSN = ESSN AND DNO=5
GROUP BY PNUMBER, PNAME;
```

Beispiel 82: Beschränkung der Gruppierungseingangsdaten

Gruppierungsschritte können auch in Unterabfragen auftreten, wie das Beispiel 83 zur Ermittlung des Abteilungsnamens und der Anteil der darin arbeitenden Personen mit einem Gehalt über 40000 für alle Abteilungen mit mindestens 2 Mitgliedern zeigt:

```
SELECT DNAME, COUNT(*)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER = DNO AND SALARY > 4000 AND DNO IN (
                SELECT DNO
                FROM EMPLOYEE
                GROUP BY DNO
                HAVING COUNT(*) > 2)
GROUP BY DNUMBER;
```

Beispiel 83: Gruppierung in Unterabfrage

Der Datenmanipulationsteil von SQL

Neben den bisher betrachteten Eigenschaften der Sprache SQL zur Definition von Datenbankstrukturen und zur Abfrage von Datenbankinhalten stehen auch Befehle zur Manipulation in Form von Einfüge-, Aktualisierung- und Löschoperationen zur Verfügung.

Der Einfügebefehl INSERT

Zum Hinzufügen neuer Tupel in eine bestehende Tabelle durch Angabe von Werten für einen oder mehrere Spalten dieser Tabelle wird der Befehl INSERT anboten.
Typischerweise wird dieser Befehlstyp zum Einfügen neuer Datensätze in bestehende Tabellen laufender Applikationen, ebenso wie zur Übernahme kompletter Datenbestände aus existierenden Datenquellen oder zur Neuladung einer Datenbank im Rahmen der Wiederherstellungsprozesses nach einem Systemausfall mit Datenverlust verwendet.

Die allgemeine Syntax des INSERT-Ausdruckes lautet:

```
INSERT INTO tbl_name (col_name,...)? VALUES(constant|NULL ...)
```

```
INSERT INTO EMPLOYEE VALUES(
        'John',
        'B',
        'Smith',
        123456789,
        '1965-01-09',
        '731 Fondren, Houston, TX',
        'M',
        30000,
        333445555,
        5);
```

Beispiel 84: Einfügen eines vollständigen Tupels

Beispiel 84 zeigt den Befehl zur Erzeugung eines neuen Eintrages in der Tabelle EMPLYOEE der Demodatenbank. Die Aufzählung der einzufügenden Werte ist vollständig, d.h. für jede Spalte der Tabelle wird explizit ein konstanter Wert angegeben. Per Konvention müssen alle nichtnumerischen Werte in einfache oder doppelte Hochkommata eingeschlossen werden. Hierunter fallen neben den Zeichenkettentypen auch alle Datumstypen.

Eine Sonderstellung innerhalb der angebbaren Konstanten zur Eintragung stellt die Zeichenkette NULL dar. Sie repräsentiert explizit fehlende Werte, deren Tabelleneinträge entsprechend gekennzeichnet werden. Zur Abgrenzung von der Zeichenkette *NULL* wird diese Angabe nicht in Anführungzeichen eingeschlossen, selbst wenn es sich um eine Spalte eines Zeichenkettentypen handelt.
Beispiel 85 zeigt eine exemplarische Befehlskonstruktion:

```
INSERT INTO EMPLOYEE VALUES(
        'James',
        'E',
        'Borg',
        888665555,
        '1937-11-10',
        '450 Stone, Houston, TX',
        'M',
        55000,
        NULL,
        1);
```

Beispiel 85: Einfügen eines vollständigen Tupels mit NULL-Wert

Neben der Möglichkeit vollständige Tupel einzufügen, kann durch explizite Angabe der einzufügenden Spalten auch eine partielle Befüllung des neu erzeugten Tupels vorgenommen werden.
Für die im INSERT-Befehl nicht angegebenen Spalten wird der spezifizierte Vorgabewert oder NULL eingefügt.
Beispiel 86 zeigt dies exemplarisch anhand des Einfügens der drei Attribute FNAME, LNAME und SSN. Gleichzeitig stellt das Beispiel auch heraus, daß bei expliziter Angabe der einzufügenden Spalten die gewählte Reihenfolge von der in der Tabelle realisierten abweichen kann.

```
INSERT INTO EMPLOYEE (LNAME, FNAME, SSN) VALUES(
        'Doe',
        'John',
        912873465);
```

Beispiel 86: Einfügen eines unvollständigen Tupels

Prinzipiell kann jedes Element der Potenzmenge der Attribute einer Relation eingefügt werden. Es muß jedoch zwingend einen Wert für das Primärschlüsselattribut enthalten, da hierfür der Wert NULL nicht gesetzt werden darf.

***Der Aktualisierungsbefehl** UPDATE*

Zur Aktualisierung von Werten innerhalb bestehender Datenbankeinträge bietet der SQL-Sprachumfang den Befehl UPDATE an, der es gestattet frei wählbare Mengen von Tupeln einer Tabelle zu modifizieren.

Die allgemeine Syntax des Befehls lautet:

```
UPDATE tbl_name SET col_name=expression, ... [WHERE search_condition]
```

Beispiel Beispiel 87 zeigt den Befehl zur null-Setzung aller in der Tabelle EMPLOYEE verwalteten Geburtsdaten (BDATA):

```
UPDATE EMPLOYEE SET BDATE=NULL;
```

Beispiel 87: Modifikation aller Tupel durch Setzen eines konstanten Wertes

Neben der Eintragung von Konstanten können auch neue Inhalte aus den Bisherigen errechnet werden. So zeigt Beispiel 88 eine Aktualisierung, die das Gehalt (SALARY) aller Mitarbeiter um zehn Prozent erhöht:

```
UPDATE EMPLOYEE SET Salary=Salary*1.1;
```

Beispiel 88: Modifikation aller Tupel durch Setzen eines berechneten Wertes

Durch Nutzung der, identisch zum SELECT-Ausdruck aufgebauten, WHERE-Klausel kann die Menge der von der Änderung betroffenen Datensätze eingeschänkt werden.
Das Beispiel Beispiel 89 ändert in allen Einträgen, deren LNAME auf Zelaya lautet den Wert zu Jones.
Die Anzahl der betroffenen Tupel ist durch den UPDATE-Ausdruck nicht festlegbar, sondern richtet sich ausschließlich nach der durch die WHERE-Klausel selektierten Eintragsmenge.

```
UPDATE EMPLOYEE SET LNAME='Jones'
WHERE LNAME='Zelaya';
```

Beispiel 89: Modifikation von Tupeln

Durch die Nutzbarkeit der vollständigen Möglichkeiten der aus dem SELECT-Befehl bekannten Mächtigkeit der WHERE-Klausel lassen sich selbst komplexe Aktualisierungen realiseren.
Beispiel 90 zeigt führt die Erhöhung der Gehälter derjenigen Mitarbeiter durch, die Abteilungen zugewiesen sind, die mehr als zwei Projekte bearbeiten.

```
UPDATE EMPLOYEE SET SALARY=SALARY*1.1 WHERE DNO IN
        (SELECT DNUMBER
        FROM PROJECT AS p, DEPARTMENT AS d
        WHERE d.DNUMBER=p.DNUM
        GROUP BY 1
        HAVING COUNT(*) > 2);
```

Beispiel 90: Modifikation von Tupeln (Ermittlung der betroffenen Tupel durch Subanfrage)

***Der Löschbefehl** DELETE*

Zur Löschung von verwalteten Tupeln aus einer Tabelle existiert der DELETE-Befehl, der die betroffenen Datensätze ohne weite Nachfrage entfernt.

Seine allgemeine Syntax lautet:
`DELTE FROM tbl_name [WHERE search_condition]`

Die einfachste Ausprägung der DELETE-Anweisung löscht alle Tupel einer Tabelle:

```
DELETE FROM EMPLOYEE;
```

Beispiel 91: Löschen aller Tupel einer Tabelle

Durch Angabe der WHERE-Klausel können, wie bereits bei UPDATE für die zu aktualisierenden Tupel gezeigt, die zu löschenden Tupel eingegrenzt werden.
So entfernt der Ausdruck aus Beispiel 92 alle Mitarbeiter die in Houston wohnen.

```
DELETE FROM EMPLOYEE
WHERE ADDRESS LIKE "%Houston%";
```

Beispiel 92: Löschen aller Mitarbeiter, die in Houston wohnhaft sind

Durch die Nutzung der expliziten Mengenangabe innerhalb der WHERE-Klausel läßt sich die Menge der zu entfernenden Datensätze statische eingrenzen wie Beispiel 93 zeigt.

```
DELETE EMPLOYEE
WHERE SSN IN (333445555, 888665555, 987987987);
```

Beispiel 93: Löschen bestimmter Datenstätze

Beispiel 94 zeigt die Nutzung einer Unterabfrage zur Ermittlung aller Abteilungen, die nur genau ein Projekt durchführen und anschließenden Löschung dieser Abteilungen aus der Tabelle DEPARTMENT.

```
DELETE FROM DEPARTMENT
WHERE DNUMER IN
        (SELECT DNUM
        FROM PROJECT
        GROUP BY 1
        HAVING COUNT(*) = 1);
```

Beispiel 94: Löschen aller Abteilungen, die nur genau ein Projekt durchführen

Definitionsverzeichnis

Assoziation
Assoziationstyp
Äußerer Verbund
Boyce/Codd-Normalform
Daten
Datenbank
Datenbankmanagementsystem (DBMS)
Datenbanksprache
Datenunabhängigkeit
Dritte Normalform (3NF)
Entität
Entitätstyp
Erste Normalform (1NF)
Fünfte Normalform
Hybrider Entitäts-Assoziationstyp
Index
Information
Innerer Verbund
Kardinalitätsintervall
Konzeptuelles Schema
Logisches Schema
Mehrwertige Abhängigkeit
Metainformation
Modell
NULL-Wert
Physisches Schema
Primärschüssel
Projektion
Referentielle Integrität
Relation
Relationales DBMS
Repräsentation
Repräsentationstyp
Rolle
Schüssel
Selektion
Spezialisierungsassoziationstyp
Superschlüssel
Tabelle
Transitive Abhängigkeit
Triviale mehrwertige Abhängigkeit
Vierte Normalform
Volle funktionale Abhängigkeit
Vollständiges konzeptuelles Schema
Zweite Normalform (2NF)

Schlagwortverzeichnis

Abbildungsverzeichnis

Verzeichnis der Beispiele

```
import java.sql.Array;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCSelect8 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException cnfe) {
			System.err.println("Driver class not found");
			cnfe.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException sqle) {
			System.err.println("Error establishing database connection");
			Throwable t = sqle;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		Statement stmt = null;
		try {
			stmt = (Statement) con.createStatement();
		} catch (SQLException sqle) {
			System.err.println("Error creating SQL-Statement");
			Throwable t = sqle;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
```

```
            }
        }

        try {
            ResultSet rs = stmt.executeQuery("SELECT * FROM EMPLOYEE;");
            while (!rs.isLast()) {
                rs.first();
                System.out.print(rs.getString("FNAME") + "\t");
                Array cars = rs.getArray("CAR");
                ResultSet carsRS = cars.getResultSet();
                System.out.print("(");
                while (!carsRS.isLast()) {
                    rs.first();
                    System.out.print(carsRS.getString("CAR"));
                    carsRS.next();
                }
                System.out.println(")");
                rs.next();
            }
        } catch (SQLException sqle) {
            System.err.println("Error selecting values from table EMPLOYEE");
            Throwable t = sqle;
            while (t != null) {
            System.err.println("Type: " + t.getClass().getName());
        System.err.println("Message: " + t.getMessage());
        System.err.println("-----");
        t = t.getCause();
            }
        }
    }
}
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCSelect9 {
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException cnfe) {
			System.err.println("Driver class not found");
			cnfe.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException sqle) {
			System.err.println("Error establishing database connection");
			Throwable t = sqle;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}

		try {
			File file = new File(args[0]);
			FileInputStream fis = new FileInputStream(args[0]);
			PreparedStatement pstmt =
				con.prepareStatement(
					"UPDATE EMPLOYEE SET binData =? WHERE  SSN=123456789");
```

```
                pstmt.setBinaryStream(1, fis, (int) file.length());
                pstmt.executeUpdate();
                fis.close();

                //read it back from the database
                Statement stmt = (Statement) con.createStatement();
                ResultSet rs =
                    stmt.executeQuery(
                        "SELECT binData FROM EMPLOYEE WHERE SSN='123456789';");

                FileOutputStream fos = new FileOutputStream(args[1]);
                if (rs.next())
                    fos.write(rs.getBytes(1));
                fos.close();

            } catch (SQLException sqle) {
                System.err.println("Error selecting values from table EMPLOYEE");
                Throwable t = sqle;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            } catch (IOException ioe) {
                ioe.printStackTrace();
            }
        }
}
```

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCTransact1 {
	private static void printContent(Statement stmt) throws SQLException {
		ResultSet rs =
			stmt.executeQuery("SELECT FNAME,MINIT,LNAME FROM EMPLOYEE;");
		while (!rs.isLast()) {
			rs.next();
			System.out.println(
				rs.getString("LNAME")
					+ "\t"
					+ rs.getString("MINIT")
					+ "\t"
					+ rs.getString("LNAME"));
		}
	}
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException cnfe) {
			System.err.println("Driver class not found");
			cnfe.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException sqle) {
			System.err.println("Error establishing database connection");
			Throwable t = sqle;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
```

```
        }

        Statement stmt = null;
        try {
            stmt = (Statement) con.createStatement();
        } catch (SQLException sqle) {
            System.err.println("Error creating SQL-Statement");
            Throwable t = sqle;
            while (t != null) {
            System.err.println("Type: " + t.getClass().getName());
        System.err.println("Message: " + t.getMessage());
        System.err.println("-----");
        t = t.getCause();
            }
        }

        try {
            int transactionIsolation = con.getTransactionIsolation();
            switch (transactionIsolation) {
                case Connection.TRANSACTION_NONE :
                    System.out.println("Transactions are not supported");
                    break;
                case Connection.TRANSACTION_READ_UNCOMMITTED :
                    System.out.println(
                        "Dirty reads, non-repeatable reads and phantom reads can occur");
                    break;
                case Connection.TRANSACTION_READ_COMMITTED :
                    System.out.println(
                        "Dirty reads are prevented; non-repeatable reads and phantom reads can
occur");
                    break;
                case Connection.TRANSACTION_REPEATABLE_READ :
                    System.out.println(
                        "Dirty reads and non-repeatable reads are prevented; phantom reads can
occur");
                    break;
                case Connection.TRANSACTION_SERIALIZABLE :
                    System.out.println(
                        "Dirty reads, non-repeatable reads and phantom reads are prevented");
                    break;
            }
            if (transactionIsolation < Connection.TRANSACTION_SERIALIZABLE) {
                con.setTransactionIsolation(
                    Connection.TRANSACTION_SERIALIZABLE);
                if (con.getTransactionIsolation()
                    != Connection.TRANSACTION_SERIALIZABLE) {
```

```
                        System.out.println(
                            "cannot set Connection.TRANSACTION_SERIALIZABLE");
                    } else {
                        System.out.println(
                            "reached highest possible isolation level");
                    }
                }

                con.setAutoCommit(false);
                stmt.executeUpdate(
                    "INSERT INTO EMPLOYEE VALUES('Hans','X','Hinterhuber','111111111',
NULL,NULL,NULL,NULL,NULL,NULL);");
                stmt.executeUpdate(
                    "INSERT INTO EMPLOYEE VALUES('Franz','X','Obermüller','222222222',
NULL,NULL,NULL,NULL,NULL,NULL);");
                printContent(stmt);
                //suppose error happens here
                Thread.sleep(5000);
                boolean error = true;
                if (error) {
                    con.rollback();
                } else {
                    stmt.executeUpdate(
                        "INSERT INTO EMPLOYEE VALUES('Fritz','X','Meier','333333333',
NULL,NULL,NULL,NULL,NULL,NULL);");
                }
                printContent(stmt);
            } catch (SQLException sqle) {
                Throwable t = sqle;
                while (t != null) {
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            } catch (InterruptedException ie) {
                ie.printStackTrace();
            }
        }
}
```

Dirty reads and non-repeatable reads are prevented; phantom reads can occur
reached highest possible isolation level
Hinterhube X Hinterhube
Smith B Smith
Obermüller X Obermüller
Wong T Wong
English A English
Narayan K Narayan
Borg E Borg
Wallace S Wallace
Jabbar V Jabbar
Zelaya J Zelaya
Smith B Smith
Wong T Wong
English A English
Narayan K Narayan
Borg E Borg
Wallace S Wallace
Jabbar V Jabbar
Zelaya J Zelaya

```
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Savepoint;

import com.mysql.jdbc.Connection;
import com.mysql.jdbc.Statement;

public class JDBCTransact2 {
	private static void printContent(Statement stmt) throws SQLException {
		ResultSet rs =
			stmt.executeQuery("SELECT FNAME,MINIT,LNAME FROM EMPLOYEE;");
		while (!rs.isLast()) {
			rs.next();
			System.out.println(
				rs.getString("LNAME")
					+ "\t"
					+ rs.getString("MINIT")
					+ "\t"
					+ rs.getString("LNAME"));
		}
	}
	public static void main(String[] args) {
		try {
			Class.forName("com.mysql.jdbc.Driver");
		} catch (ClassNotFoundException cnfe) {
			System.err.println("Driver class not found");
			cnfe.printStackTrace();
		}
		Connection con = null;

		try {
			con =
				(Connection) DriverManager.getConnection(
					"jdbc:mysql://localhost/jdbctest/",
					"mario",
					"thePassword");
		} catch (SQLException sqle) {
			System.err.println("Error establishing database connection");
			Throwable t = sqle;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
```

```
            }
        }

        Statement stmt = null;
        try {
            stmt = (Statement) con.createStatement();
        } catch (SQLException sqle) {
            System.err.println("Error creating SQL-Statement");
            Throwable t = sqle;
            while (t != null) {
            System.err.println("Type: " + t.getClass().getName());
        System.err.println("Message: " + t.getMessage());
        System.err.println("-----");
        t = t.getCause();
            }
        }

        try {
            int transactionIsolation = con.getTransactionIsolation();
            switch (transactionIsolation) {
                case Connection.TRANSACTION_NONE :
                    System.out.println("Transactions are not supported");
                    break;
                case Connection.TRANSACTION_READ_UNCOMMITTED :
                    System.out.println(
                        "Dirty reads, non-repeatable reads and phantom reads can occur");
                    break;
                case Connection.TRANSACTION_READ_COMMITTED :
                    System.out.println(
                        "Dirty reads are prevented; non-repeatable reads and phantom reads can
occur");
                    break;
                case Connection.TRANSACTION_REPEATABLE_READ :
                    System.out.println(
                        "Dirty reads and non-repeatable reads are prevented; phantom reads can
occur");
                    break;
                case Connection.TRANSACTION_SERIALIZABLE :
                    System.out.println(
                        "Dirty reads, non-repeatable reads and phantom reads are prevented");
                    break;
            }
            if (transactionIsolation < Connection.TRANSACTION_SERIALIZABLE) {
                con.setTransactionIsolation(
                    Connection.TRANSACTION_SERIALIZABLE);
                if (con.getTransactionIsolation()
```

```
                    != Connection.TRANSACTION_SERIALIZABLE) {
                    System.out.println(
                        "cannot set Connection.TRANSACTION_SERIALIZABLE");
                } else {
                    System.out.println(
                        "reached highest possible isolation level");
                }
            }

            con.setAutoCommit(false);
            stmt.executeUpdate(
                "INSERT INTO EMPLOYEE VALUES('Hans','X','Hinterhuber','111111111',
NULL,NULL,NULL,NULL,NULL,NULL);");
            Savepoint sp = con.setSavepoint();
            stmt.executeUpdate(
                "INSERT INTO EMPLOYEE VALUES('Franz','X','Obermüller','222222222',
NULL,NULL,NULL,NULL,NULL,NULL);");
            printContent(stmt);
            //suppose error happens here
            Thread.sleep(5000);
            boolean error = true;
            if (error) {
                con.rollback(sp);
            }
            stmt.executeUpdate(
                "INSERT INTO EMPLOYEE VALUES('Fritz','X','Meier','333333333',NULL,
NULL,NULL,NULL,NULL,NULL);");
            printContent(stmt);
            con.commit();
        } catch (SQLException sqle) {
            Throwable t = sqle;
            while (t != null) {
            System.err.println("Type: " + t.getClass().getName());
        System.err.println("Message: " + t.getMessage());
        System.err.println("-----");
        t = t.getCause();
            }
        } catch (InterruptedException ie) {
            ie.printStackTrace();
        }
    }
}
```

```
0000 4  00 00 00 \n 4 . 0 . 1 2 - s t a n d a r d - l o g 00 0e 00 00 00 L # d
0020 S T * q | 00 ,  \b 02 00 00 00 00 00 00 00 00 00 00 00 00 00 14 00 00 01 87 00 ÿ ÿ
0040 ÿ m a r i o 00 B O A Q E @ V V 00 03 00 00 02 00 00 00 \t 00 00 00 02 j d b c
0060 t e s t 03 00 00 01 00 00 00 0f 00 00 00 03 S H O W  V A R I A B L E S 01 00
0080 00 01 02 19 00 00 02 00 \r V a r i a b l e _ n a m e 03 1e 00 00 01 þ 03 01 00 1f
00a0 11 00 00 03 00 05 V a l u e 03 00 01 00 01 þ 03 01 00 1f 01 00 00 04 þ \f 00 00 05 \b b
00c0 a c k _ l o g 02 5 0 7 00 00 06 07 b a s e d i r . / o p t / r a i d
00e0 / m y s q l - s t a n d a r d - 4 . 0 . 1 2 - p c - l i n u x -
0100 i 6 8 6 / 18 00 00 07 11 b i n l o g _ c a c h e _ s i z e 05 3 2 7 6
0120 8  00 00 \b 17 b u l k _ i n s e r t _ b u f f e r _ s i z e 07 8 3
0140 8 8 6 0 8 15 00 00 \t \r c h a r a c t e r _ s e t 06 l a t i n 1 á 00
0160 00 \n 0e c h a r a c t e r _ s e t s Ñ l a t i n 1  b i g 5  c z
0180 e c h  e u c _ k r  g b 2 3 1 2  g b k  l a t i n 1 _ d e
01a0 s j i s  t i s 6 2 0  u j i s  d e c 8  d o s  g e r m a n
01c0 1  h p 8  k o i 8 _ r u  l a t i n 2  s w e 7  u s a 7  c
01e0 p 1 2 5 1  d a n i s h  h e b r e w  w i n 1 2 5 1  e s t o
0200 n i a  h u n g a r i a n  k o i 8 _ u k r  w i n 1 2 5 1 u k
0220 r  g r e e k  w i n 1 2 5 0  c r o a t  c p 1 2 5 7  l a t
0240 i n 5 15 00 00 0b 11 c o n c u r r e n t _ i n s e r t 02 O N 12 00 00 \f
0260 0f c o n n e c t _ t i m e o u t 01 5 17 00 00 \r 15 c o n v e r t _ c
0280 h a r a c t e r _ s e t 00 < 00 00 0e 07 d a t a d i r 3 / o p t / r
02a0 a i d / m y s q l - s t a n d a r d - 4 . 0 . 1 2 - p c - l i n
02c0 u x - i 6 8 6 / d a t a / 13 00 00 0f 0f d e l a y _ k e y _ w r i t
02e0 e 02 O N 19 00 00 10 14 d e l a y e d _ i n s e r t _ l i m i t 03 1 0
0300 0  1b 00 00 11 16 d e l a y e d _ i n s e r t _ t i m e o u t 03 3 0 0
0320 18 00 00 12 12 d e l a y e d _ q u e u e _ s i z e 04 1 0 0 0 \n 00 00 13
0340 05 f l u s h 03 O F F \r 00 00 14 \n f l u s h _ t i m e 01 0 ! 00 00 15 11
0360 f t _ b o o l e a n _ s y n t a x 0e +  - > < ( ) ~ * : " " & |
0380 12 00 00 16 0f f t _ m i n _ w o r d _ l e n 01 4 14 00 00 17 0f f t _ m a
03a0 x _ w o r d _ l e n 03 2 5 4 1c 00 00 18 18 f t _ m a x _ w o r d _ l
03c0 e n _ f o r _ s o r t 02 2 0 1c 00 00 19 10 f t _ s t o p w o r d _ f
03e0 i l e \n ( b u i l t - i n ) \f 00 00 1a \b h a v e _ b d b 02 N O 0f 00
0400 00 1b \n h a v e _ c r y p t 03 Y E S 10 00 00 1c 0b h a v e _ i n n o d
0420 b 03 Y E S 0e 00 00 1d \t h a v e _ i s a m 03 Y E S \r 00 00 1e \t h a v e
0440 _ r a i d 02 N O 16 00 00 1f \f h a v e _ s y m l i n k \b D I S A B L
0460 E D 10 00 00  \f h a v e _ o p e n s s l 02 N O 15 00 00 ! 10 h a v e _
0480 q u e r y _ c a c h e 03 Y E S 0b 00 00 " \t i n i t _ f i l e 00 ( 00
04a0 00 # 1f i n n o d b _ a d d i t i o n a l _ m e m _ p o o l _ s i
04c0 z e 07 2 0 9 7 1 5 2 ! 00 00 $ 17 i n n o d b _ b u f f e r _ p o o
04e0 l _ s i z e \b 1 6 7 7 7 2 1 6 - 00 00 % 15 i n n o d b _ d a t a _
0500 f i l e _ p a t h 16 i b d a t a 1 : 1 0 M : a u t o e x t e n d
0520 16 00 00 & 14 i n n o d b _ d a t a _ h o m e _ d i r 00 19 00 00 ' 16 i
0540 n n o d b _ f i l e _ i o _ t h r e a d s 01 4 18 00 00 ( 15 i n n o
0560 d b _ f o r c e _ r e c o v e r y 01 0 1c 00 00 ) 19 i n n o d b _ t
0580 h r e a d _ c o n c u r r e n c y 01 8 ! 00 00 * 1e i n n o d b _ f
```

```
05a0 l u s h _ l o g _ a t _ t r x _ c o m m i t 01 1 18 00 00 + 14 i n n
05c0 o d b _ f a s t _ s h u t d o w n 02 O N 15 00 00 , 13 i n n o d b _
05e0 f l u s h _ m e t h o d 00 1c 00 00 - 18 i n n o d b _ l o c k _ w a
0600 i t _ t i m e o u t 02 5 0 17 00 00 . 13 i n n o d b _ l o g _ a r c
0620 h _ d i r 02 . / 17 00 00 / 12 i n n o d b _ l o g _ a r c h i v e 03
0640 O F F 1f 00 00 0 16 i n n o d b _ l o g _ b u f f e r _ s i z e 07 8
0660 3 8 8 6 0 8 1d 00 00 1 14 i n n o d b _ l o g _ f i l e _ s i z e 07
0680 5 2 4 2 8 8 0 1c 00 00 2 19 i n n o d b _ l o g _ f i l e s _ i n _
06a0 g r o u p 01 2 1d 00 00 3 19 i n n o d b _ l o g _ g r o u p _ h o m
06c0 e _ d i r 02 . / 1d 00 00 4 1a i n n o d b _ m i r r o r e d _ l o g
06e0 _ g r o u p s 01 1 1a 00 00 5 13 i n t e r a c t i v e _ t i m e o u
0700 t 05 2 8 8 0 0 18 00 00 6 10 j o i n _ b u f f e r _ s i z e 06 1 3 1
0720 0 7 2 19 00 00 7 0f k e y _ b u f f e r _ s i z e \b 1 6 7 7 7 2 1 6
0740 L 00 00 8 \b l a n g u a g e B / o p t / r a i d / m y s q l - s t
0760 a n d a r d - 4 . 0 . 1 2 - p c - l i n u x - i 6 8 6 / s h a r
0780 e / m y s q l / e n g l i s h / 17 00 00 9 13 l a r g e _ f i l e s
07a0 _ s u p p o r t 02 O N 10 00 00 : \f l o c a l _ i n f i l e 02 O N 15
07c0 00 00 ; 10 l o c k e d _ i n _ m e m o r y 03 O F F \b 00 00 < 03 l o g
07e0 03 O F F 0f 00 00 = \n l o g _ u p d a t e 03 O F F 0b 00 00 > 07 l o g _
0800 b i n 02 O N 16 00 00 ? 11 l o g _ s l a v e _ u p d a t e s 03 O F F
0820 15 00 00 @ 10 l o g _ s l o w _ q u e r i e s 03 O F F 11 00 00 A \f l o
0840 g _ w a r n i n g s 03 O F F 13 00 00 B 0f l o n g _ q u e r y _ t i
0860 m e 02 1 0 19 00 00 C 14 l o w _ p r i o r i t y _ u p d a t e s 03 O
0880 F F 1b 00 00 D 16 l o w e r _ c a s e _ t a b l e _ n a m e s 03 O F
08a0 F 1b 00 00 E 12 m a x _ a l l o w e d _ p a c k e t 07 1 0 4 7 5 5 2
08c0 ! 00 00 F 15 m a x _ b i n l o g _ c a c h e _ s i z e \n 4 2 9 4 9
08e0 6 7 2 9 5 1b 00 00 G 0f m a x _ b i n l o g _ s i z e \n 1 0 7 3 7 4
0900 1 8 2 4 14 00 00 H 0f m a x _ c o n n e c t i o n s 03 1 0 0 16 00 00 I
0920 12 m a x _ c o n n e c t _ e r r o r s 02 1 0 17 00 00 J 13 m a x _ d
0940 e l a y e d _ t h r e a d s 02 2 0 1d 00 00 K 13 m a x _ h e a p _ t
0960 a b l e _ s i z e \b 1 6 7 7 7 2 1 6 19 00 00 L \r m a x _ j o i n _
0980 s i z e \n 4 2 9 4 9 6 7 2 9 5 15 00 00 M 0f m a x _ s o r t _ l e n
09a0 g t h 04 1 0 2 4 17 00 00 N 14 m a x _ u s e r _ c o n n e c t i o n
09c0 s 01 0 12 00 00 O 0e m a x _ t m p _ t a b l e s 02 3 2  00 00 P 14 m a
09e0 x _ w r i t e _ l o c k _ c o u n t \n 4 2 9 4 9 6 7 2 9 5 * 00 00
0a00 Q 1f m y i s a m _ m a x _ e x t r a _ s o r t _ f i l e _ s i z
0a20 e \t 2 6 8 4 3 5 4 5 6 % 00 00 R 19 m y i s a m _ m a x _ s o r t _
0a40 f i l e _ s i z e \n 2 1 4 7 4 8 3 6 4 7 1b 00 00 S 16 m y i s a m _
0a60 r e c o v e r _ o p t i o n s 03 O F F  00 00 T 17 m y i s a m _ s
0a80 o r t _ b u f f e r _ s i z e 07 8 3 8 8 6 0 8 17 00 00 U 11 n e t _
0aa0 b u f f e r _ l e n g t h 04 8 1 9 2 14 00 00 V 10 n e t _ r e a d _
0ac0 t i m e o u t 02 3 0 13 00 00 W 0f n e t _ r e t r y _ c o u n t 02 1
0ae0 0 15 00 00 X 11 n e t _ w r i t e _ t i m e o u t 02 6 0 \b 00 00 Y 03 n
0b00 e w 03 O F F 13 00 00 Z 10 o p e n _ f i l e s _ l i m i t 01 0 F 00 00
0b20 [ \b p i d _ f i l e < / o p t / r a i d / m y s q l - s t a n d
0b40 a r d - 4 . 0 . 1 2 - p c - l i n u x - i 6 8 6 / d a t a / l i
```

```
0b60 n u x . p i d 0b 00 00 \ \t l o g _ e r r o r 00 \n 00 00 ] 04 p o r t 04
0b80 3 3 0 6 14 00 00 ^ 10 p r o t o c o l _ v e r s i o n 02 1 0 18 00 00 _
0ba0 10 r e a d _ b u f f e r _ s i z e 06 1 3 1 0 7 2 1c 00 00 ` 14 r e a
0bc0 d _ r n d _ b u f f e r _ s i z e 06 2 6 2 1 4 4 14 00 00 a 11 r p l
0be0 _ r e c o v e r y _ r a n k 01 0 1a 00 00 b 11 q u e r y _ c a c h e
0c00 _ l i m i t 07 1 0 4 8 5 7 6 13 00 00 c 10 q u e r y _ c a c h e _ s
0c20 i z e 01 0 14 00 00 d 10 q u e r y _ c a c h e _ t y p e 02 O N \f 00 00
0c40 e \t s e r v e r _ i d 01 1 17 00 00 f 11 s l a v e _ n e t _ t i m e
0c60 o u t 04 3 6 0 0 19 00 00 g 15 s k i p _ e x t e r n a l _ l o c k i
0c80 n g 02 O N 14 00 00 h 0f s k i p _ n e t w o r k i n g 03 O F F 17 00 00
0ca0 i 12 s k i p _ s h o w _ d a t a b a s e 03 O F F 13 00 00 j 10 s l o
0cc0 w _ l a u n c h _ t i m e 01 2 17 00 00 k 06 s o c k e t 0f / t m p /
0ce0 m y s q l . s o c k 18 00 00 l 10 s o r t _ b u f f e r _ s i z e 06
0d00 5 2 4 2 8 0 0b 00 00 m \b s q l _ m o d e 01 0 0f 00 00 n 0b t a b l e _
0d20 c a c h e 02 6 4 12 00 00 o \n t a b l e _ t y p e 06 I N N O D B 14 00
0d40 00 p 11 t h r e a d _ c a c h e _ s i z e 01 0 14 00 00 q \f t h r e a
0d60 d _ s t a c k 06 1 9 6 6 0 8 1d 00 00 r \f t x _ i s o l a t i o n 0f
0d80 R E P E A T A B L E - R E A D 0e 00 00 s \b t i m e z o n e 04 C E S
0da0 T 18 00 00 t 0e t m p _ t a b l e _ s i z e \b 3 3 5 5 4 4 3 2 \r 00 00
0dc0 u 06 t m p d i r 05 / t m p / 1c 00 00 v 07 v e r s i o n 13 4 . 0 . 1
0de0 2 - s t a n d a r d - l o g 13 00 00 w \f w a i t _ t i m e o u t 05
0e00 2 8 8 0 0 01 00 00 x þ 11 00 00 00 03 S E T   a u t o c o m m i t = 1 03
0e20 00 00 01 00 00 00 18 00 00 00 03 S E L E C T   *   F R O M   E M P L O Y E
0e40 E ; 01 00 00 01 \n 19 00 00 02 \b E M P L O Y E E 05 F N A M E 03 \n 00 00 01 ý
0e60 03 01 00 00 19 00 00 03 \b E M P L O Y E E 05 M I N I T 03 01 00 00 01 þ 03 00 00
0e80 00 19 00 00 04 \b E M P L O Y E E 05 L N A M E 03 \n 00 00 01 ý 03 01 00 00 17 00
0ea0 00 05 \b E M P L O Y E E 03 S S N 03 \t 00 00 01 03 03 01 00 00 19 00 00 06 \b E M
0ec0 P L O Y E E 05 B D A T E 03 \n 00 00 01 \n 03 00 00 00 1b 00 00 07 \b E M P L O
0ee0 Y E E 07 A D D R E S S 03 1e 00 00 01 ý 03 00 00 00 17 00 00 \b \b E M P L O Y
0f00 E E 03 S E X 03 01 00 00 01 þ 03 00 01 00 1a 00 00 \t \b E M P L O Y E E 06 S A
0f20 L A R Y 03 07 00 00 01 05 03   00 02 1c 00 00 \n \b E M P L O Y E E \b S U P E
0f40 R S S N 03 \t 00 00 01 03 03 00 00 00 17 00 00 0b \b E M P L O Y E E 03 D N O 03
0f60 01 00 00 01 03 03 00 00 00 01 00 00 \f þ R 00 00 \r 04 J o h n 01 B 05 S m i t h \t
0f80 1 2 3 4 5 6 7 8 9 \n 1 9 6 5 - 0 1 - 0 9 18 7 3 1   F o n d r e n
0fa0 ,   H o u s t o n ,   T X 01 M \b 3 0 0 0 0 . 0 0 \t 3 3 3 4 4 5 5
0fc0 5 5 01 5 R 00 00 0e \b F r a n k l i n 01 T 04 W o n g \t 3 3 3 4 4 5 5
0fe0 5 5 \n 1 9 5 5 - 1 2 - 0 8 15 6 3 8   V o s s ,   H o u s t o n ,
1000   T X 01 M \b 4 0 0 0 0 . 0 0 \t 8 8 8 6 6 5 5 5 5 01 5 T 00 00 0f 06 A
1020 l i c i a 01 J 06 Z e l a y a \t 9 9 9 8 8 7 7 7 \n 1 9 6 8 - 0 7
1040 - 1 9 17 3 3 2 1   C a s t l e ,   S p r i n g ,   T X 01 F \b 2 5
1060 0 0 0 . 0 0 \t 9 8 7 6 5 4 3 2 1 01 4 W 00 00 10 \b J e n n i f e r 01
1080 S 07 W a l l a c e \t 9 8 7 6 5 4 3 2 1 \n 1 9 4 1 - 0 6 - 2 0 17 2
10a0 9 1   B e r r y ,   B e l l a i r e ,   T X 01 F \b 4 3 0 0 0 . 0
10c0 0 \t 8 8 8 6 6 5 5 5 5 01 4 V 00 00 11 06 R a m e s h 01 K 07 N a r a y
10e0 a n \t 6 6 6 8 8 4 4 4 4 \n 1 9 6 2 - 0 9 - 1 5 18 9 7 5   F i r e
1100   O a k ,   H u m b l e ,   T X 01 M \b 3 8 0 0 0 . 0 0 \t 3 3 3 4
```

1120 4 5 5 5 5 01 5 S 00 00 12 05 J o y c e 01 A 07 E n g l i s h \t 4 5 3 4
1140 5 3 4 5 3 \n 1 9 7 2 - 0 7 - 3 1 16 5 6 3 1 R i c e , H o u s
1160 t o n , T X 01 F \b 2 5 0 0 0 . 0 0 \t 3 3 3 4 4 5 5 5 5 01 5 S 00
1180 00 13 05 A h m a d 01 V 06 J a b b a r \t 9 8 7 9 8 7 9 8 7 \n 1 9 6 9
11a0 - 0 3 - 2 9 17 9 8 0 D a l l a s , H o u s t o n , T X 01 M
11c0 \b 2 5 0 0 0 . 0 0 \t 9 8 7 6 5 4 3 2 1 01 4 G 00 00 14 05 J a m e s 01
11e0 E 04 B o r g \t 8 8 8 6 6 5 5 5 5 \n 1 9 3 7 - 1 1 - 1 0 16 4 5 0
1200 S t o n e , H o u s t o n , T X 01 M \b 5 5 0 0 0 . 0 0 û 01 1
1220 01 00 00 15 þ 18 00 00 00 03 S E L E C T * F R O M E M P L O Y E E
1240 ; 01 00 00 01 \n 19 00 00 02 \b E M P L O Y E E 05 F N A M E 03 \n 00 00 01 ý 03
1260 01 00 00 19 00 00 03 \b E M P L O Y E E 05 M I N I T 03 01 00 00 01 þ 03 00 00 00
1280 19 00 00 04 \b E M P L O Y E E 05 L N A M E 03 \n 00 00 01 ý 03 01 00 00 17 00 00
12a0 05 \b E M P L O Y E E 03 S S N 03 \t 00 00 01 03 03 01 00 00 19 00 00 06 \b E M P
12c0 L O Y E E 05 B D A T E 03 \n 00 00 01 \n 03 00 00 00 1b 00 00 07 \b E M P L O Y
12e0 E E 07 A D D R E S S 03 1e 00 00 01 ý 03 00 00 00 17 00 00 \b \b E M P L O Y E
1300 E 03 S E X 03 01 00 00 01 þ 03 00 01 00 1a 00 00 \t \b E M P L O Y E E 06 S A L
1320 A R Y 03 07 00 00 01 05 03 00 02 1c 00 00 \n \b E M P L O Y E E \b S U P E R
1340 S S N 03 \t 00 00 01 03 03 00 00 00 17 00 00 0b \b E M P L O Y E E 03 D N O 03 01
1360 00 00 01 03 03 00 00 00 01 00 00 \f þ R 00 00 \r 04 J o h n 01 B 05 S m i t h \t 1
1380 2 3 4 5 6 7 8 9 \n 1 9 6 5 - 0 1 - 0 9 18 7 3 1 F o n d r e n ,
13a0 H o u s t o n , T X 01 M \b 3 0 0 0 0 . 0 0 \t 3 3 3 4 4 5 5 5
13c0 5 01 5 R 00 00 0e \b F r a n k l i n 01 T 04 W o n g \t 3 3 3 4 4 5 5 5
13e0 5 \n 1 9 5 5 - 1 2 - 0 8 15 6 3 8 V o s s , H o u s t o n ,
1400 T X 01 M \b 4 0 0 0 0 . 0 0 \t 8 8 8 6 6 5 5 5 5 01 5 T 00 00 0f 06 A l
1420 i c i a 01 J 06 Z e l a y a \t 9 9 9 8 8 7 7 7 7 \n 1 9 6 8 - 0 7 -
1440 1 9 17 3 3 2 1 C a s t l e , S p r i n g , T X 01 F \b 2 5 0
1460 0 0 . 0 0 \t 9 8 7 6 5 4 3 2 1 01 4 W 00 00 10 \b J e n n i f e r 01 S
1480 07 W a l l a c e \t 9 8 7 6 5 4 3 2 1 \n 1 9 4 1 - 0 6 - 2 0 17 2 9
14a0 1 B e r r y , B e l l a i r e , T X 01 F \b 4 3 0 0 0 . 0 0
14c0 \t 8 8 8 6 6 5 5 5 5 01 4 V 00 00 11 06 R a m e s h 01 K 07 N a r a y a
14e0 n \t 6 6 6 8 8 4 4 4 4 \n 1 9 6 2 - 0 9 - 1 5 18 9 7 5 F i r e
1500 O a k , H u m b l e , T X 01 M \b 3 8 0 0 0 . 0 0 \t 3 3 3 4 4
1520 5 5 5 5 01 5 S 00 00 12 05 J o y c e 01 A 07 E n g l i s h \t 4 5 3 4 5
1540 3 4 5 3 \n 1 9 7 2 - 0 7 - 3 1 16 5 6 3 1 R i c e , H o u s t
1560 o n , T X 01 F \b 2 5 0 0 0 . 0 0 \t 3 3 3 4 4 5 5 5 5 01 5 S 00 00
1580 13 05 A h m a d 01 V 06 J a b b a r \t 9 8 7 9 8 7 9 8 7 \n 1 9 6 9 -
15a0 0 3 - 2 9 17 9 8 0 D a l l a s , H o u s t o n , T X 01 M \b
15c0 2 5 0 0 0 . 0 0 \t 9 8 7 6 5 4 3 2 1 01 4 G 00 00 14 05 J a m e s 01 E
15e0 04 B o r g \t 8 8 8 6 6 5 5 5 5 \n 1 9 3 7 - 1 1 - 1 0 16 4 5 0 S
1600 t o n e , H o u s t o n , T X 01 M \b 5 5 0 0 0 . 0 0 û 01 1 01
1620 00 00 15 þ

Secure Shell

The term you selected is being presented by searchSecurity.com, a TechTarget site for Security professionals.

Secure Shell (SSH), sometimes known as Secure Socket Shell, is a Unix-based command interface and protocol for securely getting access to a remote computer. It is widely used by network administrators to control Web and other kinds of servers remotely. SSH is actually a suite of three utilities - slogin, ssh, and scp - that are secure versions of the earlier UNIX utilities, rlogin, rsh, and rcp. SSH commands are encrypted and secure in several ways. Both ends of the client/server connection are authenticated using a digital certificate, and passwords are protected by being encrypted.

SSH uses RSA public key cryptography for both connection and authentication. Encryption algorithms include Blowfish, DES, and IDEA. IDEA is the default.

SSH2, the latest version, is a proposed set of standards from the Internet Engineering Task Force (IETF).

>> Find products and vendors related to Secure Shell.

Read more about it:

>> Steve Acheson's Secure Shell (SSH) Frequently Asked Questions is recommended.
>> The developer of SSH is SSH Communications Security .
>> Rajpaul Bagga offers a Secure Shell (SSH) Howto .

WHAT'S NEW
on searchSecurity

1. Cisco Resource Center
2. Top 10 Clicks of the Week
3. Attend Data Center Decisions June 2-4
4. Subscribe to Info Security Magazine

Last updated on: Apr 22, 2003

<< Back to previous page Go to whatis.com home page >>

What's New ?

Sunopsis

Data Integration Software for the Real-Time Enterprise

Sunopsis is the only Data Integration software that harnesses existing investments in RDBMS and IT staff's current skills to deliver high-performance solutions under budget.

Learn More...

Select Language

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JDBCXML {
	public static void main(String[] args) {
		try {
			Class.forName("com.sunopsis.jdbc.driver.xml.SnpsXmlDriver");
		} catch (ClassNotFoundException cnfe) {
			System.err.println("Driver class not found");
			cnfe.printStackTrace();
		}
		Connection con = null;
		try {
			con = (Connection) DriverManager.getConnection("jdbc:snps:xml");
		} catch (SQLException e1) {
			System.err.println("Error establishing database connection");
			Throwable t = sqle;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}
		Statement stmt = null;
		try {
			stmt = con.createStatement();
		} catch (SQLException sqle) {
			Throwable t = sqle;
			while (t != null) {
			System.err.println("Type: " + t.getClass().getName());
		System.err.println("Message: " + t.getMessage());
		System.err.println("-----");
		t = t.getCause();
			}
		}
		try {
			stmt.execute("load file \"employee.xml\" on schema EMP readonly");
			stmt.execute("set schema EMP");
			ResultSet rs = con.getMetaData().getTables("", "EMP", "%", null);

			while (rs.next()) {
```

```
                System.out.println("TABLE=" + rs.getString("TABLE_NAME"));
                System.out.print("(");
                ResultSet colRS =
                    con.getMetaData().getColumns(
                        "",
                        "EMP",
                        rs.getString("TABLE_NAME"),
                        "%");
                while (colRS.next()) {
                    System.out.println(colRS.getString("COLUMN_NAME") + ",");
                }
                System.out.println(")\n");
            }
        } catch (SQLException sqle) {
            Throwable t = sqle;
            while (t != null) {
            System.err.println("Type: " + t.getClass().getName());
        System.err.println("Message: " + t.getMessage());
        System.err.println("-----");
        t = t.getCause();
            }
        }

        ResultSet rs = null;
        try {
            rs = (ResultSet) stmt.executeQuery("SELECT * FROM EMPLOYEE;");
            while (!rs.isLast()) {
                rs.next();
                System.out.println(rs.getString("FNAME"));
            }
        } catch (SQLException sqle) {
            Throwable t = sqle;
            while (t != null) {
            System.err.println("Type: " + t.getClass().getName());
        System.err.println("Message: " + t.getMessage());
        System.err.println("-----");
        t = t.getCause();
            }
        }

        try {
            stmt.executeUpdate("INSERT INTO EMPLOYEE (FNAME, MINIT, LNAME)
VALUES('James', 'E', 'Borg');");
        } catch (SQLException sqle) {
            Throwable t = sqle;
            while (t != null) {
```

```
                System.err.println("Type: " + t.getClass().getName());
            System.err.println("Message: " + t.getMessage());
            System.err.println("-----");
            t = t.getCause();
                }
            }

        }
}
```

TABLE=EMPLOYEES
(EMPLOYEESPK,
EMPLOYEESORDER,
)

TABLE=EMPLOYEE
(LNAMEORDER,
FNAME,
EMPLOYEEORDER,
LNAME,
MINITORDER,
EMPLOYEESFK,
FNAMEORDER,
MINIT,
)

Joyce
Ramesh
Jennnifer
Alicia
Franklin
John

Tutorials & Code Camps

Aids for Learning to Use the JDBC API

Printable Page

Educational Materials

Aids for Learning to Use the JDBC API

- Java Series book *JDBC API Tutorial and Reference, Third Edition*
  covers all of the JDBC 3.0 API, which includes the core API and the Optional Package API. It offers tutorials on basic features, advanced features, using metadata, and using rowsets. The reference section explains much more than is found in the specification and is the definitive reference for the JDBC API.

- Online Tutorials -- based on the tutorials in the Java Series book
  - Basic Tutorial
  - Advanced Tutorial
  - Rowset Tutorial

- Online Tutorial -- from the Java Developer Connection^{SM}

Trail: JDBC(TM) Database Access
Lesson: New Features in the JDBC 2.0 API

Inserting and Deleting Rows Programmatically

In the previous section you saw how to modify a column value using methods in the JDBC 2.0 API rather than having to use SQL commands. With the JDBC 2.0 API, you can also insert a new row into a table or delete an existing row programmatically.

Let's suppose that our coffee house proprietor is getting a new variety from one of his coffee suppliers, The High Ground, and wants to add the new coffee to his database. Using the JDBC 1.0 API, he would write code that passes an SQL insert statement to the DBMS. The following code fragment, in which `stmt` is a `Statement` object, shows this approach:

```
stmt.executeUpdate("INSERT INTO COFFEES " +
                   "VALUES ('Kona', 150, 10.99,
0, 0)");
```

You can do the same thing without using any SQL commands by using `ResultSet` methods in the JDBC 2.0 API. Basically, after you have a `ResultSet` object with results from the table `COFFEES` , you can build the new row and then insert it into both the result set and the table `COFFEES` in one step. You build a new row in what is called the insert row, a special row associated with every `ResultSet` object. This row is not actually part of the result set; you can think of it as a separate buffer in which to compose a new row.

Your first step will be to move the cursor to the insert row, which you do by invoking the method `moveToInsertRow` . The next step is to set a value for each column in the row. You do this by calling the appropriate `updateXXX` method for each value. Note that these are the same `updateXXX` methods you used in the previous section for changing a column value. Finally, you call the method `insertRow` to insert the row you have just populated with values into the result set. This one method simultaneously inserts the row into both the `ResultSet` object and the database table from which the result set was selected.

The following code fragment creates the scrollable and updatable `ResultSet` object `uprs` , which contains all of the rows and columns in the table `COFFEES` :

```
Connection con = DriverManager.getConnection
("jdbc:mySubprotocol:mySubName");
Statement stmt = con.createStatement(ResultSet.
TYPE_SCROLL_SENSITIVE,
                                     ResultSet.
CONCUR_UPDATABLE);
ResultSet uprs = stmt.executeQuery("SELECT *
FROM COFFEES");
```

The next code fragment uses the `ResultSet` object `uprs` to insert the row for Kona coffee, shown in the SQL code example. It moves the cursor to the insert row, sets the five column values, and inserts the new row into `uprs` and `COFFEES` :

```
uprs.moveToInsertRow();
uprs.updateString("COF_NAME", "Kona");
uprs.updateInt("SUP_ID", 150);
uprs.updateFloat("PRICE", 10.99);
uprs.updateInt("SALES", 0);
uprs.updateInt("TOTAL", 0);
uprs.insertRow();
```

Because you can use either the column name or the column number to indicate the column to be set, your code for setting the column values could also have looked like this:

```
uprs.updateString(1, "Kona");
uprs.updateInt(2, 150);
uprs.updateFloat(3, 10.99);
uprs.updateInt(4, 0);
uprs.updateInt(5, 0);
```

You might be wondering why the `updateXXX` methods seem to behave differently here from the way they behaved in the update examples. In those examples, the value set with an `updateXXX` method immediately replaced the column value in the result set. That was true because the cursor was on a row in the result set. When the cursor is on the insert row, the value set with an `updateXXX` method is likewise immediately set, but it is set in the insert row rather than in the result set itself. In both updates and insertions, calling an `updateXXX` method does not affect the underlying database table. The method `updateRow` must be called to have updates occur in the database. For insertions, the method `insertRow` inserts the new row into the result set and the database at the same time.

You might also wonder what happens if you insert a row but do not supply a value for every column in the row. If you fail to supply a value for a column that was defined to accept SQL `NULL` values, then the value assigned to that column is `NULL` . If a column does not accept null values, however, you will get an `SQLException` when you do not call an `updateXXX` method to set a value for it. This is also true if a table column is missing in your `ResultSet` object. In the example above, the query was `SELECT * FROM COFFEES` , which produced a result set with all the columns of all the rows. When you want to insert one or more rows, your query does not have to select all rows, but it is safer to select all columns. Especially if your table has hundreds or thousands of rows, you might want to use a `WHERE` clause to limit the number of rows returned by your `SELECT` statement.

After you have called the method `insertRow` , you can start building another row to be inserted, or you can move the cursor back to a result set row. You can, for instance, invoke any of the methods that put the cursor on a specific row, such as `first` , `last` , `beforeFirst` , `afterLast` , and `absolute` . You can also use the methods `previous` , `relative` , and `moveToCurrentRow` . Note that you can invoke `moveToCurrentRow` only when the cursor is on the insert row.

When you call the method `moveToInsertRow` , the result set records which row the cursor is sitting on, which is by definition the current row. As a consequence, the method `moveToCurrentRow` can move the cursor from the insert row back to the row that was previously the current row. This also explains why you can use the methods `previous` and `relative` , which require movement relative to the current row.

Start of Tutorial > Start of Trail > Start of Lesson

Search
Feedback Form

JDBC FAQ Home Page

View: What's New GO

ASK A QUESTION

FAQ Manager is guru **Joe Sam Shirah** PREMIUM.

Java Database Connectivity is the standard for communication between a Java application and a relational database. The JDBC API is released in two versions; JDBC version 1.22 (released with JDK 1.1.X in package java.sql) and version 2.0 (released with Java platform 2 in packages java.sql and javax.sql). It is a simple and powerful largely database-independent way of extracting and inserting data to or from any database. [FAQ Previously managed by Lennart Jorelid.]

What's New

- **What is the best way to generate a universally unique object ID? Do I need to use an external resource like a file or database, or can I do it all in memory?**
  Languages:Markup:XML, **Java:API:Servlets**, **Java:API:JDBC**, **Java:API:EJB:EntityBean:Primary Keys**, **Process:Patterns:BestPractices**
  Alessandro A. Garbagnati PREMIUM, Nov 25, 2002
  [I need to generate unique id's that will be used for node 'ID' attribute values within XML documents. This id must be unique system-wide. The generator...

- **Whan happens when I close a Connection application obtained from a connection Pool? How does a connection pool maintain the Connections that I had closed through the application?**
  Java:API:JDBC:Connections
  Christopher Koenigsberg PREMIUM, Apr 14, 2002
  It is the magic of polymorphism, and of Java interface vs. implementation types. Two objects can both be "instanceof" the same interface type,...

- **How can I know when I reach the last record in a table, since JDBC doesn't provide an EOF method?**
  Java:API:JDBC:ResultSets
  Yusuf Dönmez, Mar 23, 2002
  You can use last() method of java.sql.ResultSet, if you make it scrollable. Joe Sam Shirah adds: You can also use isLast() as you are reading the...

Related Links

JDBC Forum

Sun's JDBC pages

JDBC 2.0 optional package

- **Problem with getDouble() We use the Java method getDouble() to get numeric values. Up to now it worked properly but suddenly we got false values, like 49066.429000000004 for the true value 49066.429....**
  Java:API:JDBC:Data Types
  Joe Sam Shirah PREMIUM, Mar 23, 2002
  First, let me assure you that something has changed, whether driver, database column definition, or database update. Even so, you were just lucky in the...

- **How can I correctly parse CSV (comma separated values) files? StringTokenizer doesn't seem to fit many conditions.**
  Java:API:JDBC, **Java:API:IO:Tokenizing**, **Java:API:IO:Regular expressions**
  Joe Sam Shirah PREMIUM, Mar 23, 2002
  Ian Darwin has two classes (CSV.java and CSVRE.java) to handle CSV files in his Java Cookbook, including a way with regular expressions. You can download...

- **Where can I find info, frameworks and example source for writing a JDBC driver?**
  Java:API:JDBC:Drivers
  Joe Sam Shirah PREMIUM, Mar 23, 2002
  There a several drivers with source available, like MM.MySQL, SimpleText Database, FreeTDS, and RmiJdbc. There is at least one free framework, the j...

- **Is any JDBC driver available that can access EDI messages?**
  Java:API:JDBC:Design, Implementation, and Performance, **Java:API:JDBC: Drivers**
  Laurent Mihalkovic PREMIUM, Mar 23, 2002
  I don't know about existing drivers, but JavaPro had a good article on writing your own driver. The example was a driver to access XML files. check it...

- **How can I create a custom RowSetMetaData object from scratch?**
  Java:API:JDBC:2.0
  Joe Sam Shirah PREMIUM, Feb 28, 2002
  One unfortunate aspect of RowSetMetaData for custom versions is that it is an interface. This means that implementations almost have to be proprietary....

- **How does a custom RowSetReader get called from a CachedRowSet?**
  Java:API:JDBC:2.0
  Joe Sam Shirah PREMIUM, Feb 28, 2002
  The Reader must be registered with the CachedRowSet using CachedRowSet.setReader (javax.sql.RowSetReader reader). Once that is done, a call to CachedR...

- **How do I implement a RowSetReader? I want to populate a CachedRowSet myself and the documents specify that a RowSetReader should be used. The single method accepts a RowSetInternal caller and returns...**
  Java:API:JDBC:2.0
  Joe Sam Shirah PREMIUM, Feb 28, 2002
  The documentation says "It can be implemented in a wide variety of ways..." and is pretty vague about what can actually be done. In general, readData()...

- **How can I instantiate and load a new CachedRowSet object from a non-JDBC source?**
  Java:API:JDBC:2.0
  Joe Sam Shirah PREMIUM, Feb 28, 2002
  The basics are: Create an object that implements javax.sql.RowSetReader, which loads the data. Instantiate a CachedRowset object. Set the...

- **Can I set up a conection pool with multiple user IDs? The single ID we are forced to use causes probelems when debugging the DBMS.**
  Java:API:JDBC:Connections
  Joe Sam Shirah PREMIUM, Feb 26, 2002
  Since the Connection interface (and the underlying DBMS) requires a specific user and password, there's not much of a way around this in a pool. While...

- **How can I protect my database password ? I'm writing a client-side java application that will access a database over the internet. I have concerns about the security of the database passwords. The client...**
  Java:API:JDBC:Design, Implementation, and Performance
  Jay Meyer, Feb 26, 2002
  This is a very common question. I answered a similiar question at Remote database over internet + Java Swing app with JDBC != secure? Conclusion:...

- **Detecting Duplicate Keys I have a program that inserts rows in a table. My table has a column 'Name' that has a unique constraint. If the user attempts to insert a duplicate name into the table, I want...**
  Java:API:JDBC:Exceptions and Warnings
  JIA Java Italian Association PREMIUM, Feb 26, 2002
  A solution that is perfectly portable to all databases, is to execute a query for checking if that unique value is present before inserting the row. The...

- **What driver should I use for scalable Oracle JDBC applications?**
  Java:API:JDBC:DBMS/Product Specific
  Joe Sam Shirah PREMIUM, Feb 16, 2002
  Sun recommends using the thin (type 4) driver. On single processor machines to avoid JNI overhead. On multiple processor machines, especially...

- **Can you scroll a result set returned from a stored procedure? I am returning a result set from a stored procedure with type SQLRPGLE but once I reach the end of the result set it does not allow repositioning....**
  Java:API:JDBC:ResultSets, **Java:API:JDBC:Stored Procedures**
  Joe Sam Shirah PREMIUM, Feb 16, 2002
  A CallableStatement is no different than other Statements in regard to whether related ResultSets are scrollable. You should create the CallableStatement...

- **Driver memory problem I am using interclient driver to connect to interbase. My program does lots of inserts, updates, selects, etc. The problem I found is that if I create a statement for each operation...**
  Java:API:JDBC:Statements
  Joe Sam Shirah PREMIUM, Feb 16, 2002
  Regardless of DBMS, there's no particular reason to close a Statement until you are done with your operations - assuming 1) a JDBC compliant driver works...

- **How do I write Greek (or other non-ASCII/8859-1) characters to a database?**
  Java:API:JDBC:Design, Implementation, and Performance
  Joe Sam Shirah PREMIUM, Jan 31, 2002
  From the standard JDBC perspective, there is no difference between ASCII/8859-1 characters and those above 255 (hex FF). The reason for that is that...

- **How can I insert images into a Mysql database?**
  Java:API:JDBC:DBMS/Product Specific
  Kasi Mono, Jan 31, 2002
  This code snippet shows the basics: File file = new File(fPICTURE); FileInputStream fis = new FileInputStream(file); PreparedStatement ps = ...

- **I'd like to evaluate the Object/Relational Mapping approach and products. Any discussion is helpful.**
  Java:API:JDBC:Design, Implementation, and Performance
  Matt Goodall, Jan 31, 2002
  jguru Disclaimer: The following views and opinions are entirely those of the respondents and provided for our readers to consider. jGuru is not in the...

« previous beginning **next »**

O'Reilly Home
Press Room
Jobs

Resource Centers
Perl
Java
Python
C/C++
Scripting
Web
Digital Media
Web Services
XML
Oracle
SysAdm/Networking
Security
Databases
Linux/Unix
Macintosh/OS X
Windows
.NET
Open Source
Wireless
Bioinformatics
Enterprise Development

Book Series
Hacks
Head First
Cookbooks
In a Nutshell
CD Bookshelves
Pocket References
The Missing Manuals
Developer's Notebooks

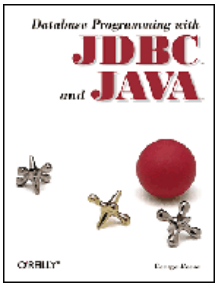
Publishing Partners
No Starch Press
Paraglyph Press
Pragmatic Bookshelf
Syngress Publishing

Online Publications
LinuxDevCenter.com
MacDevCenter.com
WindowsDevCenter.com
ONDotnet.com
ONJava.com
ONLamp.com
OpenP2P.com
Perl.com
WebServices.XML.com
XML.com

Special Interest
Events
Meerkat News
Ask Tim
tim.oreilly.com
From the Editors List
Letters
Beta Chapters
Newsletters
Open Books

Special Sales
Academic
Corporate
Government

Inside O'Reilly
About O'Reilly



see larger cover

Table of Contents
Index
Errata
Sample Chapter
Examples
Java Persistence Library
Colophon

Register your book to get email notification of new editions, special offers, and more.

Database Programming with JDBC and Java

By George Reese
1st Edition June 1997
ISBN: 1-56592-270-0

This book has been updated--the edition you're requesting is out of print. Please visit the catalog page of the latest edition.

The latest edition is also available on Safari Bookshelf.

Buy from O'Reilly: View Cart

This book describes the standard Java interfaces that make portable object-oriented access to relational databases possible and offers a robust model for writing applications that are easy to maintain. It introduces the JDBC and RMI packages and includes a set of patterns that separate the functions of the Java application and facilitate the growth and maintenance of an application. *Database Programming with JDBC and Java* recently won a Reader's Choice Special Mention Award from *Visual Basic Programmers Journal*. It was the only winner in the category of Java publications. [Full Description]

Sample Chapter and Code Examples

Chapter 4: Database Access Through JDBC

Download the code examples from this book. The complete set of examples is available at: http://examples.oreilly.com/javadata/

Related Books

- Developing Java Beans (O'Reilly)
- Java AWT Reference (O'Reilly)
- Java Cryptography (O'Reilly)
- Java Distributed Computing (O'Reilly)
- Java Examples in a Nutshell, 3rd Edition (O'Reilly)
- Java Fundamental Classes Reference (O'Reilly)
- Java in a Nutshell, 4th Edition (O'Reilly)
- Java Language Reference, 2nd Edition (O'Reilly)
- Java Network Programming (O'Reilly)
- Java Security, 2nd Edition (O'Reilly)
- Java Servlet Programming, 2nd Edition (O'Reilly)
- Java Threads, 2nd Edition (O'Reilly)
- Java Virtual Machine (O'Reilly)
- JavaScript: The Definitive Guide, 4th Edition (O'Reilly)
- Learning Java, 2nd Edition (O'Reilly)
- Netscape IFC in a Nutshell (O'Reilly)

O'Reilly Newsletters

Stay informed. Sign up to receive announcements about O'Reilly products and news.

SQL Access Group's Call-Level Interface

An independent interface for database development

Roger Sippl

Roger is chairman of the SQL Access Group and founder of Visigenic Software. He can be contacted at Visigenic Software, 951 Mariner's Island Blvd., Suite 460, San Mateo, CA 94404, (415) 286-1900.

The SQL Access Group (SAG) was formed in 1989 to define and promote standards for database portability and interoperability. Initially, the membership roster included database heavyweights Oracle, Informix, and Ingres (now Computer Associates), as well as hardware vendors Digital, Hewlett-Packard, Tandem, and Sun. Table 1 lists the current membership roster. The group's initial projects involved developing the draft ISO/ANSI standard for SQL (including the embedded-SQL interface specifications) and a specification for remote data access (RDA).

In 1990, SAG took the lead in developing an SQL-based Call Level Interface (CLI). The CLI SAG is an API for database access, offering an alternative invocation technique to embedded SQL

that provides essentially equivalent operations. SAG envisioned an interface that would enable client/server applications to access data stored in heterogeneous relational and nonrelational databases. The interface would be platform, vendor, database, and language neutral. SAG and X/Open published the CLI Snapshot Specification in 1992 as a "work in progress," and it was adopted for use in commercial software products.

Microsoft helped define the X/Open CLI specification and became the first company to commercialize the CLI specification by shipping Open Database Connectivity (ODBC) 1.0 for Windows in 1992. To create ODBC, Microsoft extended the CLI specification and created a three-layer specification in which the "core" layer corresponds to the SAG CLI. Over the next two years, the CLI specification underwent several transformations, reemerging in 1994 as an X/Open Preliminary Specification. Also in 1994, Microsoft released ODBC 2.0, whose core functionality was still aligned with the SAG CLI. Earlier this year, Microsoft announced that ODBC 3.0 (to be released in 1996) will be fully aligned with both ISO's CLI standard and SAG's CLI Specification.

In March 1995, the CLI was finalized and published as an X/Open Common Application Environment (CAE) specification. CAE specifications are adopted by consensus and are the basis against which suppliers brand their products. The adoption and publication of the CLI as an X/Open CAE specification represents the culmination of five years of cooperative development work within the SQL Access Group.

In the meantime, ODBC has gained broad support in the database industry. All major software vendors--Oracle, Sybase, Informix, Computer Associates, IBM, Gupta, Powersoft, and Borland, as well as more than 140 application-software developers and VARs--have added ODBC support. Many of these vendors also offer proprietary APIs; nonetheless, they see the X/Open CLI specification and ODBC as important to their strategy. In 1994, Microsoft granted an exclusive source-code license to Visigenic Software for porting and licensing the ODBC SDK to non-Windows platforms. As a result, the ODBC SDK is now also available on all major UNIX platforms, as well as OS/2 and Macintosh.

A Closer Look at the SAG CLI

The X/Open CLI specification is a standard API for database access that is vendor, platform, and database neutral. It defines a set of functions that a program can call directly using normal function-call facilities. The specification is language independent and includes header files for both C and Cobol.

The CLI specification defines 57 functions that support a rich set of database-access operations sufficient for creating robust database applications, including:

- Allocating and deallocating handles (eight calls).
- Getting and setting attributes (ten calls).
- Opening and closing database connections (two calls).
- Accessing descriptors (six calls).
- Executing SQL statements (nine calls).
- Retrieving results (eight calls).
- Accessing schema metadata (four calls).
- Performing introspection (four calls).
- Controlling transactions (two calls).
- Accessing diagnostic information (three calls).
- Canceling functions (one call).

A database application calls these functions for all interactions with a database. The CLI enables applications to establish multiple database connections simultaneously and to process multiple statements simultaneously, depending on the capabilities of the database servers being accessed. Figure 1 and Figure 2 show the basic control flow for using the CLI functions.

The X/Open CLI specification was developed with client/server architectures in mind. In fact, the CLI is ideal for this environment, in which the developer often knows little (if anything) about the database at the time the application is written. The X/Open CLI specification defines a set of introspection functions that enable an application to discover the characteristics and capabilities of a particular CLI implementation and of any database server accessed through that implementation. For example, *SQLGetTypeInfo* lets you find out what data types are supported by a particular server, and *SQLDataSources* returns a list of available database servers and descriptions. Introspection functions facilitate a technique known as "adaptive programming," whereby an application adapts its behavior at run time to take advantage of the capabilities of a particular database environment.

Processing SQL Statements

The X/Open CLI specification, including sample programs and header files, is available directly from X/Open. Listings One, Two, and Three illustrate how the CLI works. Listing One is from a typical database application. Listings Two and Three are the significant portions of functions called by Listing One.

The application allocates memory for an environment handle and a connection handle; both are required to establish a database connection. The *SQLConnect* call establishes the database connection, specifying the server name (*server_name*), user id (*uid*), and password (*pwd*). The application then allocates memory for a statement handle and calls *SQLExecDirect*, which both prepares and executes an SQL statement. The *SQLGetDiagField* call takes advantage of the CLI's ability to interrogate the database server. In this case, it returns a code describing the nature of the SQL statement just executed. With this information in hand, the application calls the user-defined *DoSelect()* function to process the results of the SQL statement. Finally, the program frees the statement handle, disconnects from the database, and frees the memory previously allocated for the connection and environment handles.

The body of the *DoStatement()* function in Listing Two is built around a switch statement that processes the results of an SQL statement based on the return value from the *SQLGetDiagField* call in Listing One. In the case of a SELECT statement, which requires its own complex processing, the function calls another user-defined function, *DoSelect()*; see Listing Three. In the case of an UPDATE, DELETE, or INSERT statement, the *DoStatement()* function calls the CLI diagnostic function *SQLGetDiagField* to find out how many rows were affected by the statement, then calls *SQLEndTran* to commit the transaction. The function prints one message indicating whether the commit was successful and another giving the number of affected rows.

In the case of any Data Definition Language (DDL) statement, the *DoStatement()* function first calls the CLI introspection function *SQLGetInfo* to find out if the CLI implementation being used supports transaction processing for DDL statements. If so, the function calls *SQLEndTran* to commit the transaction, then prints a message indicating whether the commit was successful.

The *DoSelect()* function in Listing Three processes the results of an SQL SELECT statement. It is called by the *DoStatement()* function. First, *DoSelect()* calls *SQLNumResultCols* to determine how many columns are in the result set. Then, for each column in the result set, the function calls *SQLDescribeCol* to get descriptive information about the column (that is, its length, scale, and data type), prints an appropriate row of column headings to the standard output, allocates memory to bind the results, and calls *SQLBindCol* to establish the bindings. Next, *DoSelect()* calls *SQLFetch* to fetch rows from the result set until none are left. For each row, the *DoSelect()* function prints the column values, followed by a new line. The CLI can provide various types of diagnostic information, such as whether a value is truncated or null; *DoSelect()* tests for these two conditions and, when they occur, calls the user-defined *BuildMessage()* function (not shown) to generate appropriate error messages. At the end, the function prints a list of any such error messages. Finally, the application closes the cursor for the statement handle and frees the data buffers.

What's Next?

The SQL Access Group formally merged with X/Open in 1995. The charters of SAG and X/Open's Data Management Working Group were essentially the same, and X/Open had always edited and published the work of the SQL Access Group. The merger made it possible to eliminate duplicate efforts, reduce costs, and unify development efforts. The X/Open Data Management Technical Committee disbanded, and the X/Open SQL Access Group, now functioning within the X/Open Technical Program, has assumed all of its responsibilities.

SAG is working in close cooperation with ISO on its upcoming SQL CLI specification, which is intended to mirror the X/Open specification. SAG is also actively pursuing the next logical step for the CLI--the development of an X/Open test suite for CLI conformance. Such a test suite will enable developers to verify conformance to the CLI Specification. Development of the test suite should be well underway by the time this article is published.

SAG has already begun work on the next version of the CLI specification. Our mission is to extend and refine the CLI for even more successful interoperability and to define standards that incorporate newer database technologies, including XA transaction processing, stored procedures, BLOBs, triggers, and asynchronous calls.

The efforts of the major standards organizations, including ISO, ANSI, and X/Open, as well as

the strategies of all the major players in the database industry, now incorporate the X/Open CLI Specification. Vendors and standards organizations are moving rapidly in the same direction, a direction the marketplace has already validated.

Figure 1: Basic CLI control flow.

Figure 2: CLI control flow for processing SQL statements.

Table 1: SQL Access Group members.

AT&T
Borland International
Computer Associates
Fulcrum Technologies
Hitachi
IBM
Information Builders
Informix Software
INTERSOLV
Microsoft
Oracle
Progress Software
Sybase
Visigenic Software

Listing One

```
/* allocate an environment handle */
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
/* allocate a connection handle */
SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
/* connect to database */
if (SQLConnect(hdbc, server_name, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS) !=
        SQL_SUCCESS)
    return(PrintErr(SQL_HANDLE_DBC, hdbc));
/* allocate a statement handle */
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
/* execute the SQL statement */
if (SQLExecDirect(hstmt, sqlstr, SQL_NTS) != SQL_SUCCESS)
    return(PrintErr(SQL_HANDLE_STMT, hstmt));
/* see what kind of statement it was */
SQLGetDiagField(SQL_HANDLE_STMT, hstmt, 0, SQL_DIAG_DYNAMIC_FUNCTION_CODE,
        (SQLPOINTER)&stmttype, 0, (SQLSMALLINT *)NULL);
/* process the SQL statement */
DoStatement(stmttype);
/* free statement handle */
SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
/* disconnect from database */
SQLDisconnect(hdbc);
/* free connection handle */
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
/* free environment handle */
SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

Listing Two

```
switch(stmttype) {
    /* SELECT statement */
    case SQL_DIAG_SELECT_CURSOR:
        DoSelect();
        break;
    /* searched UPDATE, searched DELETE, or INSERT statement */
    case SQL_DIAG_UPDATE_WHERE:
    case SQL_DIAG_DELETE_WHERE:
    case SQL_DIAG_INSERT:
        /* get row count */
        SQLGetDiagField(SQL_HANDLE_STMT, hstmt, 0, SQL_DIAG_ROW_COUNT,
            (SQL_POINTER)&rowcount, 0, (SQLSMALLINT *)NULL);
        if (SQLEndTran(SQL_HANDLE_ENV, henv, SQL_COMMIT) == SQL_SUCCESS)
```

```
            printf("Operation successful\n");
        else
            printf("Operation failed\n");
        printf("%ld rows affected\n", rowcount);
        break;
    /* other statements */
    case SQL_DIAG_ALTER_TABLE:
    case SQL_DIAG_CREATE_INDEX:
    case SQL_DIAG_CREATE_TABLE:
    case SQL_DIAG_CREATE_VIEW:
    case SQL_DIAG_DROP_INDEX:
    case SQL_DIAG_DROP_TABLE:
    case SQL_DIAG_DROP_VIEW:
    case SQL_DIAG_GRANT:
    case SQL_DIAG_REVOKE:
        SQLGetInfo(hdbc, SQL_TXN_CAPABLE, &txn_type, 0, 0);
        if(txn_type == SQL_TC_ALL) {
            if (SQLEndTran(SQL_HANDLE_ENV, henv, SQL_COMMIT) ==
                    SQL_SUCCESS)
                printf("Operation successful\n");
            else
                printf("Operation failed\n");
        }
        break;
    /* other implementation-defined statements */
    default:
        printf("Statement type=%ld\n", stmttype);
        break;
}
```

Listing Three

```
/* determine number of result columns */
SQLNumResultCols(hstmt, &nresultcols);
/* display column names */
for (i=0; i<nresultcols; i++) {
    SQLDescribeCol(hstmt, i+1, colname, sizeof(colname), &colnamelen,
            &coltype, collen[i], &scale, &nullable);
    /* user-defined function to get the display length for the data type */
    collen[i] = DisplayLength(coltype, collen[i], colname);
    printf("%*.*s", collen[i], collen[i], colname);
    /* allocate memory to bind column */
    data[i] = (SQLCHAR *) malloc(collen[i]+1);
    /* bind columns to program vars, converting all types to CHAR */
    SQLBindCol(hstmt, i+1, SQL_CHAR, data[i], collen[i]+1, &outlen[i]);
}
printf("\n");
/*display result rows */
while (SQL_SUCCEEDED(rc=SQLFetch(hstmt))) {
    errmsg[0] = '\0';
    for (i=0; i<nresultcols; i++)  {
        if (outlen[i] == SQL_NULL_DATA || outlen[i] >= collen[i])
            /* set data text to "NULL" or add to errmsg */
            BuildMessage(errmsg, (SQLPOINTER *)&data[i], collen[i],
                    &outlen[i], i);
        printf("%*,*s ", outlen[i], outlen[i], data[i]);
    } /* for all columns in this row */
    /* print any accumulated error messages and new line */
    printf("%s\n", errmsg);
} /* while rows to fetch */
SQLCloseCursor(hstmt);
/* free data buffers */
for (i=0; i<nresultcols; i++) {
    free(data[i]);
}
End Listings
```

MarketPlace

Telecom DB Software Tech Guide -- for Developers

Learn developer strategies Cisco, Motorola & Lucent use to deliver higher performing products faster, at low TCO. Free guide features Berkeley DB, incl. support for ACID transactions, recovery, replication for carrier class apps to mobile devices.

Journyx Timesheet is Free for 10 users

Increase your consulting billings 20% by automating billing with this FREE 100% web-based timesheet for Linux-Wintel-Solaris-AIX-FreeBSD. SOAP XML API. Which 25% of your projects are unprofitable? Journyx will tell you.

MAKE a QUANTUM LEAP in Java Speed and Productivity

Build Web apps up to 10 times faster.Get down deep into expert-level coding-your way, with complete control.Build better quality and higher performance from the start.Get your FREE Java" Developer Resource Kit. Free trial-Borland®JBuilder® X Developer

Gain Useful VS .NET Skills at VSLive! New York

VSLive! returns to New York City with 65+hours of sessions/workshops that cover the issues you are wrestling with:scalability, security, and modern approaches including design patterns, Web services, defensive coding.Register by June9th-save $200+

Register for the Free TechNet Flash newsletter.

Get all of the latest Microsoft® IT news and information delivered right to your in-box when you register for the FREE TechNet Flash. This bi-weekly newsletter is designed to help IT professionals stay up-to-date on the latest Microsoft products, technologies and events.

Wanna see your ad here?

SPONSORED LINKS

Travel: new york hotels • Hotels • Las Vegas Hotels • Popular Hotels | Finance: Secured Loans • Cover Letter • Personal Loans | Electronics Canon digital camera • Consumer Electronics • Digital Cameras • Digital Camera • Sony Digital Camera • Camcorders • Inkjet Cartridges - Printer Ink Cartridges • Tape Drives and Tape Libraries | Miscellaneous: Shoes store online • Systems Management Software • Hiking shoes & boots •

Tutorials & Code Camps

Java Beans, Pt. 1: Definition: What is a Bean?

Printable Page

Training Index

Definition: What is a Bean?

by Greg Voss

[Introducing Java Beans] [Reusable Software Components]
[Application Builder Tools] [Basic Bean Concepts]

If you have used Delphi, or Visual Basic, you are already familiar with the notion of a bean. The idea is the same; the programming language is different. A Java Bean is a reusable software component that works with Java. More specifically: a Java Bean is a reusable software component that can be visually manipulated in builder tools.

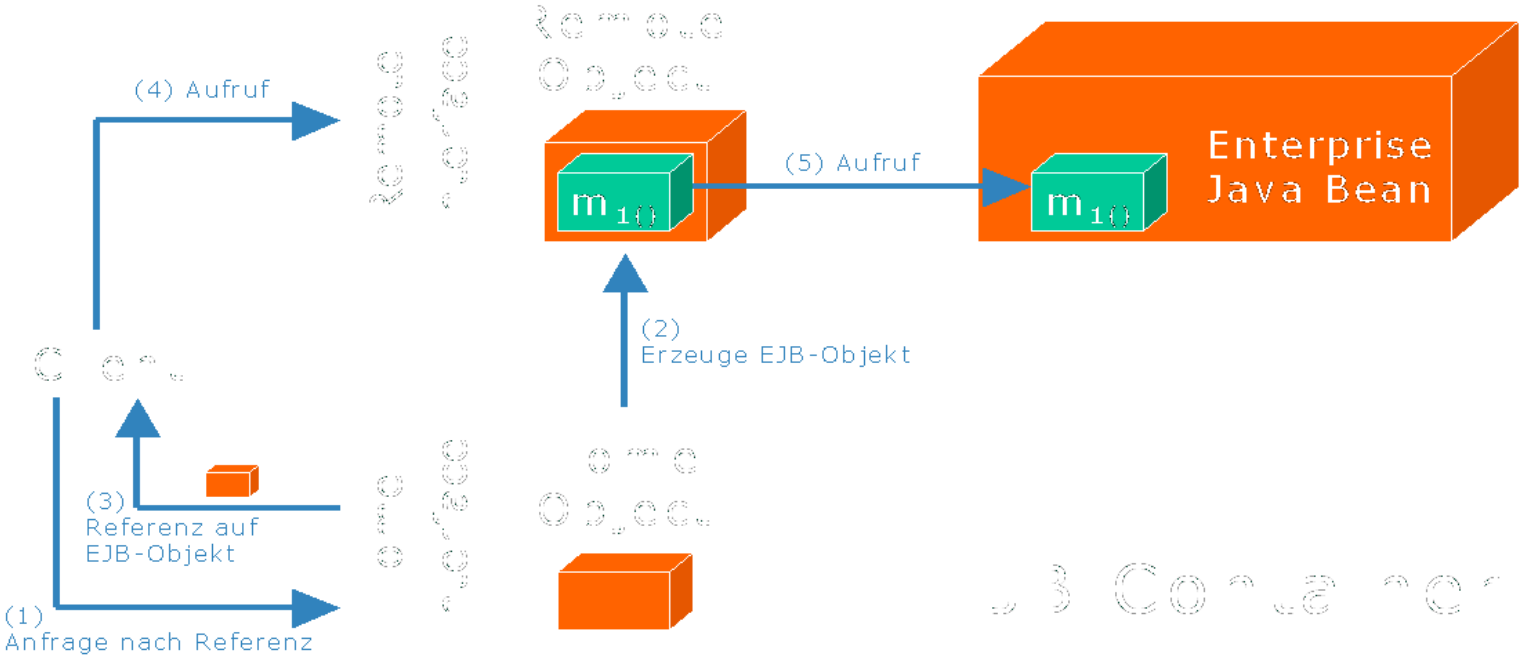
Definition:

A Java Bean is a reusable software component that can be visually manipulated in builder tools.

To understand the precise meaning of this definition of a Bean, clarification is required for the following terms:

- Software component
- Builder tool
- Visual manipulation

Each of these will be addressed in turn.


(4) Aufruf
(5) Aufruf
Enterprise
Java Bean
(2)
Erzeuge EJB-Objekt
(3)
Referenz auf
EJB-Objekt
(1)
Anfrage nach Referenz

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface SayHelloHome extends EJBHome {
	public SayHello create() throws RemoteException, CreateException;
}
```

```
import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface SayHello extends EJBObject {
	public String sayHello(String name) throws RemoteException;
}
```

```
import java.rmi.RemoteException;
import java.util.Date;
import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class HelloWorldBean implements SessionBean {
	public HelloWorldBean() {
	}

	public String sayHello(String name) {
		return ("Hello " + name + " it is now " + new Date().toString());
	}

	public void setSessionContext(SessionContext arg0)
		throws EJBException, RemoteException {
	}
	public void ejbRemove() throws EJBException, RemoteException {
	}

	public void ejbCreate() throws EJBException {
	}

	public void ejbActivate() throws EJBException, RemoteException {
	}

	public void ejbPassivate() throws EJBException, RemoteException {
	}
}
```

java.io

Interface Serializable

All Known Subinterfaces:

AdapterActivator, Attribute, Attribute, Attributes, BindingIterator, ClientRequestInfo, ClientRequestInterceptor, Codec, CodecFactory, Control, Current, Current, Current, CustomValue, DataInputStream, DataOutputStream, DHPrivateKey, DHPublicKey, DocAttribute, DomainManager, DSAPrivateKey, DSAPublicKey, DynAny, DynAnyFactory, DynArray, DynEnum, DynFixed, DynSequence, DynStruct, DynUnion, DynValue, DynValueBox, DynValueCommon, ExtendedRequest, ExtendedResponse, Externalizable, IdAssignmentPolicy, IDLEntity, IDLType, IdUniquenessPolicy, ImplicitActivationPolicy, Interceptor, IORInfo, IORInterceptor, IRObject, Key, LifespanPolicy, Name, NamingContext, NamingContextExt, ORBInitializer, ORBInitInfo, PBEKey, POA, POAManager, Policy, PolicyFactory, PrintJobAttribute, PrintRequestAttribute, PrintServiceAttribute, PrivateKey, PublicKey, RemoteRef, RequestInfo, RequestProcessingPolicy, RSAMultiPrimePrivateCrtKey, RSAPrivateCrtKey, RSAPrivateKey, RSAPublicKey, RunTime, SecretKey, ServantActivator, ServantLocator, ServantManager, ServantRetentionPolicy, ServerRef, ServerRequestInfo, ServerRequestInterceptor, StreamableValue, SupportedValuesAttribute, ThreadPolicy, UnsolicitedNotification, ValueBase

---

public interface **Serializable**

Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable.

To allow subtypes of non-serializable classes to be serialized, the subtype may assume responsibility for saving and restoring the state of the supertype's public, protected, and (if accessible) package fields. The subtype may assume this responsibility only if the class it extends has an accessible no-arg constructor to initialize the class's state. It is an error to declare a class Serializable if this is not the case. The error will be detected at runtime.

During deserialization, the fields of non-serializable classes will be initialized using the public or

protected no-arg constructor of the class. A no-arg constructor must be accessible to the subclass that is serializable. The fields of serializable subclasses will be restored from the stream.

When traversing a graph, an object may be encountered that does not support the Serializable interface. In this case the NotSerializableException will be thrown and will identify the class of the non-serializable object.

Classes that require special handling during the serialization and deserialization process must implement special methods with these exact signatures:

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
```

The writeObject method is responsible for writing the state of the object for its particular class so that the corresponding readObject method can restore it. The default mechanism for saving the Object's fields can be invoked by calling out.defaultWriteObject. The method does not need to concern itself with the state belonging to its superclasses or subclasses. State is saved by writing the individual fields to the ObjectOutputStream using the writeObject method or by using the methods for primitive data types supported by DataOutput.

The readObject method is responsible for reading from the stream and restoring the classes fields. It may call in.defaultReadObject to invoke the default mechanism for restoring the object's non-static and non-transient fields. The defaultReadObject method uses information in the stream to assign the fields of the object saved in the stream with the correspondingly named fields in the current object. This handles the case when the class has evolved to add new fields. The method does not need to concern itself with the state belonging to its superclasses or subclasses. State is saved by writing the individual fields to the ObjectOutputStream using the writeObject method or by using the methods for primitive data types supported by DataOutput.

Serializable classes that need to designate an alternative object to be used when writing an object to the stream should implement this special method with the exact signature:

```
 ANY-ACCESS-MODIFIER Object writeReplace() throws
ObjectStreamException;
```

This writeReplace method is invoked by serialization if the method exists and it would be accessible from a method defined within the class of the object being serialized. Thus, the method can have private, protected and package-private access. Subclass access to this method follows java accessibility rules.

Classes that need to designate a replacement when an instance of it is read from the stream should

implement this special method with the exact signature.

```
 ANY-ACCESS-MODIFIER Object readResolve() throws
ObjectStreamException;
```

This readResolve method follows the same invocation rules and accessibility rules as writeReplace.

Since:

JDK1.1

See Also:

`ObjectOutputStream`, `ObjectInputStream`, `ObjectOutput`, `ObjectInput`, `Externalizable`

Submit a bug or feature

For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

java.rmi

Interface Remote

All Known Subinterfaces:

ActivationInstantiator, ActivationMonitor, ActivationSystem, Activator, DGC, Registry

All Known Implementing Classes:

_Remote_Stub, ActivationGroup, ActivationGroup_Stub, RemoteObject

---

public interface **Remote**

The `Remote` interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine. Any object that is a remote object must directly or indirectly implement this interface. Only those methods specified in a "remote interface", an interface that extends `java.rmi.Remote` are available remotely.

Implementation classes can implement any number of remote interfaces and can extend other remote implementation classes. RMI provides some convenience classes that remote object implementations can extend which facilitate remote object creation. These classes are `java.rmi.server.UnicastRemoteObject` and `java.rmi.activation.Activatable`.

For complete details on RMI, see the RMI Specification which describes the RMI API and system.

Since:

JDK1.1

See Also:

`UnicastRemoteObject`, `Activatable`

For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

javax.ejb

Interface EJBObject

All Superinterfaces:

java.rmi.Remote

public interface **EJBObject**
extends java.rmi.Remote

The EJBObject interface is extended by all enterprise Beans' remote interfaces. An enterprise Bean's remote interface provides the remote client view of an EJB object. An enterprise Bean's remote interface defines the business methods callable by a remote client.

The remote interface must extend the javax.ejb.EJBObject interface, and define the enterprise Bean specific business methods.

The enterprise Bean's remote interface is defined by the enterprise Bean provider and implemented by the enterprise Bean container.

Method Summary

EJBHome	**getEJBHome**() Obtain the enterprise Bean's remote home interface.
Handle	**getHandle**() Obtain a handle for the EJB object.
java.lang.Object	**getPrimaryKey**() Obtain the primary key of the EJB object.
boolean	**isIdentical**(EJBObject obj) Test if a given EJB object is identical to the invoked EJB object.
void	**remove**() Remove the EJB object.

Method Detail

getEJBHome

```
public EJBHome getEJBHome()
                    throws java.rmi.RemoteException
```

Obtain the enterprise Bean's remote home interface. The remote home interface defines the enterprise Bean's create, finder, remove, and home business methods.

Returns:

A reference to the enterprise Bean's home interface.

Throws:

`java.rmi.RemoteException` - Thrown when the method failed due to a system-level failure.

---

getPrimaryKey

```
public java.lang.Object getPrimaryKey()
                                throws java.rmi.RemoteException
```

Obtain the primary key of the EJB object.

This method can be called on an entity bean. An attempt to invoke this method on a session bean will result in RemoteException.

Returns:

The EJB object's primary key.

---

remove

```
public void remove()
            throws java.rmi.RemoteException,
                   RemoveException
```

Remove the EJB object.

Throws:

`java.rmi.RemoteException` - Thrown when the method failed due to a system-level failure.

`RemoveException` - The enterprise Bean or the container does not allow destruction of the object.

---

getHandle

```
public Handle getHandle()
                  throws java.rmi.RemoteException
```

Obtain a handle for the EJB object. The handle can be used at later time to re-obtain a reference to the EJB object, possibly in a different Java Virtual Machine.

Returns:

A handle for the EJB object.

Throws:

`java.rmi.RemoteException` - Thrown when the method failed due to a system-level failure.

---

isIdentical

```
public boolean isIdentical(EJBObject obj)
                    throws java.rmi.RemoteException
```

Test if a given EJB object is identical to the invoked EJB object.

Parameters:

`obj` - An object to test for identity with the invoked object.

Returns:

True if the given EJB object is identical to the invoked object, false otherwise.

Throws:

`java.rmi.RemoteException` - Thrown when the method failed due to a system-level failure.

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;

public class CallHelloWorldBean {
	public static void main(String[] args) {
		try {
			Context initial = new InitialContext();
			Object objRef = initial.lookup("helloBean");

			SayHelloHome home =
				(SayHelloHome) PortableRemoteObject.narrow(
					objRef,
					SayHelloHome.class);
			SayHello sh = home.create();

			System.out.println(sh.sayHello("Mario"));

		} catch (NamingException ne) {
			ne.printStackTrace();
		} catch (RemoteException re) {
			re.printStackTrace();
		} catch (CreateException ce) {
			ce.printStackTrace();
		}
	}
}
```

javax.rmi

Class PortableRemoteObject

java.lang.Object
  └ **javax.rmi.PortableRemoteObject**

public class **PortableRemoteObject**
extends Object

Server implementation objects may either inherit from javax.rmi.PortableRemoteObject or they may implement a remote interface and then use the exportObject method to register themselves as a server object. The toStub method takes a server implementation and returns a stub that can be used to access that server object. The connect method makes a Remote object ready for remote communication. The unexportObject method is used to deregister a server object, allowing it to become available for garbage collection. The narrow method takes an object reference or abstract interface type and attempts to narrow it to conform to the given interface. If the operation is successful the result will be an object of the specified type, otherwise an exception will be thrown.

Constructor Summary

protected	**PortableRemoteObject**() Initializes the object by calling `exportObject(this)`.

Method Summary

static void	**connect**(Remote target, Remote source) Makes a Remote object ready for remote communication.
static void	**exportObject**(Remote obj) Makes a server object ready to receive remote calls.
static Object	**narrow**(Object narrowFrom, Class narrowTo) Checks to ensure that an object of a remote or abstract interface type can be cast to a desired type.

static Remote	**toStub**(Remote obj) Returns a stub for the given server object.
static void	**unexportObject**(Remote obj) Deregisters a server object from the runtime, allowing the object to become available for garbage collection.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PortableRemoteObject

```
protected PortableRemoteObject()
                          throws RemoteException
```

Initializes the object by calling `exportObject(this)`.

Throws:

RemoteException - if export fails.

Method Detail

exportObject

```
public static void exportObject(Remote obj)
                           throws RemoteException
```

Makes a server object ready to receive remote calls. Note that subclasses of PortableRemoteObject do not need to call this method, as it is called by the constructor.

Parameters:

`obj` - the server object to export.

Throws:

RemoteException - if export fails.

---

toStub

```
public static Remote toStub(Remote obj)
                     throws NoSuchObjectException
```

Returns a stub for the given server object.

Parameters:

`obj` - the server object for which a stub is required. Must either be a subclass of PortableRemoteObject or have been previously the target of a call to `exportObject(java.rmi.Remote)`.

Returns:

the most derived stub for the object.

Throws:

`NoSuchObjectException` - if a stub cannot be located for the given server object.

---

unexportObject

```
public static void unexportObject(Remote obj)
                           throws NoSuchObjectException
```

Deregisters a server object from the runtime, allowing the object to become available for garbage collection.

Parameters:

`obj` - the object to unexport.

Throws:

`NoSuchObjectException` - if the remote object is not currently exported.

---

narrow

```
public static Object narrow(Object narrowFrom,
                            Class narrowTo)
                     throws ClassCastException
```

Checks to ensure that an object of a remote or abstract interface type can be cast to a desired type.

Parameters:

`narrowFrom` - the object to check.
`narrowTo` - the desired type.

Returns:

an object which can be cast to the desired type.

Throws:

`ClassCastException` - if narrowFrom cannot be cast to narrowTo.

---

connect

```
public static void connect(Remote target,
                           Remote source)
                    throws RemoteException
```

Makes a Remote object ready for remote communication. This normally happens implicitly when the object is sent or received as an argument on a remote method call, but in some circumstances it is useful to perform this action by making an explicit call. See the `Stub#connect` method for more information.

Parameters:

`target` - the object to connect.
`source` - a previously connected object.

Throws:

`RemoteException` - if `source` is not connected or if `target` is already connected to a different ORB than `source`.

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
import javax.ejb.FinderException;

public interface PersonHome extends EJBHome {
	public Person create(int year, int month, int day, String name, String street) throws
CreateException, RemoteException;
	public Person findByPrimaryKey(String name) throws FinderException, RemoteException;
}
```

```
import java.rmi.RemoteException;
import javax.ejb.EJBObject;

public interface Person extends EJBObject {
	public int getAge() throws RemoteException;
	public String getStreet() throws RemoteException;
	public void setStreet(String street) throws RemoteException;
}
```

javax.ejb

Interface EntityBean

All Superinterfaces:

EnterpriseBean, java.io.Serializable

public interface **EntityBean**
extends EnterpriseBean

The EntityBean interface is implemented by every entity enterprise Bean class. The container uses the EntityBean methods to notify the enterprise Bean instances of the instance's life cycle events.

Method Summary

void	**ejbActivate**() A container invokes this method when the instance is taken out of the pool of available instances to become associated with a specific EJB object.
void	**ejbLoad**() A container invokes this method to instruct the instance to synchronize its state by loading it state from the underlying database.
void	**ejbPassivate**() A container invokes this method on an instance before the instance becomes disassociated with a specific EJB object.
void	**ejbRemove**() A container invokes this method before it removes the EJB object that is currently associated with the instance.
void	**ejbStore**() A container invokes this method to instruct the instance to synchronize its state by storing it to the underlying database.
void	**setEntityContext**(EntityContext ctx) Set the associated entity context.

void	**unsetEntityContext**() Unset the associated entity context.

Method Detail

setEntityContext

```
public void setEntityContext(EntityContext ctx)
                      throws EJBException,
                             java.rmi.RemoteException
```

Set the associated entity context. The container invokes this method on an instance after the instance has been created.

This method is called in an unspecified transaction context.

Parameters:

`ctx` - An EntityContext interface for the instance. The instance should store the reference to the context in an instance variable.

Throws:

`EJBException` - Thrown by the method to indicate a failure caused by a system-level error.

`java.rmi.RemoteException` - This exception is defined in the method signature to provide backward compatibility for enterprise beans written for the EJB 1.0 specification. Enterprise beans written for the EJB 1.1 specification should throw the javax.ejb.EJBException instead of this exception. Enterprise beans written for the EJB2.0 and higher specifications must throw the javax.ejb.EJBException instead of this exception.

---

unsetEntityContext

```
public void unsetEntityContext()
                        throws EJBException,
                               java.rmi.RemoteException
```

Unset the associated entity context. The container calls this method before removing the instance.

This is the last method that the container invokes on the instance. The Java garbage collector will eventually invoke the finalize() method on the instance.

This method is called in an unspecified transaction context.

Throws:

EJBException - Thrown by the method to indicate a failure caused by a system-level error.

`java.rmi.RemoteException` - This exception is defined in the method signature to provide backward compatibility for enterprise beans written for the EJB 1.0 specification. Enterprise beans written for the EJB 1.1 specification should throw the javax.ejb.EJBException instead of this exception. Enterprise beans written for the EJB2.0 and higher specifications must throw the javax.ejb.EJBException instead of this exception.

---

ejbRemove

```
public void ejbRemove()
            throws RemoveException,
                   EJBException,
                   java.rmi.RemoteException
```

A container invokes this method before it removes the EJB object that is currently associated with the instance. This method is invoked when a client invokes a remove operation on the enterprise Bean's home interface or the EJB object's remote interface. This method transitions the instance from the ready state to the pool of available instances.

This method is called in the transaction context of the remove operation.

Throws:

RemoveException - The enterprise Bean does not allow destruction of the object.

EJBException - Thrown by the method to indicate a failure caused by a system-level error.

`java.rmi.RemoteException` - This exception is defined in the method signature to provide backward compatibility for enterprise beans written for the EJB 1.0 specification. Enterprise beans written for the EJB 1.1 specification should throw the javax.ejb.EJBException instead of this exception. Enterprise beans written for the EJB2.0 and higher specifications must throw the javax.ejb.EJBException instead of this exception.

---

ejbActivate

```
public void ejbActivate()
              throws EJBException,
                     java.rmi.RemoteException
```

A container invokes this method when the instance is taken out of the pool of available instances to become associated with a specific EJB object. This method transitions the instance to the ready state.

This method executes in an unspecified transaction context.

Throws:

`EJBException` - Thrown by the method to indicate a failure caused by a system-level error.

`java.rmi.RemoteException` - This exception is defined in the method signature to provide backward compatibility for enterprise beans written for the EJB 1.0 specification. Enterprise beans written for the EJB 1.1 specification should throw the javax.ejb.EJBException instead of this exception. Enterprise beans written for the EJB2.0 and higher specifications must throw the javax.ejb.EJBException instead of this exception.

---

ejbPassivate

```
public void ejbPassivate()
                  throws EJBException,
                         java.rmi.RemoteException
```

A container invokes this method on an instance before the instance becomes disassociated with a specific EJB object. After this method completes, the container will place the instance into the pool of available instances.

This method executes in an unspecified transaction context.

Throws:

`EJBException` - Thrown by the method to indicate a failure caused by a system-level error.

`java.rmi.RemoteException` - This exception is defined in the method signature to provide backward compatibility for enterprise beans written for the EJB 1.0 specification. Enterprise beans written for the EJB 1.1 specification should throw the javax.ejb.EJBException instead of this exception. Enterprise beans written for the EJB2.0 and higher specifications must throw the javax.ejb.EJBException instead of this exception.

---

ejbLoad

```
public void ejbLoad()
```

```
        throws EJBException,
               java.rmi.RemoteException
```

A container invokes this method to instruct the instance to synchronize its state by loading it state from the underlying database.

This method always executes in the transaction context determined by the value of the transaction attribute in the deployment descriptor.

Throws:

EJBException - Thrown by the method to indicate a failure caused by a system-level error.

`java.rmi.RemoteException` - This exception is defined in the method signature to provide backward compatibility for enterprise beans written for the EJB 1.0 specification. Enterprise beans written for the EJB 1.1 specification should throw the javax.ejb.EJBException instead of this exception. Enterprise beans written for the EJB2.0 and higher specifications must throw the javax.ejb.EJBException instead of this exception.

---

ejbStore

```
public void ejbStore()
            throws EJBException,
                   java.rmi.RemoteException
```

A container invokes this method to instruct the instance to synchronize its state by storing it to the underlying database.

This method always executes in the transaction context determined by the value of the transaction attribute in the deployment descriptor.

Throws:

EJBException - Thrown by the method to indicate a failure caused by a system-level error.

`java.rmi.RemoteException` - This exception is defined in the method signature to provide backward compatibility for enterprise beans written for the EJB 1.0 specification. Enterprise beans written for the EJB 1.1 specification should throw the javax.ejb.EJBException instead of this exception. Enterprise beans written for the EJB2.0 and higher specifications must throw the javax.ejb.EJBException instead of this exception.

---

javax.ejb

Interface EnterpriseBean

All Superinterfaces:

java.io.Serializable

All Known Subinterfaces:

EntityBean, MessageDrivenBean, SessionBean

---

public interface **EnterpriseBean**
extends java.io.Serializable

The EnterpriseBean interface must be implemented by every enterprise Bean class. It is a common superinterface for the SessionBean, EntityBean and MessageDrivenBean interfaces.

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

```
import java.rmi.RemoteException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Calendar;
import java.util.GregorianCalendar;

import javax.ejb.CreateException;
import javax.ejb.EJBException;
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;
import javax.ejb.FinderException;
import javax.ejb.RemoveException;

public class PersonBean implements EntityBean {
	//persistent fields
	private String name;
	private GregorianCalendar birthdate;
	private String street;

	public PersonBean() {
	}

	//methods which are part of the remote interface
	public int getAge() {
		return (
			new GregorianCalendar().get(Calendar.YEAR)
				- birthdate.get(Calendar.YEAR));
	}
	public String getStreet() {
		return street;
	}
	public void setStreet(String street) throws RemoteException {
		if (street.length() <= 30) {
			this.street = street;
		} else {
			throw new RemoteException("cannot update street");
		}
	}
	//	---------------------------------------------------------------
	public String ejbCreate(
		int year,
		int month,
```

```
        int day,
        String name,
        String street)
        throws CreateException {
        if (year > 1900
                && month >= 1
                && month <= 12
                && day >= 1
                && day <= 31
                && name.length() <= 20
                && street.length() <= 30) {

                this.name = name;
                this.birthdate = new GregorianCalendar(year, month, day);
                this.street = street;
                Statement stmt = getStatement();
                String s =
                        new String(
                                "INSERT INTO PERSON VALUES ('"
                                        + name
                                        + "','"
                                        + birthdate.get(Calendar.YEAR)
                                        + "-"
                                        + birthdate.get(Calendar.MONTH)
                                        + "-"
                                        + birthdate.get(Calendar.DATE)
                                        + "','"
                                        + "'"
                                        + street
                                        + "'"
                                        + ");");
                try {
                        stmt.executeUpdate(s);
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        } else {
                throw new CreateException("Invalid values supplied");
        }
        return name;
    }
    public void ejbPostCreate(
        int year,
        int month,
        int day,
        String name,
```

```
        String street) {
    }
    //      ---------------------------------------------------------------
    public int ejbHomeGetAge() {
        return 0;
    }

    public String ejbHomeGetStreet() {
        return new String();
    }

    public void ejbHomeSetStreet(String street) {
    }

    public Statement getStatement() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        Connection con = null;
        try {
            con =
                (Connection) DriverManager.getConnection(
                    "jdbc:mysql://10.0.0.1/Address/",
                    "mario",
                    "thePassword");
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
        try {
            return ((Statement) con.createStatement());
        } catch (SQLException e2) {
            e2.printStackTrace();
        }
        return null; //never gets here
    }

    // ---------------------------------------------------------------
    public void setEntityContext(EntityContext ectx)
        throws EJBException, RemoteException {
    }

    public void unsetEntityContext() throws EJBException, RemoteException {
    }
```

```
    public void ejbRemove()
            throws RemoveException, EJBException, RemoteException {
            Statement stmt = getStatement();
            String s = new String("DELETE FROM PERSON WHERE Name='" + name + "';");
            try {
                    stmt.executeUpdate(s);
            } catch (SQLException e) {
                    e.printStackTrace();
            }
    }

    public void ejbActivate() throws EJBException, RemoteException {
    }

    public void ejbPassivate() throws EJBException, RemoteException {
    }

    public void ejbLoad() throws EJBException, RemoteException {
    }

    public void ejbStore() throws EJBException, RemoteException {
            Statement stmt = getStatement();
            String s =
                    new String(
                            "UPDATE PERSON SET Name='"
                                    + name
                                    + "', BDATE='"
                                    + birthdate.get(Calendar.YEAR)
                                    + "-"
                                    + birthdate.get(Calendar.MONTH)
                                    + "-"
                                    + birthdate.get(Calendar.DATE)
                                    + "', Street = '"
                                    + street
                                    + "' WHERE Name = '"
                                    + name
                                    + "';");
            try {
                    stmt.executeUpdate(s);
            } catch (SQLException e) {
                    e.printStackTrace();
            }
    }

    public String ejbFindByPrimaryKey(String name) throws FinderException {
            Statement stmt = getStatement();
```

```
        String s =
            new String("SELECT Name FROM PERSON WHERE Name='" + name + "';");
        try {
            ResultSet rs = stmt.executeQuery(s);
            rs.first();
            return rs.getString("Name");
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null; //never gets here
    }
}
```

java.util

Class GregorianCalendar

java.lang.Object
  └ java.util.Calendar
      └ **java.util.GregorianCalendar**

All Implemented Interfaces:

Cloneable, Serializable

---

public class **GregorianCalendar**
extends Calendar

`GregorianCalendar` is a concrete subclass of `Calendar` and provides the standard calendar used by most of the world.

The standard (Gregorian) calendar has 2 eras, BC and AD.

This implementation handles a single discontinuity, which corresponds by default to the date the Gregorian calendar was instituted (October 15, 1582 in some countries, later in others). The cutover date may be changed by the caller by calling `setGregorianChange()`.

Historically, in those countries which adopted the Gregorian calendar first, October 4, 1582 was thus followed by October 15, 1582. This calendar models this correctly. Before the Gregorian cutover, `GregorianCalendar` implements the Julian calendar. The only difference between the Gregorian and the Julian calendar is the leap year rule. The Julian calendar specifies leap years every four years, whereas the Gregorian calendar omits century years which are not divisible by 400.

`GregorianCalendar` implements *proleptic* Gregorian and Julian calendars. That is, dates are computed by extrapolating the current rules indefinitely far backward and forward in time. As a result, `GregorianCalendar` may be used for all years to generate meaningful and consistent results. However, dates obtained using `GregorianCalendar` are historically accurate only from March 1, 4 AD onward, when modern Julian calendar rules were adopted. Before this date, leap year rules were applied irregularly, and before 45 BC the Julian calendar did not even exist.

Prior to the institution of the Gregorian calendar, New Year's Day was March 25. To avoid confusion,

this calendar always uses January 1. A manual adjustment may be made if desired for dates that are prior to the Gregorian changeover and which fall between January 1 and March 24.

Values calculated for the `WEEK_OF_YEAR` field range from 1 to 53. Week 1 for a year is the earliest seven day period starting on `getFirstDayOfWeek()` that contains at least `getMinimalDaysInFirstWeek()` days from that year. It thus depends on the values of `getMinimalDaysInFirstWeek()`, `getFirstDayOfWeek()`, and the day of the week of January 1. Weeks between week 1 of one year and week 1 of the following year are numbered sequentially from 2 to 52 or 53 (as needed).

For example, January 1, 1998 was a Thursday. If `getFirstDayOfWeek()` is `MONDAY` and `getMinimalDaysInFirstWeek()` is 4 (these are the values reflecting ISO 8601 and many national standards), then week 1 of 1998 starts on December 29, 1997, and ends on January 4, 1998. If, however, `getFirstDayOfWeek()` is `SUNDAY`, then week 1 of 1998 starts on January 4, 1998, and ends on January 10, 1998; the first three days of 1998 then are part of week 53 of 1997.

Values calculated for the `WEEK_OF_MONTH` field range from 0 to 6. Week 1 of a month (the days with `WEEK_OF_MONTH = 1`) is the earliest set of at least `getMinimalDaysInFirstWeek()` contiguous days in that month, ending on the day before `getFirstDayOfWeek()`. Unlike week 1 of a year, week 1 of a month may be shorter than 7 days, need not start on `getFirstDayOfWeek()`, and will not include days of the previous month. Days of a month before week 1 have a `WEEK_OF_MONTH` of 0.

For example, if `getFirstDayOfWeek()` is `SUNDAY` and `getMinimalDaysInFirstWeek()` is 4, then the first week of January 1998 is Sunday, January 4 through Saturday, January 10. These days have a `WEEK_OF_MONTH` of 1. Thursday, January 1 through Saturday, January 3 have a `WEEK_OF_MONTH` of 0. If `getMinimalDaysInFirstWeek()` is changed to 3, then January 1 through January 3 have a `WEEK_OF_MONTH` of 1.

Example:

```
 // get the supported ids for GMT-08:00 (Pacific Standard
Time)
 String[] ids = TimeZone.getAvailableIDs(-8 * 60 * 60 *
1000);
 // if no ids were returned, something is wrong. get out.
 if (ids.length == 0)
     System.exit(0);

  // begin output
 System.out.println("Current Time");

 // create a Pacific Standard Time time zone
 SimpleTimeZone pdt = new SimpleTimeZone(-8 * 60 * 60 *
1000, ids[0]);
```

```
 // set up rules for daylight savings time
 pdt.setStartRule(Calendar.APRIL, 1, Calendar.SUNDAY, 2 *
60 * 60 * 1000);
 pdt.setEndRule(Calendar.OCTOBER, -1, Calendar.SUNDAY, 2
* 60 * 60 * 1000);

 // create a GregorianCalendar with the Pacific Daylight
time zone
 // and the current date and time
 Calendar calendar = new GregorianCalendar(pdt);
 Date trialTime = new Date();
 calendar.setTime(trialTime);

 // print out a bunch of interesting things
 System.out.println("ERA: " + calendar.get(Calendar.ERA));
 System.out.println("YEAR: " + calendar.get(Calendar.
YEAR));
 System.out.println("MONTH: " + calendar.get(Calendar.
MONTH));
 System.out.println("WEEK_OF_YEAR: " + calendar.get
(Calendar.WEEK_OF_YEAR));
 System.out.println("WEEK_OF_MONTH: " + calendar.get
(Calendar.WEEK_OF_MONTH));
 System.out.println("DATE: " + calendar.get(Calendar.
DATE));
 System.out.println("DAY_OF_MONTH: " + calendar.get
(Calendar.DAY_OF_MONTH));
 System.out.println("DAY_OF_YEAR: " + calendar.get
(Calendar.DAY_OF_YEAR));
 System.out.println("DAY_OF_WEEK: " + calendar.get
(Calendar.DAY_OF_WEEK));
 System.out.println("DAY_OF_WEEK_IN_MONTH: "
                    + calendar.get(Calendar.
DAY_OF_WEEK_IN_MONTH));
 System.out.println("AM_PM: " + calendar.get(Calendar.
AM_PM));
 System.out.println("HOUR: " + calendar.get(Calendar.
HOUR));
 System.out.println("HOUR_OF_DAY: " + calendar.get
(Calendar.HOUR_OF_DAY));
 System.out.println("MINUTE: " + calendar.get(Calendar.
MINUTE));
 System.out.println("SECOND: " + calendar.get(Calendar.
SECOND));
 System.out.println("MILLISECOND: " + calendar.get
```

```
(Calendar.MILLISECOND));
 System.out.println("ZONE_OFFSET: "
                    + (calendar.get(Calendar.ZONE_OFFSET)/
(60*60*1000)));
 System.out.println("DST_OFFSET: "
                    + (calendar.get(Calendar.DST_OFFSET)/
(60*60*1000)));

 System.out.println("Current Time, with hour reset to 3");
 calendar.clear(Calendar.HOUR_OF_DAY); // so doesn't
override
 calendar.set(Calendar.HOUR, 3);
 System.out.println("ERA: " + calendar.get(Calendar.ERA));
 System.out.println("YEAR: " + calendar.get(Calendar.
YEAR));
 System.out.println("MONTH: " + calendar.get(Calendar.
MONTH));
 System.out.println("WEEK_OF_YEAR: " + calendar.get
(Calendar.WEEK_OF_YEAR));
 System.out.println("WEEK_OF_MONTH: " + calendar.get
(Calendar.WEEK_OF_MONTH));
 System.out.println("DATE: " + calendar.get(Calendar.
DATE));
 System.out.println("DAY_OF_MONTH: " + calendar.get
(Calendar.DAY_OF_MONTH));
 System.out.println("DAY_OF_YEAR: " + calendar.get
(Calendar.DAY_OF_YEAR));
 System.out.println("DAY_OF_WEEK: " + calendar.get
(Calendar.DAY_OF_WEEK));
 System.out.println("DAY_OF_WEEK_IN_MONTH: "
                    + calendar.get(Calendar.
DAY_OF_WEEK_IN_MONTH));
 System.out.println("AM_PM: " + calendar.get(Calendar.
AM_PM));
 System.out.println("HOUR: " + calendar.get(Calendar.
HOUR));
 System.out.println("HOUR_OF_DAY: " + calendar.get
(Calendar.HOUR_OF_DAY));
 System.out.println("MINUTE: " + calendar.get(Calendar.
MINUTE));
 System.out.println("SECOND: " + calendar.get(Calendar.
SECOND));
 System.out.println("MILLISECOND: " + calendar.get
(Calendar.MILLISECOND));
 System.out.println("ZONE_OFFSET: "
        + (calendar.get(Calendar.ZONE_OFFSET)/
```

```
    (60*60*1000)); // in hours
     System.out.println("DST_OFFSET: "
            + (calendar.get(Calendar.DST_OFFSET)/
    (60*60*1000)); // in hours
```

Since:

JDK1.1

See Also:

`Calendar`, `TimeZone`, Serialized Form

Field Summary

`static int`	**`AD`** Value of the ERA field indicating the common era (Anno Domini), also known as CE.
`static int`	**`BC`** Value of the ERA field indicating the period before the common era (before Christ), also known as BCE.

Fields inherited from class java.util.Calendar

`AM`, `AM_PM`, `APRIL`, `areFieldsSet`, `AUGUST`, `DATE`, `DAY_OF_MONTH`, `DAY_OF_WEEK`, `DAY_OF_WEEK_IN_MONTH`, `DAY_OF_YEAR`, `DECEMBER`, `DST_OFFSET`, `ERA`, `FEBRUARY`, `FIELD_COUNT`, `fields`, `FRIDAY`, `HOUR`, `HOUR_OF_DAY`, `isSet`, `isTimeSet`, `JANUARY`, `JULY`, `JUNE`, `MARCH`, `MAY`, `MILLISECOND`, `MINUTE`, `MONDAY`, `MONTH`, `NOVEMBER`, `OCTOBER`, `PM`, `SATURDAY`, `SECOND`, `SEPTEMBER`, `SUNDAY`, `THURSDAY`, `time`, `TUESDAY`, `UNDECIMBER`, `WEDNESDAY`, `WEEK_OF_MONTH`, `WEEK_OF_YEAR`, `YEAR`, `ZONE_OFFSET`

Constructor Summary

`GregorianCalendar``()` Constructs a default GregorianCalendar using the current time in the default time zone with the default locale.
`GregorianCalendar``(int year, int month, int date)` Constructs a GregorianCalendar with the given date set in the default time zone with the default locale.

GregorianCalendar(int year, int month, int date, int hour, int minute) Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale.
GregorianCalendar(int year, int month, int date, int hour, int minute, int second) Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale.
GregorianCalendar(Locale aLocale) Constructs a GregorianCalendar based on the current time in the default time zone with the given locale.
GregorianCalendar(TimeZone zone) Constructs a GregorianCalendar based on the current time in the given time zone with the default locale.
GregorianCalendar(TimeZone zone, Locale aLocale) Constructs a GregorianCalendar based on the current time in the given time zone with the given locale.

Method Summary

void	**add**(int field, int amount) Adds the specified (signed) amount of time to the given time field, based on the calendar's rules.
protected void	**computeFields**() Converts UTC as milliseconds to time field values.
protected void	**computeTime**() Overrides Calendar Converts time field values to UTC as milliseconds.
boolean	**equals**(Object obj) Compares this GregorianCalendar to an object reference.
int	**getActualMaximum**(int field) Return the maximum value that this field could have, given the current date.
int	**getActualMinimum**(int field) Return the minimum value that this field could have, given the current date.
int	**getGreatestMinimum**(int field) Returns highest minimum value for the given field if varies.
Date	**getGregorianChange**() Gets the Gregorian Calendar change date.

int	**getLeastMaximum**(int field) Returns lowest maximum value for the given field if varies.
int	**getMaximum**(int field) Returns maximum value for the given field.
int	**getMinimum**(int field) Returns minimum value for the given field.
int	**hashCode**() Override hashCode.
boolean	**isLeapYear**(int year) Determines if the given year is a leap year.
void	**roll**(int field, boolean up) Adds or subtracts (up/down) a single unit of time on the given time field without changing larger fields.
void	**roll**(int field, int amount) Add to field a signed amount without changing larger fields.
void	**setGregorianChange**(Date date) Sets the GregorianCalendar change date.

Methods inherited from class java.util.Calendar

after, before, clear, clear, clone, complete, get, getAvailableLocales, getFirstDayOfWeek, getInstance, getInstance, getInstance, getInstance, getMinimalDaysInFirstWeek, getTime, getTimeInMillis, getTimeZone, internalGet, isLenient, isSet, set, set, set, set, setFirstDayOfWeek, setLenient, setMinimalDaysInFirstWeek, setTime, setTimeInMillis, setTimeZone, toString

Methods inherited from class java.lang.Object

finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

BC

```
public static final int BC
```

Value of the ERA field indicating the period before the common era (before Christ), also known as BCE. The sequence of years at the transition from BC to AD is ..., 2 BC, 1 BC, 1 AD, 2 AD,...

See Also:
Calendar.ERA, Constant Field Values

AD

```
public static final int AD
```

Value of the ERA field indicating the common era (Anno Domini), also known as CE. The sequence of years at the transition from BC to AD is ..., 2 BC, 1 BC, 1 AD, 2 AD,...

See Also:
Calendar.ERA, Constant Field Values

Constructor Detail

GregorianCalendar

```
public GregorianCalendar()
```

Constructs a default GregorianCalendar using the current time in the default time zone with the default locale.

GregorianCalendar

```
public GregorianCalendar(TimeZone zone)
```

Constructs a GregorianCalendar based on the current time in the given time zone with the default locale.

Parameters:
zone - the given time zone.

GregorianCalendar

```
public GregorianCalendar(Locale aLocale)
```

Constructs a GregorianCalendar based on the current time in the default time zone with the given locale.

Parameters:

`aLocale` - the given locale.

GregorianCalendar

```
public GregorianCalendar(TimeZone zone,
                         Locale aLocale)
```

Constructs a GregorianCalendar based on the current time in the given time zone with the given locale.

Parameters:

`zone` - the given time zone.
`aLocale` - the given locale.

GregorianCalendar

```
public GregorianCalendar(int year,
                         int month,
                         int date)
```

Constructs a GregorianCalendar with the given date set in the default time zone with the default locale.

Parameters:

`year` - the value used to set the YEAR time field in the calendar.
`month` - the value used to set the MONTH time field in the calendar. Month value is 0-based. e.g., 0 for January.
`date` - the value used to set the DATE time field in the calendar.

GregorianCalendar

```
public GregorianCalendar(int year,
                         int month,
                         int date,
                         int hour,
                         int minute)
```

Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale.

Parameters:

`year` - the value used to set the YEAR time field in the calendar.
`month` - the value used to set the MONTH time field in the calendar. Month value is 0-based. e.g., 0 for January.
`date` - the value used to set the DATE time field in the calendar.
`hour` - the value used to set the HOUR_OF_DAY time field in the calendar.
`minute` - the value used to set the MINUTE time field in the calendar.

GregorianCalendar

```
public GregorianCalendar(int year,
                         int month,
                         int date,
                         int hour,
                         int minute,
                         int second)
```

Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale.

Parameters:

`year` - the value used to set the YEAR time field in the calendar.
`month` - the value used to set the MONTH time field in the calendar. Month value is 0-based. e.g., 0 for January.
`date` - the value used to set the DATE time field in the calendar.
`hour` - the value used to set the HOUR_OF_DAY time field in the calendar.
`minute` - the value used to set the MINUTE time field in the calendar.
`second` - the value used to set the SECOND time field in the calendar.

Method Detail

setGregorianChange

```
public void setGregorianChange(Date date)
```

Sets the GregorianCalendar change date. This is the point when the switch from Julian dates to Gregorian dates occurred. Default is October 15, 1582. Previous to this, dates will be in the Julian calendar.

To obtain a pure Julian calendar, set the change date to `Date(Long.MAX_VALUE)`. To obtain a pure Gregorian calendar, set the change date to `Date(Long.MIN_VALUE)`.

Parameters:

`date` - the given Gregorian cutover date.

---

getGregorianChange

```
public final Date getGregorianChange()
```

Gets the Gregorian Calendar change date. This is the point when the switch from Julian dates to Gregorian dates occurred. Default is October 15, 1582. Previous to this, dates will be in the Julian calendar.

Returns:

the Gregorian cutover date for this calendar.

---

isLeapYear

```
public boolean isLeapYear(int year)
```

Determines if the given year is a leap year. Returns true if the given year is a leap year.

Parameters:

`year` - the given year.

Returns:

true if the given year is a leap year; false otherwise.

---

equals

```
public boolean equals(Object obj)
```

Compares this GregorianCalendar to an object reference.

Overrides:
`equals` in class `Calendar`
Parameters:
`obj` - the object reference with which to compare
Returns:
true if this object is equal to `obj`; false otherwise

---

hashCode

```
public int hashCode()
```

Override hashCode. Generates the hash code for the GregorianCalendar object

Overrides:
`hashCode` in class `Calendar`
Returns:
a hash code value for this object.

---

add

```
public void add(int field,
                int amount)
```

Adds the specified (signed) amount of time to the given time field, based on the calendar's rules.

Add rule 1. The value of `field` after the call minus the value of `field` before the call is `amount`, modulo any overflow that has occurred in `field`. Overflow occurs when a field value exceeds its range and, as a result, the next larger field is incremented or decremented and the field value is adjusted back into its range.

Add rule 2. If a smaller field is expected to be invariant, but it is impossible for it to be equal to its prior value because of changes in its minimum or maximum after `field` is changed, then its value is adjusted to be as close as possible to its expected value. A smaller field represents a smaller unit of time. `HOUR` is a smaller field than `DAY_OF_MONTH`. No

adjustment is made to smaller fields that are not expected to be invariant. The calendar system determines what fields are expected to be invariant.

Specified by:
`add` in class `Calendar`

Parameters:
`field` - the time field.
`amount` - the amount of date or time to be added to the field.

Throws:
`IllegalArgumentException` - if an unknown field is given.

---

roll

```
public void roll(int field,
                 boolean up)
```

Adds or subtracts (up/down) a single unit of time on the given time field without changing larger fields.

Example: Consider a `GregorianCalendar` originally set to December 31, 1999. Calling `roll(Calendar.MONTH, true)` sets the calendar to January 31, 1999. The `Year` field is unchanged because it is a larger field than `MONTH`.

Specified by:
`roll` in class `Calendar`

Parameters:
`up` - indicates if the value of the specified time field is to be rolled up or rolled down. Use true if rolling up, false otherwise.
`field` - the time field.

Throws:
`IllegalArgumentException` - if an unknown field value is given.

See Also:
`add(int, int)`, `Calendar.set(int, int)`

---

roll

```
public void roll(int field,
                 int amount)
```

Add to field a signed amount without changing larger fields. A negative roll amount means to

subtract from field without changing larger fields.

Example: Consider a `GregorianCalendar` originally set to August 31, 1999. Calling `roll(Calendar.MONTH, 8)` sets the calendar to April 30, **1999**. Using a `GregorianCalendar`, the `DAY_OF_MONTH` field cannot be 31 in the month April. `DAY_OF_MONTH` is set to the closest possible value, 30. The `YEAR` field maintains the value of 1999 because it is a larger field than `MONTH`.

Example: Consider a `GregorianCalendar` originally set to Sunday June 6, 1999. Calling `roll(Calendar.WEEK_OF_MONTH, -1)` sets the calendar to Tuesday June 1, 1999, whereas calling `add(Calendar.WEEK_OF_MONTH, -1)` sets the calendar to Sunday May 30, 1999. This is because the roll rule imposes an additional constraint: The `MONTH` must not change when the `WEEK_OF_MONTH` is rolled. Taken together with add rule 1, the resultant date must be between Tuesday June 1 and Saturday June 5. According to add rule 2, the `DAY_OF_WEEK`, an invariant when changing the `WEEK_OF_MONTH`, is set to Tuesday, the closest possible value to Sunday (where Sunday is the first day of the week).

Overrides:

`roll` in class `Calendar`

Parameters:

`field` - the time field.

`amount` - the signed amount to add to `field`.

Since:

1.2

See Also:

`add(int, int)`, `Calendar.set(int, int)`

---

getMinimum

```
public int getMinimum(int field)
```

Returns minimum value for the given field. e.g. for Gregorian DAY_OF_MONTH, 1 Please see Calendar.getMinimum for descriptions on parameters and the return value.

Specified by:

`getMinimum` in class `Calendar`

Parameters:

`field` - the given time field.

Returns:

the minimum value for the given time field.

---

getMaximum

```
public int getMaximum(int field)
```

Returns maximum value for the given field. e.g. for Gregorian DAY_OF_MONTH, 31 Please see Calendar.getMaximum for descriptions on parameters and the return value.

Specified by:
`getMaximum` in class `Calendar`
Parameters:
`field` - the given time field.
Returns:
the maximum value for the given time field.

---

getGreatestMinimum

```
public int getGreatestMinimum(int field)
```

Returns highest minimum value for the given field if varies. Otherwise same as getMinimum (). For Gregorian, no difference. Please see Calendar.getGreatestMinimum for descriptions on parameters and the return value.

Specified by:
`getGreatestMinimum` in class `Calendar`
Parameters:
`field` - the given time field.
Returns:
the highest minimum value for the given time field.

---

getLeastMaximum

```
public int getLeastMaximum(int field)
```

Returns lowest maximum value for the given field if varies. Otherwise same as getMaximum (). For Gregorian DAY_OF_MONTH, 28 Please see Calendar.getLeastMaximum for descriptions on parameters and the return value.

Specified by:
`getLeastMaximum` in class `Calendar`

Parameters:
`field` - the given time field.
Returns:
the lowest maximum value for the given time field.

---

getActualMinimum

```
public int getActualMinimum(int field)
```

Return the minimum value that this field could have, given the current date. For the Gregorian calendar, this is the same as getMinimum() and getGreatestMinimum().

Overrides:
`getActualMinimum` in class `Calendar`
Parameters:
`field` - the field to determine the minimum of
Returns:
the minimum of the given field for the current date of this Calendar
Since:
1.2

---

getActualMaximum

```
public int getActualMaximum(int field)
```

Return the maximum value that this field could have, given the current date. For example, with the date "Feb 3, 1997" and the DAY_OF_MONTH field, the actual maximum would be 28; for "Feb 3, 1996" it s 29. Similarly for a Hebrew calendar, for some years the actual maximum for MONTH is 12, and for others 13.

Overrides:
`getActualMaximum` in class `Calendar`
Parameters:
`field` - the field to determine the maximum of
Returns:
the maximum of the given field for the current date of this Calendar
Since:
1.2

---

computeFields

```
protected void computeFields()
```

Converts UTC as milliseconds to time field values. The time is *not* recomputed first; to recompute the time, then the fields, call the `complete` method.

Specified by:
`computeFields` in class `Calendar`

See Also:
`Calendar.complete()`

---

computeTime

```
protected void computeTime()
```

Overrides Calendar Converts time field values to UTC as milliseconds.

Specified by:
`computeTime` in class `Calendar`

Throws:
`IllegalArgumentException` - if any fields are invalid.

A Summary of the International Standard Date and Time Notation

by Markus Kuhn

International Standard ISO 8601 specifies numeric representations of date and time. This standard notation helps to avoid confusion in international communication caused by the many different national notations and increases the portability of computer user interfaces. In addition, these formats have several important advantages for computer usage compared to other traditional date and time notations. The time notation described here is already the de-facto standard in almost all countries and the date notation is becoming increasingly popular.

Especially authors of Web pages and software engineers who design user interfaces, file formats, and communication protocols should be familiar with ISO 8601.

Contents: Date, Time of Day, Time Zone.

Date

The international standard date notation is

YYYY-MM-DD

where YYYY is the year in the usual Gregorian calendar, MM is the month of the year between 01 (January) and 12 (December), and DD is the day of the month between 01 and 31.

For example, the fourth day of February in the year 1995 is written in the standard notation as

1995-02-04

Other commonly used notations are e.g. 2/4/95, 4/2/95, 95/2/4, 4.2.1995, 04-FEB-1995, 4-February-1995, and many more. Especially the first two examples are dangerous, because as both are used quite often in the U.S. and in Great Britain and both can not be distinguished, it is unclear whether 2/4/95 means 1995-04-02 or 1995-02-04. The date notation 2/4/5 has at least six reasonable interpretations (assuming that only the twentieth and twenty-first century are reasonable candidates in our life time).

Advantages of the ISO 8601 standard date notation compared to other commonly used variants:

- easily readable and writeable by software (no 'JAN', 'FEB', ... table necessary)
- easily comparable and sortable with a trivial string comparison

- language independent
- can not be confused with other popular date notations
- consistency with the common 24h time notation system, where the larger units (hours) are also written in front of the smaller ones (minutes and seconds)
- strings containing a date followed by a time are also easily comparable and sortable (e.g. write "1995-02-04 22:45:00")
- the notation is short and has constant length, which makes both keyboard data entry and table layout easier
- identical to the Chinese date notation, so the largest cultural group (>25%) on this planet is already familiar with it :-)
- date notations with the order "year, month, day" are in addition already widely used e.g. in Japan, Korea, Hungary, Sweden, Finland, Denmark, and a few other countries and people in the U.S. are already used to at least the "month, day" order
- a 4-digit year representation avoids overflow problems after 2099-12-31

As dates will look a little bit strange anyway starting with 2000-01-01 (e.g. like 1/1/0), it has been suggested that the year 2000 is an excellent opportunity to change to the standard date notation.

ISO 8601 is only specifying numeric notations and does not cover dates and times where words are used in the representation. It is not intended as a replacement for language-dependent worded date notations such as "24. Dezember 2001" (German) or "February 4, 1995" (US English). ISO 8601 should however be used to replace notations such as "2/4/95" and "9.30 p.m.".

Apart from the recommended primary standard notation **YYYY-MM-DD**, ISO 8601 also specifies a number of alternative formats for use in applications with special requirements. All of these alternatives can easily and automatically be distinguished from each other:

The hyphens can be omitted if compactness of the representation is more important than human readability, for example as in

19950204

For situations where information about the century is really not required, a 2-digit year representation is available:

95-02-04 or **950204**

If only the month or even only the year is of interest:

1995-02 or **1995**

In commercial and industrial applications (delivery times, production plans, etc.), especially in Europe, it is often required to refer to a week of a year. Week 01 of a year is per definition the first week that has the Thursday in this year, which is equivalent to the week that contains the fourth day of January. In other words, the first week of a new year is the week that has the majority of its days in

the new year. Week 01 might also contain days from the previous year and the week before week 01 of a year is the last week (52 or 53) of the previous year even if it contains days from the new year. A week starts with Monday (day 1) and ends with Sunday (day 7). For example, the first week of the year 1997 lasts from 1996-12-30 to 1997-01-05 and can be written in standard notation as

1997-W01 or **1997W01**

The week notation can also be extended by a number indicating the day of the week. For example, the day 1996-12-31, which is the Tuesday (day 2) of the first week of 1997, can also be written as

1997-W01-2 or **1997W012**

for applications like industrial planning where many things like shift rotations are organized per week and knowing the week number and the day of the week is more handy than knowing the day of the month.

An abbreviated version of the year and week number like

95W05

is sometimes useful as a compact code printed on a product that indicates when it has been manufactured.

The ISO standard avoids explicitly stating the possible range of week numbers, but this can easily be deduced from the definition:

Theorem: Possible ISO week numbers are in the range 01 to 53. A year always has a week 52. (There is one historic exception: the year in which the Gregorian calendar was introduced had less than 365 days and less than 52 weeks.)

Proof: Per definition, the first week of a year is W01 and consequently days before week W01 belong to the previous year and so there is no week with lower numbers. Considering the highest possible week number, the worst case is a leap year like 1976 that starts with a Thursday, because this keeps the highest possible number of days of W01 in the previous year, i.e. 3 days. In this case, the Sunday of W52 of the worst case year is day number 4+51*7=361 and 361-366=5 days of W53 belong still to this year, which guarantees that in the worst case year day 4 (Thursday) of W53 is not yet in the next year, so a week number 53 is possible. For example, the 53 weeks of the worst case year 1976 started with 1975-12-29 = 1976-W01-1 and ended with 1977-01-02 = 1976-W53-7. On the other hand, considering the lowest number of the last week of a year, the worst case is a non-leap year like 1999 that starts with a Friday, which ensures that the first three days of the year belong to the last week of the previous year. In this case, the Sunday of week 52 would be day number 3+52*7=367, i.e. only the last 367-365=2 days of the W52 reach into the next year and consequently, even a worst case year like 1999 has a week W52 including the days 1999-12-27 to 2000-01-02. q.e.d.

[The new 1999 version of the C programming language standard (ISO 9899) added in the `strftime()` function means to generate the ISO 8601 week notation. The author of this text developed a further proposal for a modernised clock and calendar API for C, which provides full proper treatment of leap seconds and timezones and fixes numerous other problems in the current C timing library functions. It also serves as a model for those who want to design clock library functions for other programming languages.]

Both day and year are useful units of structuring time, because the position of the sun on the sky, which influences our lives, is described by them. However the 12 months of a year are of some obscure mystic origin and have no real purpose today except that people are used to having them (they do not even describe the current position of the moon). In some applications, a date notation is preferred that uses only the year and the day of the year between 001 and 365 (366 in leap years). The standard notation for this variant representing the day 1995-02-04 (that is day 035 of the year 1995) is

1995-035 or **1995035**

Leap years are years with an additional day YYYY-02-29, where the year number is a multiple of four with the following exception: If a year is a multiple of 100, then it is only a leap year if it is also a multiple of 400. For example, 1900 was not a leap year, but 2000 is one.

Time of Day

The international standard notation for the time of day is

hh:mm:ss

where hh is the number of complete hours that have passed since midnight (00-24), mm is the number of complete minutes that have passed since the start of the hour (00-59), and ss is the number of complete seconds since the start of the minute (00-60). If the hour value is 24, then the minute and second values must be zero. [The value 60 for ss might sometimes be needed during an inserted leap second in an atomic time scale like Coordinated Universal Time (UTC). A single leap second 23:59:60 is inserted into the UTC time scale every few years as announced by the International Earth Rotation Service in Paris to keep UTC from wandering away more than 0.9 s from the less constant astronomical time scale UT1 that is defined by the actual rotation of the earth.]

An example time is

23:59:59

which represents the time one second before midnight.

As with the date notation, the separating colons can also be omitted as in

235959

and the precision can be reduced by omitting the seconds or both the seconds and minutes as in

23:59, **2359**, or **23**

It is also possible to add fractions of a second after a decimal dot or comma, for instance the time 5.8 ms before midnight can be written as

23:59:59.9942 or **235959.9942**

As every day both starts and ends with midnight, the two notations **00:00** and **24:00** are available to distinguish the two midnights that can be associated with one date. This means that the following two notations refer to exactly the same point in time:

1995-02-04 24:00 = **1995-02-05 00:00**

In case an unambiguous representation of time is required, 00:00 is usually the preferred notation for midnight and not 24:00. Digital clocks display 00:00 and not 24:00.

ISO 8601 does not specify, whether its notations specify a point in time or a time period. This means for example that ISO 8601 does not define whether 09:00 refers to the exact end of the ninth hour of the day or the period from 09:00 to 09:01 or anything else. The users of the standard must somehow agree on the exact interpretation of the time notation if this should be of any concern.

If a date and a time are displayed on the same line, then always write the date in front of the time. If a date and a time value are stored together in a single data field, then ISO 8601 suggests that they should be separated by a latin capital letter T, as in **19951231T235959**.

A remark for readers from the U.S.:

> The 24h time notation specified here has already been the de-facto standard all over the world in written language for decades. The only exception are a few English speaking countries, where still notations with hours between 1 and 12 and additions like "a.m." and "p.m." are in wide use. The common 24h international standard notation is widely used now even in England (e.g. at airports, cinemas, bus/train timetables, etc.). Most other languages don't even have abbreviations like "a.m." and "p.m." and the 12h notation is certainly hardly ever used on Continental Europe to write or display a time. Even in the U.S., the military and computer programmers have been using the 24h notation for a long time.
>
> The old English 12h notation has many disadvantages like:
>
> - It is longer than the normal 24h notation.

- It takes somewhat more time for humans to compare two times in 12h notation.
- It is not clear, how 00:00, 12:00 and 24:00 are represented. Even encyclopedias and style manuals contain contradicting descriptions and a common quick fix seems to be to avoid "12:00 a.m./p.m." altogether and write "noon", "midnight", or "12:01 a.m./p.m." instead, although the word "midnight" still does not distinguish between 00:00 and 24:00 (midnight at the start or end of a given day).
- It makes people often believe that the next day starts at the overflow from "12:59 a.m." to "1:00 a.m.", which is a common problem not only when people try to program the timer of VCRs shortly after midnight.
- It is not easily comparable with a string compare operation.
- It is not immediately clear for the unaware, whether the time between "12:00 a.m./p.m." and "1:00 a.m./p.m." starts at 00:00 or at 12:00, i.e. the English 12h notation is more difficult to understand.

Please consider the 12h time to be a relic from the dark ages when Roman numerals were used, the number zero had not yet been invented and analog clocks were the only known form of displaying a time. Please avoid using it today, especially in technical applications! Even in the U.S., the widely respected *Chicago Manual of Style* now recommends using the international standard time notation in publications.

A remark for readers from German speaking countries:

The German standard DIN 5008, which specifies typographical rules for German texts written on typewriters, was updated in 1996-05. The old German numeric date notations DD.MM.YYYY and DD.MM.YY have been replaced by the ISO date notations YYYY-MM-DD and YY-MM-DD. Similarly, the old German time notations hh.mm and hh.mm.ss have been replaced by the ISO notations hh:mm and hh:mm:ss. Those new notations are now also mentioned in the latest edition of the *Duden*. The German alphanumeric date notation continues to be for example "3. August 1994" or "3. Aug. 1994". The corresponding Austrian standard has already used the ISO 8601 date and time notations before.

ISO 8601 has been adopted as European Standard EN 28601 and is therefore now a valid standard in all EU countries and all conflicting national standards have been changed accordingly.

Time Zone

Without any further additions, a date and time as written above is assumed to be in some local time zone. In order to indicate that a time is measured in Universal Time (UTC), you can append a capital letter **Z** to a time as in

23:59:59Z or **2359Z**

[The Z stands for the "zero meridian", which goes through Greenwich in London, and it is also commonly used in radio communication where it is pronounced "Zulu" (the word for Z in the international radio alphabet). Universal Time (sometimes also called "Zulu Time") was called Greenwich Mean Time (GMT) before 1972, however this term should no longer be used. Since the introduction of an international atomic time scale, almost all existing civil time zones are now related to UTC, which is slightly different from the old and now unused GMT.]

The strings

+hh:mm, **+hhmm**, or **+hh**

can be added to the time to indicate that the used local time zone is hh hours and mm minutes ahead of UTC. For time zones west of the zero meridian, which are behind UTC, the notation

-hh:mm, **-hhmm**, or **-hh**

is used instead. For example, Central European Time (CET) is +0100 and U.S./Canadian Eastern Standard Time (EST) is -0500. The following strings all indicate the same point of time:

12:00Z = **13:00+01:00** = **0700-0500**

There exists no international standard that specifies abbreviations for civil time zones like CET, EST, etc. and sometimes the same abbreviation is even used for two very different time zones. In addition, politicians enjoy modifying the rules for civil time zones, especially for daylight saving times, every few years, so the only really reliable way of describing a local time zone is to specify numerically the difference of local time to UTC. Better use directly UTC as your only time zone where this is possible and then you do not have to worry about time zones and daylight saving time changes at all.

More Information about Time Zones

Arthur David Olson and others maintain a database of all current and many historic time zone changes and daylight saving time algorithms. It is available via ftp from elsie.nci.nih.gov in the `tzcode*` and `tzdata*` files. Most Unix time zone handling implementations are based on this package. If you want to join the `tz` mailing list, which is dedicated to discussions about time zones and this software, please send a request for subscription to tz-request at elsie.nci.nih.gov. You can read previous discussion there in the tz archive.

Other Links about Date, Time, and Calendars

Some other interesting sources of information about date and time on the Internet are for example the Glossary of Frequency and Timing Terms and the FAQ provided by NIST, the Yahoo Science: Measurements and Units:Time link collection, the U.S. Naval Observatory Server, the International Earth Rotation Service (IERS) (for time gurus only!), the Network Time Protocol (NTP), the time

and calendar section of the USENET sci.astro FAQ, and the Calendar FAQ.

This was a brief overview of the ISO 8601 standard, which covers only the most useful notations and includes some additional related information. The full standard defines in addition a number of more exotic notations including some for periods of time. The ISO 8601:2000 standard itself can only be ordered on paper or as a PDF file on CD-ROM either via ISO's web site online or from

> International Organization for Standardization
> Case postale 56
> 1, rue de Varembé
> CH-1211 Genève 20
> Switzerland
>
> phone: +41 22 749 01 11
> fax: +41 22 733 34 30
> email: sales at isocs.iso.ch

A more detailed online summary of ISO 8601 than this one is the text *ISO 8601:1988 Date/Time Representations* available from ftp.informatik.uni-erlangen.de/pub/doc/ISO/ISO8601.ps.Z (PostScript, 16 kb, 5 pages) written by Gary Houston, now also available in HTML. Ian Galpin (G1SMD) proposes to use ISO 8601 as a Common Date-Time Standard for Amateur Radio. Steve Adams has written another web page about the ISO date format that is partially based on this text. Another summary of ISO 8601 is Jukka Korpela's page and there are further related pages listed in the Open Directory.

The committee in charge of ISO 8601 is ISO TC 154 and the editor of the second edition ISO 8601:2000 was Louis Visser.

I wish to thank Edward M. Reingold for developing the fine GNU Emacs calendar functions, as well as Rich Wales, Mark Brader, Paul Eggert, and others in the comp.std.internat, comp.protocols.time.ntp, and sci.astro USENET discussion groups for valuable comments about this text. Further comments and hyperlinks to this page are very welcome.

Some journalists recently got interested in the international date and time format and reported about it. Examples include:

- An article by Jon G. Auerbach in the 1999-06-01 issue of the Wall Street Journal, page A1.

If you are a journalist and need information on this or related topics, please feel free to contact me.

You might also be interested in the International Standard Paper Sizes Web page.

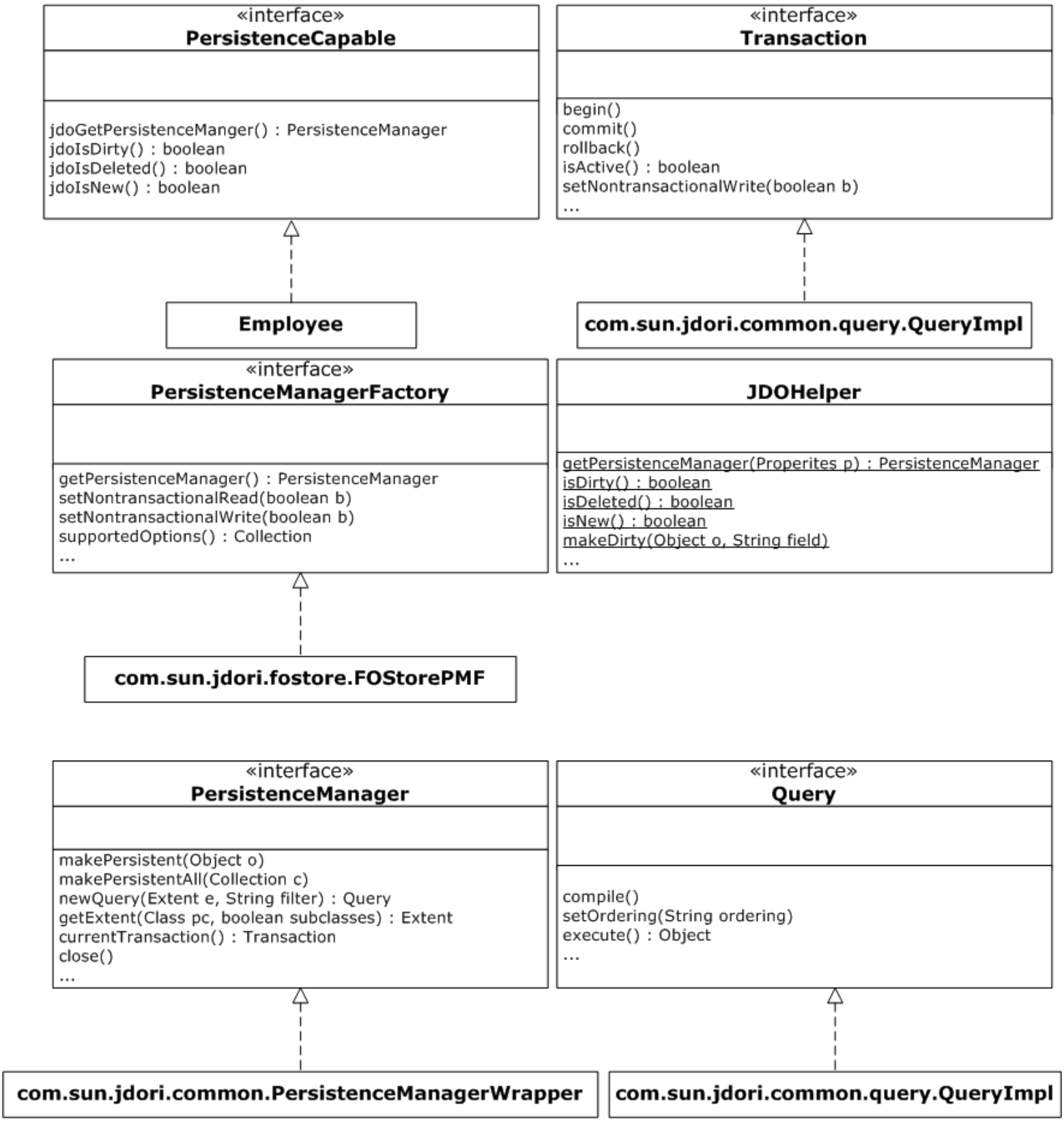
Markus Kuhn

created 1995 -- last modified 2001-11-10 -- http://www.cl.cam.ac.uk/~mgk25/iso-time.html

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import javax.ejb.RemoveException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;

public class CallPersonBean {
	public static void main(String[] args) {
		try {
			Context initial = new InitialContext();
			Object objRef = initial.lookup("personBean");

			PersonHome home =
				(PersonHome) PortableRemoteObject.narrow(
					objRef,
					PersonHome.class);
			Person p1 = home.create(1911, 1, 1, "Alice", "streetA");
			Person p2 = home.create(1922, 2, 2, "Bob", "streetB");
			System.out.println("Alice's Age: " + p1.getAge());

			System.out.println("Alice's primary key: "+p1.getPrimaryKey());
			p1.remove();

			Person p3 = home.findByPrimaryKey("Bob");
			System.out.println(
				"Bob's modified street (before modification): "
					+ p3.getStreet());
			p3.setStreet("streetC");
			System.out.println(
				"Bob's modified street (after modification): "
					+ p3.getStreet());
			System.out.println("also the other reference: " + p2.getStreet());

			System.out.println("Are both references the same Java object: "+ (p2==p3));
			System.out.println("Are both references the same EJBObject (calls isIdentical): "+p2.
isIdentical(p3));

		} catch (NamingException ne) {
			ne.printStackTrace();
		} catch (RemoteException re) {
			re.printStackTrace();
		} catch (CreateException ce) {
```

```
                ce.printStackTrace();
            } catch (RemoveException e) {
                e.printStackTrace();
            } catch (FinderException e) {
                e.printStackTrace();
            }
        }
}
```

```
javax.jdo.PersistenceManagerFactoryClass=com.sun.jdori.fostore.FOStorePMF
javax.jdo.option.ConnectionURL=fostore:jdoriDB
javax.jdo.option.ConnectionUserName=mario
javax.jdo.option.ConnectionPassword=thePassword
```


«interface»
PersistenceCapable
jdoGetPersistenceManger() : PersistenceManager
jdoIsDirty() : boolean
jdoIsDeleted() : boolean
jdoIsNew() : boolean
«interface»
Transaction
begin()
commit()
rollback()
isActive() : boolean
setNontransactionalWrite(boolean b)
...
Employee
com.sun.jdori.common.query.QueryImpl
«interface»
PersistenceManagerFactory
getPersistenceManager() : PersistenceManager
setNontransactionalRead(boolean b)
setNontransactionalWrite(boolean b)
supportedOptions() : Collection
...
JDOHelper
getPersistenceManager(Properites p) : PersistenceManager
isDirty() : boolean
isDeleted() : boolean
isNew() : boolean
makeDirty(Object o, String field)
...
com.sun.jdori.fostore.FOStorePMF
«interface»
PersistenceManager
makePersistent(Object o)
makePersistentAll(Collection c)
newQuery(Extent e, String filter) : Query
getExtent(Class pc, boolean subclasses) : Extent
currentTransaction() : Transaction
close()
...
«interface»
Query
compile()
setOrdering(String ordering)
execute() : Object
...
com.sun.jdori.common.PersistenceManagerWrapper
com.sun.jdori.common.query.QueryImpl

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;
import javax.jdo.Transaction;

public class JDOCreateDB {
	public static void main(String args[]) {
		Properties props = new Properties();
		try {
			InputStream is = ClassLoader.getSystemResourceAsStream("jdo.properties");
			props.load(is);
			props.put("com.sun.jdori.option.ConnectionCreate","true");
		} catch (IOException ioe) {
			System.out.println("Error loading properties");
			System.exit(1);
		}
		PersistenceManagerFactory pmf = JDOHelper.getPersistenceManagerFactory(props);
		PersistenceManager pm = pmf.getPersistenceManager();

		Transaction tx = pm.currentTransaction();
		tx.begin();
		tx.commit();
		pm.close();
	}
}
```

```
package de.jeckle.jdotest;

import java.util.HashSet;
import java.util.Iterator;

public class Employee {
	private String name;
	private String department;
	private HashSet projects = new HashSet();

	public String getName() {
		return name;
	}
	public String getDepartment() {
		return department;
	}
	public void setName(String name) {
		this.name = name;
	}
	public void setDepartment(String department) {
		this.department = department;
	}
	public void addProject(String project) {
		projects.add(project);
	}

	public String toString() {
		String result="Employee named "+name+" works in department "+department;
		result+="\nworks in: ";
		Iterator i = projects.iterator();
		while (i.hasNext()) {
			result+=(String) i.next();
			result+=", ";
		}
		return (result);
	}
}
```

1 Einführung in Java

1.1 Was ist *Java*?

- Laut *Brockhaus*:
  - von staatseigenen indones[ischen] Plantagen auf Java stammender Blatt-Tabak, der je nach Qualität als Deckblatt, Umblatt oder Einlage bei der Zigarrenherstellung verwendet wird.
    Aus: Brockhaus --- Die Enzyklopädie, Bd. 11, 20. Aufl., 1997, p. 149.
  - **Java**, **Jawa** früher **Djawa** [...], kleinste, aber bedeutendste der Großen Sundainseln, Indonesien, 1 060 km lang, 55-195 km breit, 118 000 km^2, (1990) 107,58 Mio. Ew.; verwaltungsmäßig (einschließlich der Insel Madura 132 186 km^2) in drei Prov., die Sonderregion Yogyakarta und den hauptstadtsonderbezirk Jakarta gegliedert. Die Insel ist überwiegend gebirgig. Tätige und erloschene Einzelvulkane und Vulkangruppen bilden die zentrale Längsachse der Insel (mehrere Gipfel über 3 000m ü. M., Semeru 3 676m ü.M.). Beiderseits der Vulkanreihe (17 tätige Vulkane) gliedern sich Berg- und Hügelländer (aus tertiären kalken und mergeln) an. Im N[orden] sind dem Bergland z.T. versumpfte Schwemmlandebenen vorgelagert.
    Aus: Brockhaus --- Die Enzyklopädie, Bd. 11, 20. Aufl., 1997, p. 149.
  - *Informatik*: objektorientierte, in starkem Maße an C^{++} angelehnte betriebssystem- und plattformunabhängige Programmiersprache, die 1995 von der Firma "Sun Microsystems" (USA) entwickelt wurde. J[ava] ist eine Sprache für in Netzen verteilbare Anwendungen, bei denen nicht nur Datensätze, sondern auch Programmteile (so genannte *Applets*;) in den Rechner geladen und ausgeführt werden können. Damit wird die in der Informatik, speziell im ->Internet, äußerst wichtige Forderung nach Systemunabhängigkeit erfüllt. J[ava]-Programme können auf beliebigen Rechnern ausgeführt werden, die über einen javafähigen WWW-Browser (->World Wide Web) verfügen; zusätzlich können wie --- wie herkömml[iche] Programme --- auch ohne Browser ausgeführt werden, sofern das J[ava]-Laufzeitsystem direkt im Betriebsystem implementiert ist.
    Aus: Brockhaus --- Die Enzyklopädie, Bd. 11, 20. Aufl., 1997, p. 149.
- Umgangssprachlicher Ausdruck für starken, heißen Kaffee in den USA
- In einem Satz (nach der Definition von Sun Microsystems): eine einfache und kleine, objektorientierte, dezentrale, interpretierte, stabil laufende, architekturneutrale, portierbare und dynamische Sprache, die Hochgeschwindigkeitsanwendungen und Multithreading unterstützt.
  Im einzelnen bedeutet das:
  - Java ist einfach; die Sprachkonzepte (Syntax) sind überschaubar. Im wesentlichen Stellen sie eine vereinfachte Untermenge der Programmierspache *C++* dar.
    Wesentliche Unterschiede: stärker objektorientiert (d.h. keine *struct*s mehr möglich, jedoch Beibehaltung des Konzepts der primitiven Werttypen), keine expliziten Zeiger (daher auch keine Zeigerarithmetik mehr möglich), keine Operatorüberladung, kein Präprozessor, keine Headerdateien (dafür Paketkonzept), automatische Speicherbereinigung durch Garbgabe Collection.
    Klein zielt sowohl auf den Umfang der entstehenden ausführbaren Dateien, als auch die Größe der Java-Laufzeitumgebung ab. So werden Importdateien in Java nicht statisch inkludiert, sondern nur extern referenziert, wodurch die Größe teilweise signifikant verringert werden kann. Beispielsweise mißt das *HelloWorld*-Beispiel als Java-Bytecode 426 Byte, wohingegen das C-Pendant (HelloWorld.c) (gelinkt als command line executable für die Windows32-Plattform als HelloWorld.exe) über 34.000 Byte, bzw. unter Nutzung der Cygwin-Bibliothek mehr als 18.000 Byte, benötigt.
  - Eine einzige Basisklasse *Object* als Wurel der gesamten Klassenhierarchie. Keine Mehrfachvererbung (aber Nachbildung mit *Interfaces* möglich, wodurch die wesentlichen Nachteile eliminiert werden, jedoch die Vorteile gewahrt beleiben).
    Mit Java-Beans steht ein eigenes Komponentenmodell zur Verfügung.
  - Java erfüllt ein wesentliches Charakteristikum von Client/Server-Anwendungen, die Fähigkeit, Informationen und die Berechnungslast der Daten zu verteilen. Der Begriff *dezentral* beschreibt dabei die Beziehung zwischen Systemobjekten, gleichgültig ob diese Objekte auf lokalen oder entfernten Systemen abgelegt sind.
    Am bekanntesten dürften die Java *Applets* sein, welche die Verteilung ganzer Applikationen über das World-Wide-Web (Abk. WWW).
  - Mit Java erstellte Applikationen sind auf jeder Plattform (Hardware, Betriebsystem) ausführbar, für die eine Java Virtual Machine (Abk. JVM) existiert. Um dies zu erreichen wird durch den Java-Compiler kein plattformspezifischer Objektcode generiert, sondern eine *Zwischenrepräsentation* --- der *Byte-Code*. Dieser wird zur Ausführungzeit (auch: Laufzeit) durch das *Laufzeitsystem* --- die JVM --- *interpretiert* und für den Anwender transparent in maschinenspezifische Instruktionsfolgen umgesetzt. Eine JVM kann auch im *Web Browser* implementiert sein (z.B. Netscape Navigator, Microsoft Internet-Explorer).
  - Java ist *stabil* im Sinne von *zuverlässig*. Dies bedeutet in der Praxis die Verringerung der Absturzwahrscheinlichkeit (gleichbedeutend mit undefiniertem, unvorherseh- und -sagbarem Verhalten) eines mit der Sprache geschriebenen Programms.
    Java ist (wie Pascal oder Oberon) eine streng typisierte Sprache, das bedeutet Typfehler können bereits zur Compilierzeit erkannt werden.
    Laufzeitfehler können in Form von explizten Ausnahmeroutinen behandelt werden (Exception Handling).
    Durch den Verzicht auf explizte Zeiger wurde eine häufige Quelle für Laufzeit- und Speicherfehler eliminiert.
  - Die Neutralität hinsichtlich der zugrundeliegenden Zielarchitektur (Hardware, Betriebssystem) wurde bereits bei der JVM angesprochen. Hier soll nochmals der Aspekt des architekturunabhängigken Bytecodes betont werden.
  - Folgt (eigentlich) direkt aus der Architekturneutralität; die Java-Laufzeitumgebung --- die virtuelle Maschine --- kann für beliebige Zielarchitekturen umgesetzt werden. Sofort danach sind alle Java-Applikationen, ohne Änderung, auf der neuen Plattform ablauffähig.
    Das selbe gilt für mit Java erstelle Oberflächen, die ebenfalls nicht auf eine Plattform (Betriebssystem oder Window Manager) festgelegt sind.
    Der --- bei nativ compilierenden Sprachen wie C/C++ notwendige --- Aufwand eine Applikation an eine neue Zielumgebung anzupassen (d.h. Portierung auf neue Hardware oder Betriebsystem (Zeigerlängen, Datentypgrößen, ...) entfällt durch die Zwischenstufe der JVM, bzw. wird in die Umsetzung dieser auf die neuen Gegebenheiten verlagert.
  - Die Architektur von Java ermöglicht es, die Sprache jederzeit an weiterentwickelnde Umgebungen anzupassen.
  - Es versteht sich von selbst, dass plattformunabhängig ausgelgter Bytecode der zur Laufzeit jedesmal interpretiert und in plattformspezifischen Maschinencode umgesetzt werden muss, Geschwindigkeitsnachteile gegenüber auf genau eine Plattform zugeschnittenen Applikationen aufweißt. Jedoch haben Untersuchungen von Sun gezeigt, dass Java-Bytecode --- der in plattformspezifischen Maschinencode übersetzt wurde --- plattformspezifischem Code anderer Sprachen ebenbürtig ist. (Hierdurch geht jedoch der Aspekt der Plattformunabhängigkeit verloren!).

Java ist jedoch reinen Scriptsprachen (wie Perl oder Python) an Ausführungsgeschwindigkeit signifikant überlegen. Im Allgemeinen ist ein Javaprogramm ca. 15-25 mal langsamer als optimierter C-Code. (Durch *Just in Time Compilation* läßt sich dieses Manko jedoch zumindest abfedern. Mittlerweile existieren auch plattformabhängige Compiler für Java, die nativen Binärcode erzeugen. Dieser erreicht üblicherweise die Geschwindigkeit von C++-Applikationen.)

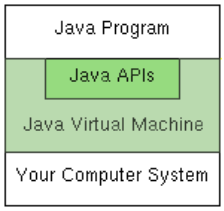
- *Multithreating* soll (zunächst) vereinfacht als die Fähigkeit eines Programms definiert werden mehrere Tätigkeiten zur selben Zeit ausführen zu können.
  Zur Synchronisation wird das Monitorkozept nach Hoare direkt in der API realisiert.

1.2 Entstehungsgeschichte

- 1990: Auf Anraten des Programmierers Patrick Naughton wird bei Sun Microsystems eine Entwicklertruppe unter dem Namen *Green* ins Leben gerufen. Zentrale Mitarbeiter sind, neben Naughton, James Gosling und Mike Sheridan. Ziel des (anfänglich hochgeheimen) Projekts ist die Analyse des EDV-Marktes hinsichtlich verwertbarer Zukunftstrends.
  Die Männer ermitteln als zuküftig erfolgversprechende Einnahmequelle für SUN den Bereich der Consumerelektronik (d.h. Telephone, Videorecorder, Wasch- und Kaffeemaschinen --- kurz allerlei elektronische Alltagsgeräte).
  Ferner definieren sie Anforderungen an ein zukünftiges plattformunabhängiges (d.h. universell einsetzbares) Betriebssystem dieser Maschinen. Vor allem Fehlertoleranz sowie leichte Bedienbarkeit und besondere Stabilität sollten dieses gegenüber existierenden Betriebsystemen auszeichnen. Die Softwaresteuerung sollte in C++ realisiert werden.
- Frühjahr 1991: erster Prototyp eines solchen Systems
- August 1991: Unter Federführung von James Gosling wird nach einer Sprache zur Programmierung des neu zu entwickelnden Betriebsystems gesucht; leider erfolglos.
  Nicht das es im Sommer 1991 an möglichen Programmiersprachenaspiranten mangelte, sondern keiner der Kandidaten deckte das im Jahr zuvor formulierte Anforderungsspektrum zur Zufriedenheit ab.
  Folglich wurde eine neue Sprache --- unter dem Namen *Oak* (engl. Eiche) --- entwickelt.
- 1992: Green-Team präsentiert ersten Prototyp (Trickfigur *Duke*)
  Die unabhängige Firma *FirstPerson* wird gegründet.
- März 1993: Neuorientierung des Projekts und Konzentration der Aktivitäten auf den Bereich interaktiven Fernsehens.
- April 1993: Der von Marc Andressen entwickelte erste graphische WWW-Browser NCSA Mosaic wird vorgestellt.
- Sept. 1993: Arthur van Hoff stößt zur Mannschaft von First Person.
- bis 1994 ist es FirstPerson nicht gelungen Oak zu vermarkten --- die Firma wird eingestellt, und das Produkt verschwindet in der Versenkung. Das Green-Team kehrt zu Sun zurück.
- 1994: Auf Initiative von William (Bill) Joy wird Oak für das Internet modifiziert (der mittlerweile legendäre *Mosaic*-Browser war ein Nebenprodukt der Arbeit des Green-Team).
- 1995: Oak wird im Internet zur Verfügung gestellt; allerdings nun unter dem neuen Namen *Java*.
  Vorstellung der β-Version auf Sun-World-Konferenz.
- 1996: Java Development Kit (Abk. JDK) v1.0
- 1997: JDK v1.1
- 1998: JDK v1.2
- 2000: JDK v1.3
- 2002: JDK: v1.4
- 2003: Erste Betaversion des JDK v1.5 (Codename *Tiger*)


1.3 Die Java-Plattform

Prinzipiell kann, wie erwähnt, eine Java-Applikation auf jeder beliebigen Plattform zur Ausführung gebracht werden --- unabhängig davon welches Gerät die notwendige virtuelle Maschine implementiert.
So existieren Visionen und erste Umsetzungen von JVMs für Elektrogeräte wie Kaffeemaschinen, Kühlschränke, Radios etc.
Zur Erinnerung: die Intention der ursprünglichen Java-Enwicklung zielte nicht auf Desktop Rechner oder gar das Internet.

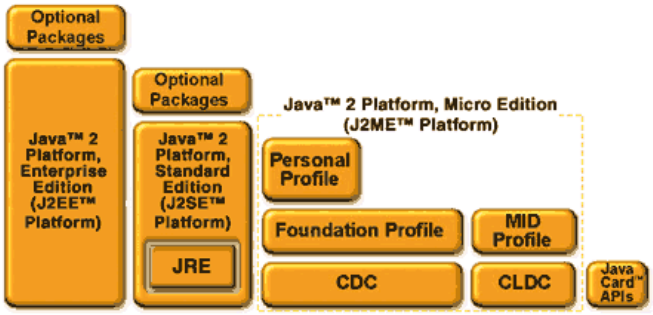


Die Abbildung schematisiert den Aufbau der Java-Plattform. Von der konkreten Plattform der physischen Hardware (Prozessor, etc. und des darauf ausgeführten Betriebsystems) wird durch die plattformspezifische virtuelle Maschine

abstrahiert.
In diese eingebettet ist das *Java-Application Programmers Interface* (Abk. API) --- die sog. *Klassenbibliothek*. Sie ist vollständig in Java realisiert, damit ist sie bereits als plattformunabhängiges Java-Programm ausgelegt. Das API liefert die wesentlichen Grundprimitven zur Erstellung leistungsfähiger Programme.
Auf dieser *virtuellen Plattform* läuft das (Benutzer) Java-Programm --- in Form des Bytecodes --- ab.

Hinweis: Beachten Sie die Unterscheidung zwischen Java-Plattform und Ausführungsplattform (=Betriebsystem und Hardware) einer Java-Applikation

Zur Applikationserstellung mit der Java-Plattform bietet SUN drei verschiedene Ausbaustufen (*Editionen*) an, die in der nachfolgenden Abbildung zusammengestellt sind.

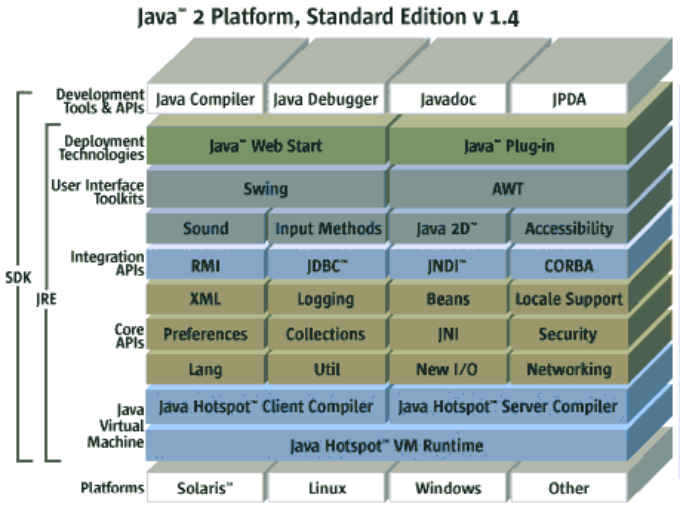


Diese Editionen basieren alle auf demselben Java-Sprachkern, verfügen jedoch über verschiedene Standard-APIs, die auf die jeweils adressierte Problemstellung zugeschnitten sind. Im Einzelnen sind dies:

- **Java 2 Enterprise Edition**: Erstellung von Webapplikationen, die durch einen Application Server ausgeführt werden.
- **Java 2 Standard Edition**: Erstellung von Clientapplikationen zur Ausführung auf Desktopmaschinen.
- **Java 2 Micro Edition**: Erstellung von Applikationen für ressourcenbeschränkte Geräte wie Mobiltelephone, PDAs, etc.

Wir werden uns nachfolgend auf die Betrachtung des Java-Sprachkernes, sowie ausgewählter APIs beschränken, überwiegend in allen drei Editionen, jedoch mindestens in J2EE und J2SE, zur Verfügung stehen.

Das Sun *Java Development Kit* (JDK)



Das JDK stellt eine Referenzimplementierung der kompletten Java-Plattform zur Verfügung.
Seine Hauptbestandteile sind:

- Java-Compiler: `javac`
  erzeugt Byte-Code
  Minimalaufruf: `javac example.java`
  Beispiel: `javac HelloWorld.java` erzeugt die Dateien HelloWorld.class und SayHello.class
- Java-Interpreter: `java`
  Laufzeitumgebung zur Ausführung der Bytecode-Dateien
  Minimalaufruf: `java example`
  Wichtig: die Dateiextension `.class` darf *nicht* angegeben werden!
  Beispiel: `java HelloWorld` führt die Applikation aus.
- Applet-Viewer: `appletviewer`
  Dient zum Testen von Java-Applets ohne Web-Browser
  Minimalaufruf: `appletviewer example.html`
  Hinweis: Das Applet muß in einer Minimal-HTML-Seite referenziert werden, eine losgelößte Ausführung ist nicht möglich.
  Beispiel (absolutes Minimalapplet (HelloWorldApplet1.java): `appletviewer HelloWorldApplet1.html`
- Dokumentationswerkzeug: `javadoc`
  Erzeugt standardisierte HTML-Dokumentation aus Java-Quellcode und darin enthaltenen speziellen formalisierten Kommentaren.

Minimalaufruf: `javadoc example.java`
Beispiel: `javadoc -private -version -author -windowtitle "Beispiel einer JavaDoc generierten Dokumentation" Hello World.java`
Generiert verschiedenste Dokumentationsdateien im HTML- bzw. CSS-Format. Als Einstiegspunkt empfiehlt sich die Datei index.html

- Textmode-Debugger: `jdb`
  Einfacher textorientierter Debugger
  Minimalaufruf: `jdb example`
- Disassembler: `javap`
  Rückübersetzer zur Wiedergewinnung von Java-Quellcode aus Bytecode-Dateien
  Minimalaufruf: `javap example`

Die Rückübersetzung des Compilates HelloWorld.class mit `javap -c HelloWorld` liefert:

```
(1)Compiled from HelloWorld.java
(2)public class HelloWorld extends java.lang.Object {
(3)    public HelloWorld();
(4)    public static void main(java.lang.String[]);
(5)}
(6)
(7)Method HelloWorld()
(8)   0 aload_0
(9)   1 invokespecial #1 &lt;Method java.lang.Object()&gt;
(10)   4 return
(11)
(12)Method void main(java.lang.String[])
(13)   0 getstatic #2 &lt;Field java.io.PrintStream out&gt;
(14)   3 ldc #3 &lt;String "Hello World!"&gt;
(15)   5 invokevirtual #4 &lt;Method void println(java.lang.String)&gt;
(16)   8 return
```

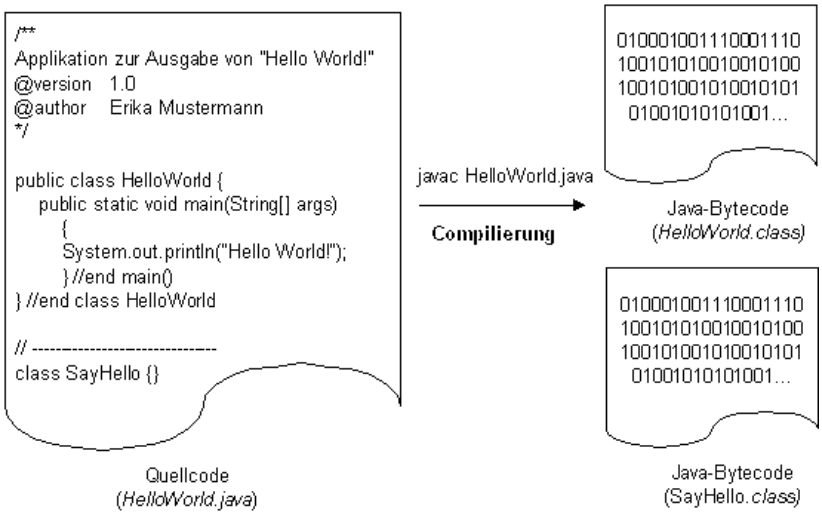
Beispiel 1: Decompilierung mit javap

Anmerkung: Zum Investitionsschutz kommerziell erstellter Java-Software sind Applikationen verfügbar, die den generierten Bytecode so nachbearbeiten, daß eine Decompilierung deutlich erschwert wird, oder die Ausgabe unbrauchbar wird. Die Ausführbarkeit des Bytecodes wird hierdurch nicht beeinträchtigt.

Zusätzlich ist ein eigenständiger Java-Interpeter, das sog. *Java Runtime Environment* (Abk. JRE) verfügbar. Sein Minimalxaufruf lautet `jre example`. Diese Applikation ist nicht Bestandteil der Standard Java-Plattform (JDK v1.3) und ist separat kostenlos über die Sun Java-Homepage beziehbar.

1.4 Vom Quellcode zum lauffähigen Programm

Die Programmentwicklung vollzieht sich im klassischen edit-build-run-Zyklus. Als Besonderheit generiert der Java-Compiler je eine Bytecode-Datei (Dateiextension `class`) für jede zugreifbare Klasse innerhalb der Quelldatei.
Die entstehenden Class-Dateien werden durch die Laufzeitumgebung interpretativ ausgeführt.



Im Beispiel enthält die Quellcodedatei HelloWorld.java die beiden Klassendateien `HelloWorld` und `SayHello`.
Diejenige Klasse, welche die Main-Methode beinhaltet muß identisch der sie enthaltenden Quelldatei benannt sein -- im Beispiel `HelloWorld`.
Mit `javac HelloWorld.java` wird die Datei übersetzt, und die beiden Class-Dateien `HelloWorld.class` und `SayHello.class` erzeugt.
Die Programmausführung erfolgt durch Absetzen von `java HelloWorld` auf der Kommandozeile. (Achtung! Keine Dateiextension angeben!)

Der Aufruf `java SayHello` führt hingegen wegen des Fehlens der Main-Methode in der Klasse `SayHello` zur Fehlermeldung `Exception in thread "main" java.lang.NoSuchMethodError: main`.

2. Syntax und Semantik der Programmiersprache Java

2.1 C, C++, C# und Java

Wie bereits in der Einführung angedeutet wurde Java nahe an der Syntax der verbreiteten hybrid objektorientiert-prozeduralen Sprache C++ entwickelt. Jedoch mit der Einschränkung, deren mitunter krypischen Syntax deutlich zu vereinfachen. Überdies wurde auf wesentliche dort anzutreffende Sprachmerkmale verzichtet.

Die Java Language Specification führt bereits im Vorwort aus, daß sich die Java-Sprachentwickler zwar in einer Vielzahl von Punkten an C und C++ orientierten, jedoch der Sprachaufbau stark von diesen Beiden Vätern abweicht. Aus praktischer Motivation heraus wurde beim Sprachdesign auf die Einführung neuer und ungetesteter Sprachelemente verzichtet.
Wie erwähnt ist Java als streng typisierte Sprache ausgelegt. Daher können die meisten Typfehler bereits zur Übersetzungszeit erkannt werden. Technisch gesehen meint strenge Typisierung (auch *statische* Typisierung), daß der Typ einer Variable während der Programmausführung nicht verändert werden kann. Durch statische Typanalyse währen des Compilierungsvorganges können Typfehler erkannt werden. Durch polymorphe Aufrufe kann es jedoch auch während der Programmausführung zu Typfehlern kommen, da hierbei der dynamische Typbindung zum Einsatz kommt.

Anders als C und C++ verfügt Java über keinerlei systemnahe Konstrukte. Direkte Hardwarezugriffe und Systemprogrammierung im Allgemeinen kann daher mit nicht realisiert werden.
Die Sprachväter begründen dies mit der Transparenz einer *high-level*-Sprache, die keinerlei Rückschlüsse auf die darunterliegende Hardware zulassen sollte.

Anders als C/C++ verfügt Java über automatische Speicherbereingung (engl. *garbage collection*), welche die fehlerträchtigen Speicheroperationen (insbesodnere *free* und *delete*) obsolet werden läßt. Konsequenterweise läßt Java die Allokation beliebiger Speicherbereiche nicht zu. Nicht mehr benötigte Speicherplätze (Variablen) können zwar durch den Programmierer (durch die explizite Zuweisung von `NULL`) als nicht mehr benötigt gekennzeichnet werden, Freigabefunktionen stellt die Sprache jedoch nicht zur Verfügung.

Bekannte, und berüchtigte, Fehlerquellen wie Arrayzugriffe ohne vorherige Indexprüfung, variabel lange Parameterlisten oder Zeiger nebst Zeigerarithmetik existieren nicht, und verleihen dem Sprachentwurf dadurch zusätzliche Sicherheit.
Der Verzicht auf (explizite) Zeiger zieht die Behandlung aller Methodenparameter als Werte (call-by-value) nach sich. Ebenso wurde auf Umgebungseigenschaften wie den Präprozessor und Includedateien vollständig verzichtet wurde. die Strukturierung des Java-Quellcodes kann über ein eigenes Paketkonzept erfolgen.

Nützlichkeiten wie die Lockerung des Variablendefinitionszwanges am Blockanfang wurden beibehalten. Selbiges gilt für die Aufhebung der Trennung zwischen Deklaration und Definition bei einfachen Variablen, wie sie bereits in anderen Sprachen verwirklicht ist. Für Objekte existiert diese Trennung -- sinnvollerweise -- weiterhin fort.

Hinsichtlich objektorientierter Konzepte geht Java deutlich über C++ hinaus. Dergestalt, daß weder klassenlos Programme entwickelt werden können, noch Structs und Unions als Zwitter zwischen Variablen und Klassen implementiert sind. Die Prämisse strengerer Objektorientierung erklärt auch das Verbot globaler Variablen und Methoden. Allerdings sind die primitiven skalaren Datentypen (wie `int`, `char`, etc.) nicht als Objekte realisisiert.
Analog C++ können Methodennamen überladen werden (Ad-hoc Polymorphie). Zur Eindeutigkeitsidentifikation wird die Signatur (gebildet aus Methodennamen und übergabeparametern) herangezogen. Die überladung von Operatoren, wie in C++ möglicht, ist nicht vorgesehen.
Der in C++ realisierte Vererbungsmechanismus wurde unter der Einschränkung übernommen, Mehrfachvererbung zu verbieten. Um trotz dieser Restriktion sinnvolle Anwendungsentwicklung betreiben zu können wurde Java als zusätzliches Sprach-Konzept die Schnittstelle (engl. *Interface*) hinzugefügt. Hiervon können eine Klasse beliebig viele implementiert werden.
Im Gegensatz zu C++ erben Klassen ohne ausdrücklich angegebene Elternklasse automatisch per Vorgabe von `java.lang.Object`.

Parametrische Polymorphie in Form von Templates, wie in C++ anzutreffen, ist in Java erst ab der Sprachversion 1.5 realisiert.

Von C++ wurde die Fehlerbehandlung in Form von Ausnahmen (engl. *exceptions*) übernommen. Hierdurch können Fehlerereignisse zentralisiert behandelt werden. Zusätzlich wird der Kontrollfluß von Fehlerbehandlungscode bereinigt und dadurch übersichtlicher.

Offensichtlich ist die übernahme des Kommentierungsstils von C und C++. So können alle Kommentarkonstrukte wie in den vorgenannten Sprachen üblich eingesetzt werden.
Zusätzlich wird ein separater Kommentarstil (`/**...*/`) für Quellen automatisierter Dokumentation angeboten. Zwischen den so abgegrenzten Regionen können vorgegebene Marken plaziert werden; diese werden beispielsweise durch das JDK-Werkzeug javadoc verarbeitet.

Augenfälligstes abgrenzendes Merkmal von Java gegenüber C/C++ ist es, daß kein plattformspezifischer Maschinencode als Resultat des Compilierungsvorganges erzeugt wird, sondern binärer Bytecode genannte Zwischenrepräsentation. Diese wird durch die Java Virtual Machine zur Ausführungzeit interpretativ abgearbeitet.
Anmerkung: Bytecode ist nicht Java-spezifisch, sondern kann auch durch andere Sprachen erzeugt werden, was mit unter auch geschicht.

Microsofts C#

Die durch Microsoft entwickelte Sprache *C#* (sprich: *C sharp*) weißt einige interessante Parallelen zu Java auf, führt jedoch auch neue Konzepte ein.
Jenseits der Einordnung in die Komponentenarchitektur der .NET-Plattform, welche nur schwer mit der der Java-Plattform (insbesondere der J2EE-Plattform) verglichen werden kann, stellt C# jedoch eine vollwertige -- sehr stark an Java orientierte -- Programmiersprache dar. Bereits das, zu erwartende, Standardbeispiel der HelloWorld-Applikation läßt die enge Verwandschaft, abzulesen an der Syntax deutlich werden (Vergleiche HelloWorld in Java, in in C#).
Im Gegensatz zu C# ist Java auf der Programmierebene nicht *rein* objektorientiert wie beispielsweise SmallTalk. So treten auch hier die primitiven Datentypen als nicht-objektartige Werte auf.

Ebenso wie in Java ist ausschließlich Einfachvererung zugelassen, ergänzt wird diese um Schnittstellen. Auch C# erlaubt es einer Klasse mehrere Schnittstellen zu implementieren. Die Syntax unterscheidet jedoch nicht mehr explizit zwischen Schnittstellenimplementierung und Erben von einer Klasse (siehe Beispiel). Unverändert zu Java wird auch jede Klasse, die über keine Elternklasse verfügt automatisch als Subklasse von `object` eingeordnet. Blattknoten einer Vererbungshierarchie (d.h. Klassen für die durch den Programmierer vorgegeben wird, sie sollen nicht weiter durch Ableitung spezialisiert werden) können mit dem Schlüsselwort `sealed` -- in Java: `final` -- gekennzeichnet werden.

Zusätzlich zu Klassen beinhaltet C#, wieder, den Sprachmechanismus der Struktur (`struct`). Hierüber wird die Differenzierung zwischen Wert- und Referenztypen realisiert. Während Klassen, Java-konform Referenztypen bezeichnen, übernehmen Strukturen die der Werttypen. Desweiteren werden Structs durch den Compiler als Bestandteile umgebender Objekte übersetzt. Durch diese Charakteristika unterscheiden sich C#-Structs stark von den dortigen Klassenstrukturen, weshalb sie auch nicht analog zu deren Schema verwirklicht sind. Strukturen können weder erben noch vererbt werden, lediglich die Möglichkeit Schnittstellen zu implementieren bleibt erhalten.

Deutlich komfortabler als in Java fällt jedoch der Umgang mit Referenz- und Werttypen aus. Das in C# angebotene *boxing* und *unboxing* übertrifft deutlich die Benutzungsfreundlichkeit der Java-Wrapperklassen (Beispiel). Dieser Mißstand wurde jedoch in der Javasprachversion 1.5 behoben, die dieses Konzept ebenfalls einführt.

Erstmals reichert C# auch den Sprachumfang C/C++-basierter Sprachen um neue syntaktische Konstrukte an. Hierunter fallen beispielsweise Schlüsselworte zur Definition von Delegationsobjekten, die als solche explizit im Programmcode deklariert werden können.

Wie Java compiliert auch C# in eine Zwischenrepräsentation. Allerdings mit dem Unterschied, daß diese nicht interpretativ abgearbeitet wird, sondern erneut in maschinenspezifisches natives Format übersetzt wird.

Die Tabelle stellt einige Sprachmerkmale von C++, Java und C# gegenüber
(Tabelle in Anlehnung an: Eisenecker, U.: Dissonanz oder Wohlklang, in: iX 9/2000, Hannover, 2000, p. 48-51)

Sprachmerkmal	C++	Java	C#
Automatische Speicherbereinigung (garbage collection)	○	✓	✓
Coercion (Typumwandlungspolymorphie)	✓	✓	✓
Globale Methoden	✓	○	○
Inklusionspolymorphie	✓	✓	✓
Operatoroverloading	✓	○	✓
Parametrische Polymorphie (Templates)	✓	○ (1)	○ (2)
Referenztypen	✓	✓	✓
Überladungspolymorphie	✓	✓	✓
Werttypen	✓	✓	✓ *unified type system*

Zeiger	✓	⊘	✓

(1) Ab Version 1.5 für Klassen der Collection API im Sprachumfang enthalten.
(2) Die Aufnahme in die Sprache ist für die Nachfolgeversion des .NET-Framworks 1.1 geplant.
Einen Ausblick auf die geplanten Sprachmerkmale liefert das Projekt *CLRGEN* von Microsoft Research.

2.2 Grundstrukturen

2.2.1 Programmaufbau

Bereits am HelloWorld-Beispiel wird der wesentliche Aufbau einer Javaquellcodedatei sichtbar.

```
(1)public class Minimal {
(2)   public static void main(String[] args) {
(3)      //..
(4)   } //main()
(5)} //class Minimal
```

Beispiel 2: Minimales Java-Programm

Zunächst erfolgt die Definition einer Klasse; im Beispiel `Minimal`.
Vor dem Schlüsselwort `class` ist die Sichtbarkeit auf `public` festgelegt, was eine allgemeine Sichtbar- und Zugrifbarkeit der Klasse bewirkt.
Im Gegensatz zu C++ ist jedoch die Klassendefinition zwingend erforderlich! Es ist nicht möglich Javaquellcode, der keine Klassendefinition enthält, zu übersetzen. Hingegen ist es ohne weiteres möglich gewöhnliche C-Programme mit einem C++-Compiler zu übersetzen.

Darüberhinaus ist es zwingend vorgeschrieben die Quellcodedatei identisch zur `public` deklarierten Klasse zu benennen.

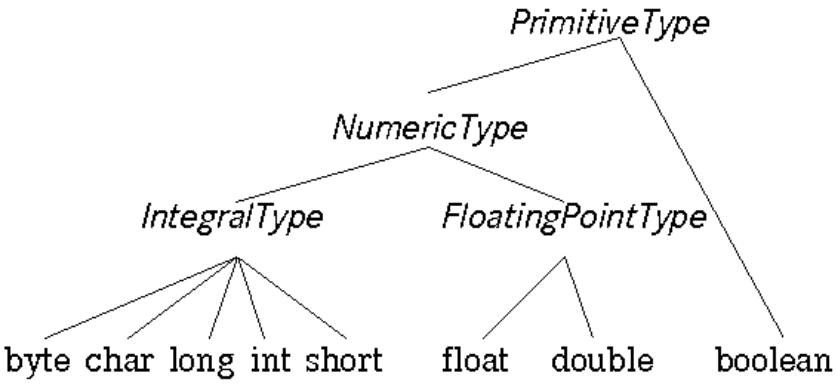
Eine weitere Besonderheit gegenüber C++ stellt die Plazierung der `main`-Funktion dar. Während diese in C++, selbst bei der Verwendung von Klassen (siehe Beispiel (HelloWorld.cpp)), außerhalb jeder Klasse angeschrieben werden muß, um automatisch beim Programmstart ausgeführt zu werden, erzwingt Java die `main`-Methode innerhalb einer `public` deklarierten Klasse.

Die Signatur der `main`-Methode ist mit `public static void main(String[] args)` fixiert. Davon Abweichende Spezifikationen sowohl in Rückgabewert als auch Parameterliste, können zwar in Bytecode übersetzt werden, jedoch wird diese abweichende Main-Methode nicht mehr automatisch durch das Laufzeitsystem aufgerufen.
Auch hier zeigt sich Java deutlich restriktiver als C/C++, die beide beliebige Rückgabetypen und -- in Grenzen -- leicht variierende Parameterlisten zulassen.

Abschließend: Innerhalb der `main`-Methode der `public` deklarierten Klasse kann der auszuführende Code angegeben werden.

2.2.2 Einfache Datentypen

Wie herkömmliche prozedurale Programmiersprachen auch stellt Java primitive Datentypen zur Verfügung. Im Gegensatz zu manchen etablierten objektorientierten Programmiersprachen (z.B. SmallTalk) sind diese Datentypen in Java intern nicht als Objekte realisiert.



Im Einzelnen werden angeboten:

Datentyp	Erklärung	Wertebereich
`boolean`	Wahrheitswert	`true` oder `false`
`char`	Einfaches Zeichen aus dem 16-Bit Unicode Zeichensatz	
`byte`	8-bittige vorzeichenbehaftete Ganzzahl	-2^7 bis 2^7-1 `-128 <= byte <= 127`
`short`	16-bittige vorzeichenbehaftete Ganzzahl	-2^{15} bis $2^{15}-1$ `-32768 <= short <= 32767`
`int`	32-bittige vorzeichenbehaftete Ganzzahl	-2^{31} bis $2^{31}-1$ `-2147483648 <= int <= 2147483647`
`long`	64-bittige vorzeichenbehaftete Ganzzahl	-2^{63} bis $2^{63}-1$ `-9223372036854775808 <= long <= 9223372036854775807`
`float`	32-bittige Gleitkommazahl (nach IEEE 754-1985)	Größtmögliche positive Zahl: 3.40282347e+38f Kleinstmögliche positive Zahl: 1.40239846e-45f
`double`	64-bittige Gleitkommazahl (nach IEEE 754-1985)	Größtmögliche positive Zahl: 1.79769313486231570e+308 Kleinstmögliche positive Zahl: 4.94065645841246544e-324
`Object`	Objektwertiger Datentyp. Jedes Objekt hat (implizit) diesen Typ.	

Siehe Java Language Specification

Anmerkungen:

- Der Wahrheitswert ist nicht wie in C++ üblich als Ganzzahl realisiert.
  Auch eine explizite Typumwandlung ist nicht möglich, und wird zur Übersetzungszeit (als Typfehler) abgefangen (Compilerfehlermeldung: `inconvertible types`).
  Hinweis: Zwar ist im ANSI-C++-Standard ein `bool` als expliziter Datentyp (üblicherweise der Länge 1 Byte) definiert, der jedoch die Wahrheitswerte als Integer-Zahlen ablegt. So gilt: `true == 1` bzw. `false == 0` (siehe Beispiel)
- Die Ganzzahlendarstellung erfolgt immer im Zweierkomplement, weshalb der positive Wertebereich eine Zahl weniger als der negative umfaßt.
- Alle numerischen Typen sind vorzeichenbehaftet. Vorzeichenlose Pendants (wie in C durch Voranstellen eines *unsigned* bildbar) existieren nicht.
- Die Bitlänge aller Datentypen ist festgelegt, und nicht plattformabhängig variierbar.
- Für die beiden Gleitkommatypen exisitert ein Literal zur Darstellung von Plus- und Minus-Unendlich.
  Zusätzlich ein mit `NaN` (*Not a Number*) ein ausgezeichneter Wert für nicht interpretierbare Belegungen.
- `char` ist zwar als Zeichensymbol ausgelegt, kann aber auch in mathematischen Termen anstelle von `short` verwendet werden; hier wird `char` jedoch vorzeichenlos(!) interpretiert.
  Die Konversion von `char` zu `int` erfolgt implizit, die Umgekehrung bedarf jedoch einer ausdrücklichen Typumwandlung.
- Die nicht standardkonforme Implementierung der virtuellen Maschine von Microsoft bietet mit `object` (32-bittige Referenz auf ein Java-Objekt) und `returnAddress` (32-Bit) zwei zusätzliche Datentypen an.
- Ebenso wie in C/C++ üblich können Konstanten durch spezielle Muster typisiert werden.
  - `\uxxxx` für Unicodezeichen (xxxx ist die hexadezimale Zeichennummer)
  - `...l` oder `...L` für Longwerte
  - `0...` für Oktalwerte
  - `0x...` für Hexadezimalwerte
  - `...f` oder `...F` für Floatwerte
  - `...d` oder `...D` für Doublewerte
- Für die primitiven built-in Typen wird keine über- oder Unterlauferkennung angeboten. (Siehe Beispiel Overflow.java).
- Auffallend ist, daß keine Arraytypen existieren. Diese sind in Java nicht als Aggregation primitiver built-in Typen realisiert, sondern als Klasse der API umgesetzt.

```
(1)public class ConstFormats {
(2)     public static void main(String[] args) {
(3)             int i = 052;            //octal
(4)             System.out.println("i as decimal = "+i);
(5)
(6)             i = 0x2a;                       //hexadecimal
(7)             System.out.println("i as decimal = "+i);
(8)
(9)             long l = 0x2aL; //hexadecimal long
(10)            System.out.println("l as decimal = "+l);
(11)
(12)            float f = 0x2af;        //hexadecimal float
(13)            System.out.println("f as decimal = "+f);
(14)    } //main()
(15)} //class ConstFormats
```

Beispiel 3: Verschiedene Deklarationen und Wertebelegungen ConstFormats.java

Das Programm liefert folgende Ausgabe:

```
i as decimal = 42
i as decimal = 42
l as decimal = 42
f as decimal = 687.0
```

VORSICHT! Die hexadezimale Definition des Floatwertes liefert nicht -- wie vielleicht intuitiv zu erwarten -- 42 als ganzzahligen Anteil, sondern die Gleitkommainterpretation des hexadezimal spezifizierten Bitmusters.

Als streng typisierte Sprache muß jede Variable in Java mit einem Typ deklariert werden, ungetypte Variablen existieren nicht.
Nach der Deklaration in einem Programm wird der Variableninhalt auf einen resiervierten, für den Programmierer nicht zugänglichen Wert *undefined* gesetzt. Hierdurch kann der Compiler lesende Referenzierungen vor Wertdefinition zur Übersetzungszeit erkennen.
Die Deklaration geschieht, angelehnt an C/C++ durch Angabe des Typs gefolgt durch den Variablennamen. Optional kann dieses Statement durch die Festlegung eines Vorgabewertes ergänzt werden.

```
(1)public class NotInitialized {
(2)     public static void main (String[] args) {
(3)             int i;
(4)             System.out.println("i= "+i); //i is not initialized yet
(5)     } //main()
(6)} //class notInitialized
```

Beispiel 4: Nicht initialisierte Referenzierung NotInitialized.java

Liefert beim Übersetzen die Fehlermeldung:

```
NotInitialized.java:6: variable i might not have been initialized
                System.out.println("i= "+i); //i is not initialized yet
                                           ^
1 error
```

```
(1)public class CharArithmetic {
(2)     public static void main (String[] args) {
(3)             char c = 'a';
(4)
(5)             System.out.println("c = "+c);
(6)             c++;
(7)             System.out.println("c = "+c);
(8)
(9)             int i = c;
(10)
(11)            System.out.println("i = "+i);
(12)            System.out.println("i = "+ (char) i);
(13)    } //main()
(14)} //class CharArithmetic
```

Beispiel 5: Verwendung von char als numerischer Typ CharArithmetic.java

Das Programm liefert bei der Ausführung folgende Ausgabe:

```
c = a
c = b
i = 98
i = b
```

2.2.3 Operatoren

Java bietet 37 verschiedene Operatoren an, die ihrer Semantik nach mit denen von C/C++ weitestgehend identisch sind (siehe Language Specification).

Je nach Typ auf dem Operator definiert ist, wird unterschieden zwischen: Integralen-, Fließkomma-, Boole'schen- und objektwertigen-Operatoren.

Operatoren auf integralen Typen (

siehe Java API Specification)

- Vergleichsoperatoren, die als Resultat `boolean` liefern:
  - numerischer Vergleich: <, <=, > und >= (siehe Java Language Specification)
  - numerische Gleichheit: == und != (siehe Java Language Specification)
- Numerische Operatoren, die ganzzahligen Wert (`int` oder `long`) zurückliefern:
  - unäres Plus und Minus (siehe Java Language Specification bzw. siehe Java Language Specification).
  - multiplikative Operatoren *, / und % (siehe Java Language Specification).
  - additive Operatoren + und - (siehe Java Language Specification).
  - inkrement Operatoren (Prefix- und Postfixvariante) ++ (siehe Java Language Specification bzw. siehe Java Language Specification).
  - dekrement Operatoren (Prefix- und Postfixvariante) -- (siehe Java Language Specification bzw. siehe Java Language Specification).
  - Shift Operatoren <<, >> bzw. >>> (Einführung von Füllnullen) (siehe Java Language Specification).
    So gilt: `-2 >>> 2 = -1` aber: `-2 >> = 1073741823`
  - bitweises Komplement ~ (siehe Java Language Specification).
  - bitweise Integeroperatoren &, | und ^ (siehe Java Language Specification).
- Konditionaloperator (Kurzschreibweise für `if`) `? :` (siehe Java Language Specification).
- explizite Typumwandlung (*cast*), zur Umwandlung jedes Integraltypen in einen beliebigen numerischen (siehe Java Language Specification, siehe Java Language Specification).

Diese built-in Operatoren nehmen keinerlei Fehlerprüfung oder -meldung vor. Lediglich die beiden Integerdivisionsoperatoren / und % werfen im Fehlerfalle eine `ArithmeticException`-Ausnahme falls der Divisor gleich Null ist (siehe Java Language Specification bzw. siehe Java Language Specification).

Operatoren auf Fließkommatypen (

siehe Java Language Specification)

Auf diesen Typen sind die auch auf integralen Typen zugelassenen arithmetischen-, numerischen Vergleichs-, Inkrement- und Dekrement-Operatoren verfügbar. Ferner existiert der numerische Cast (siehe Java API Specification).

Auch die Fließkommaoperatoren lösen keine Exceptions aus. überlauf wird als positiv Unendlich, Unterlauf entsprechend als negativ Unendlich dargestellt. Liefert die Operation kein mathematisch interpretierbares Resultat, wird der Wert auf `NaN` gesetzt. In der Konsequenz resultiert auch `NaN` falls ein so gesetzter Operand erneut verknüpft wird. Dies gilt nicht für die Gleichheitsoperation.

- Vergleichsoperatoren, die als Resultat `boolean` liefern:
  - numerischer Vergleich: <, <=, > und >= (siehe Java Language Specification)
  - numerische Gleichheit: == und != (siehe Java Language Specification)
- Numerische Operatoren, die ganzzahligen Wert (`float` oder `double`) zurückliefern:
  - unäres Plus und Minus (siehe Java Language Specification bzw. siehe Java Language Specification).
  - multiplikative Operatoren *, / und % (siehe Java Language Specification).
  - additive Operatoren + und - (siehe Java Language Specification).
  - inkrement Operatoren (Prefix- und Postfixvariante) ++ (siehe Java Language Specification bzw. siehe Java Language Specification).
  - dekrement Operatoren (Prefix- und Postfixvariante) -- (siehe Java Language Specification bzw. siehe Java Language Specification).
- Konditionaloperator (Kurzschreibweise für `if`) `? :` (siehe Java Language Specification).
- explizite Typumwandlung (*cast*), zur Umwandlung jedes Integraltypen in einen beliebigen numerischen (siehe Java Language Specification, siehe Java Language Specification).

Anmerkung: Identisch zu ANSI C/C++ wurde der unäre Operator + lediglich aus Symmetriegründen zum unären eingeführt.

Operationen auf Boole'schen Typen

Auf Boole'schen Typen sind alle relationalen und logischen Operatoren zugelassen.
Als Operand des Konditionaloperators können neben Boole'schen Werten auch Integralwerte angegeben werden. Diese werden gemäß C-Konvention zu `true` konvertiert sofern sie ungleich Null sind, andernfalls zu `false`.

- relationale Operatoren (Gleichheit und Ungleichheit) == und != (siehe Java Language Specification).
- logisches Komplement ! (siehe Java Language Specification).
- logische Operatoren &, ^ und | (siehe Java Language Specification).
- konditionelles-und und konditionelles-oder && bzw. || (siehe Java Language Specification bzw. siehe Java Language Specification).
  Die Auswertung erfolgt im sog. `short circuit`-Verfahren. Hierbei wird nicht zwingend der gesamte Ausdruck ausgewertet. Bei der *Ver-und-ung* wird abgebrochen, sobald ein falsches Ergebnis vorliegt; bzw. bei der *Ver-oder-ung* sobald ein wahres Ergebnis vorliegt.

- Konditionaloperator `? :` (siehe Java Language Specification)

Operatoren auf objektwertigen Typen

- `+` zur Stringkonkatenation. Ist einer der beiden angegebenen Operaden nicht vom Typ String, so wird zur Laufzeit eine Stringkonversion durchgeführt (siehe Java Language Specification). Diese Stringkonversion ist immer möglich, da alle Objekte von `java.lang.Object` die Methode `toString` erben.
  Anmerkung: hierbei handelt es sich strenggenommen um einen überladenen Operator
- Abfragen auf den Typ eines Objekts können durch den `instanceof`-Operator erledigt werden. Er liefert einen Wahrheitswert zurück.
  Syntax: *`object`* `instanceof` *`class`*.

Schlussbemerkungen

- Wie in C/C++ auch existieren die abkürzenden Zuweisungsschreibweisen: `+=, -=, *=, /=, &=, |=, ^=, %=, <<=, >>=, >>>=`.
- Die aus C/C++ bekannten Speicherzellen-bezogenen Operatoren `*` und `&` werden wegen des Fehlens expliziter Speicherreferenzen nicht unterstützt.
- Ebenso der `sizeof`-Operator. Für ihn existiert kein Anwendungsfall, da einerseits die Größer der built-in Typen festgelegt ist, andererseits keine Speicherblöcke wahlfrei allokiert werden können.
- Die Operanden aller diskutierten Operatoren, außer `&&, ||` und `? :`, werden vor der Ausführung des Operators ausgewertet.
  Dies gilt insbesondere für Operatoren die eine Exception auslösen können. Eine solche wird ggf. nach Auswertung aller Operaden erzeugt.

Operatorpräzedenz

Es gelten folgende Operatorpräzedenzen (geordnet von oben (entspricht höchster Präzedenz) nach unten (niedrigster Präzedenz)):

Operator	Symbol
Postfix-Operatoren	`[] . (params) expr++ expr--`
unäre Operatoren	`++expr --expr +expr -expr ~ !`
Erzeugung oder Typumwandlung	`new (type )expr`
Multiplikationsoperatoren	`* / %`
Additionsoperatoren	`+ -`
Verschiebeoperatoren	`<< >> >>>`
Vergleichsoperatoren	`< > <= >= instanceof`
Gleichheitsoperatoren	`== !=`
Bitoperator Und	`&`
Bitoperator exklusives Oder	`^`
Bitoperator inklusives Oder	`|`
logisches Und	`&&`
logisches Oder	`||`
Konditionaloperator	`? :`
Zuweisungsoperatoren	`= += -= *= /= %= >>= <<= >>>= &= ^= |=`

Automatische Typkonversion

Die automatische Typkonversion wird durch den Compiler bzw. das Laufzeitsystem immer dann angewandt, wenn nicht alle Operanden typgleich sind. Dies ist beispielsweise bei der Multiplikation einer Int-Zahl mit einem Fließkommawert der Fall.
Die automatische Typumwandlung ist im Wesentlichen identisch zur in C/C++ realisierten ausgelegt.

	byte	char	short	int	long	float	double	boolean
byte	int	int	int	int	long	float	double	
char		int	int	int	long	float	double	
short			int	int	long	float	double	
int				int	long	float	double	
long					long	float	double	
float						float	double	
double							double	
boolean								boolean

Die Tabelle stellt die automatischen Typkonversionen zur Festlegung des Ausdruckstyps dar. Da alle Operatoren kommutativ sind, ist die Tabelle symmetrisch zur Hauptdiagonale (nur die obere Dreiecksmatrix ist aus übersichtlichkeitsgründen ausgefüllt).
Alle Typkonversionen werden statisch und operatorunabhängig durchgeführt, d.h. mögliche Typfehler werden zur Compilezeit erkannt und gemeldet.

2.3 Kontrollstrukturen

2.3.1 Selektion und Mehrfachselektion -- das if und case-Statement

Das `if`-Statement ist in Java analog der aus C/C++ bekannten Semantik und Syntax realisiert (Siehe Language Specification):

```
IfThenStatement:
   if ( Expression ) Statement

IfThenElseStatement:
   if ( Expression ) StatementNoShortIf else Statement

IfThenElseStatementNoShortIf:
   if ( Expression ) StatementNoShortIf else StatementNoShortIf
```

```
(1)public class IfTest {
(2)     public static void main(String[] args) {
(3)             int i = 42;
(4)
(5)             if (1==1)
(6)                     if (i < 50)
(7)                             i++;
(8)                     else
(9)                             i--;
(10)    } //main()
(11)} //end class IfTest
```

Beispiel 6: If-Statement mit dangling else IfTest.java

Leider wurde in Java die Chance zur Ausmerzung des lästigen und fehlerträchtigen *dangling else-Problems* nicht genutzt. Daher wird -- konform zu C/C++ -- ein *else* immer dem letzten vorhergehenden `if` zugeordnet.
Strenger im Vergleich zu C/C++ ist hingegen die Typisierung der `Expression` als Bedingung innerhalb der Verzweigung gefaßt. Hier ist zwingend der Typ `boolean` erforderlich.

Auch das `switch`-Statement ist identisch zum C/C++-Analogon realisiert.
Ebenso wie dort müssen die Alternativzweige mit einem expliziten `break` abgeschlossen werden.
Innerhalb der `case`-Verzweigungen sind nur konstante Ausdrücke der Typen `char`, `byte`, `short` oder `int` zugelassen.
(siehe Java Language Specification).

Syntax:

```
SwitchStatement:
   switch ( Expression ) SwitchBlock

SwitchBlock:
   { SwitchBlockStatementGroups_optional SwitchLabels_optional }

SwitchBlockStatementGroups:
   SwitchBlockStatementGroup
   SwitchBlockStatementGroups SwitchBlockStatementGroup

SwitchBlockStatementGroup:
   SwitchLabels BlockStatements

SwitchLabels:
   SwitchLabel
   SwitchLabels SwitchLabel

SwitchLabel:
   case ConstantExpression :
   default :
```

Beispiel eines switch-Statements:

```
(1)public class SwitchTest {
(2)     public static void main(String[] args)  {
(3)             int i = 42;
(4)
(5)             here:
(6)             switch (i) {
(7)                     case 0: i++;
(8)                                             break here;
(9)                     case 1:
(10)                    case 2: i--;
(11)                                            break;
(12)                    default:
(13)                                            i *= 10;
(14)            } //switch
(15)    } //main()
(16)} //class SwitchTest
```

Beispiel 7: Switch-Statement SwitchTest.java

Anmerkungen:

- Der break-Anweisung kann eine Marke nachgestellt werden, die angibt auf welcher Schachtelungsebene fortgesetzt werden soll (siehe Java Language Specification). Hierbei sind jedoch nur Rückwärtsreferenzen zugelassen.
- Soll ein case-Körper für mehrere Alternativen gelten, so müssen diese explizit mit einleitendem case aufgeführt werden. Pro case ist jeweils nur genau eine konstante Bedingung zugelassen.
- Obwohl switch-Statement syntaktisch unverändert von C/C++ übernommen wurde, existieren in Java zustäzliche Restriktionen, welche die Kompatibilität einschränken.
  So muß der der Rumpf einer switch-Anweisung die zugehörigen case-Alternativen direkt enthalten; Konstruktionen wie *duff's device*siehe Java Language Specification sind daher nicht compilierbar.
- Die switch-Bedingung muß vom Typ byte, short, char oder int sein.

2.3.2 Iteration -- for-, do-while-Schleifen

Java bietet die bereits in C/C++ eingeführten Schleifenkonstrukte

- for (siehe Java Language Specification)
- while (siehe Java Language Specification)
- do (siehe Java Language Specification)
- break und continue (siehe Java Language Specification)

an.

Die **for-Struktur** besteht aus drei optionalen Komponenten: Initialisierung(en, Fortsetzungsbedingung(en) und Wertaktualisierung(en) (auch Reinitialisierung(en)).
Ebenso wie in C/C++ sind diese durch Semikola voneinander abgetrennt.
Im Initialisierungs- und Fortsetzungsteil sind mehrere, durch Komma separierte, Ausdrücke zuglassen. Im Fortsetzungsbedingungsteil hingegen nicht! Hier müssen mehrere Bedingungen durch logisches *Und* (&&) verbunden werden.

```
(1)public class ForTest {
(2)     public static void main(String[] args) {
(3)             for (int i=1, j=-2; i <= 10 && j <=0 ; i+=2, j++)
(4)                     System.out.println("i = "+i+"  j = "+j);
(5)     } //main()
(6)} //class ForTest
```

Beispiel 8: Eine for-Schleife ForTest.java

Alle Schleifen können vorzeitig, d.h. trotz gültiger Fortsetzungsbedingung(en) wahlfrei mit der break-Anweisung verlassen werden.
Generell versucht die break-Anweisung hinter dem nächstliegenden (d.h. innsteren erreichbaren) schließenden Schleifenkonstrukt fortzusetzen. Der Einsatz dieser Anweisung außerhalb der beschriebenen Strukturen wird per übersetzungsfehler verhindert.

Zusätzlich können Sprungziele durch Marken explizit benannt werden. Diese *labels* werden durch einen eineindeutigen Namen abgeschlossen von einem Doppelpunkt symbolisiert.
Es sind jedoch nur „Rückwärtssprünge“ möglich. (siehe Java Language Specification)

```
(1)public class BreakTest {
(2)     public static void main(String[] args) {
(3)             myLabel:
(4)             for (int i=1; i<=100; i++) {
(5)                     for(int j=1; j<=100; j++) {
(6)                             if (i*j >= 42) {
(7)                                     System.out.println("exiting loop");
(8)                                     break myLabel;
(9)                             } //if
(10)                    } //for
(11)            }//for
(12)            System.out.println("continuing...");
(13)    } //main()
(14)} //class BreakTest
```

Beispiel 9: Verlassen einer Schleife mit break BreakTest.java

Im Gegensatz zu vielen prozeduralen Programmiersprachen verfügt Java über kein goto-Statement welche beliebige Sprünge erlauben würde.
Allerdings scheint beim Sprachdesign durchaus an diese Möglichkeit gedacht worden zu sein. Findet sich doch goto in der Aufzählung der reservierten Schlüsselworte (siehe Java Language Specification) und im Index (siehe Java Language Specification).
Die Referenzimplementierung des Javacompiliers von SUN nutzt das Schlüsselwort lediglich um den illegalen Beginn eines Ausdrucks zum Übersetzungszeitpunkt anzuzeigen.

Zum vorzeitigen Rücksprung aus dem Schleifenkörper ist das das continue-Statement definiert.
Nach seiner Ausführung erfolgt unverzüglich die Auswertung der Fortsetzungsbedingung, unter Auslassung aller folgenden Anweisungen des aktuellen Blocks.
Analog der break-Anweisung kann die Schachtelungstiefe in der fortgefahren werden soll durch Angabe eines Labels gesteuert werden. (siehe Java Language Specification)

Kopf- und Fußgesteuerte Schleifen, werden mit der while ... do-Konstruktion analog zu C/C++ realisiert.
Auch hier wird der Typ der Bedingung bereits zur Übersetzungszeit auf boolean geprüft.

Häufige Anwendungsform von Schleifen ist die Traversierung einer Objektmenge.
Das nachfolgende Beispiel zeigt die „klassische“ Vorgehensweise zur Ausgabe aller Elemente einer Aufzählung.
Hierzu wird zunächst Elementanzahl ermittelt und anschließend in einer for-Schleife, beginnend mit der kleinsten Indexnummer (0) bis zur ermittelten Obergrenze inkrementiert. An jeder Indexposition wird das unter dieser Ordnungsnummer abgelegte Element ausgegeben.
Der Vorgehensweise unterliegt den beiden Grundannahmen, daß die Werte einerseits kontinuierlich (d.h. keine Indexposition ist unbesetzt) abgelegt sind. Und zweitens, daß die Wertemenge ab der Indexposition 0 aufsteigend abgelegt ist. Beide Annahmen können für die durch die Java-Standardklasse Vector realisierte Aufzählung als erfüllt angenommen werden.

```
(1)import java.util.Vector;
(2)
(3)public class NaiveIteration {
(4)     public static void main(String[] args) {
(5)             Vector v = new Vector();
(6)             v.add(new String("Berta"));
(7)             v.add(new String("Anna"));
(8)             v.add(new String("Cäsar"));
(9)
(10)            for (int i=0; i<v.size(); i++) {
(11)            System.out.println(v.get(i));
(12)            } //for
(13)    } //main()
(14)} //class NaiveIteration
```

Beispiel 10: Naive Traversierung einer Objektmenge NaiveIteration.java

Die Lösung leistet zwar das gewünschte, jedoch wirkt die Schleifenkonstruktion unnötig komplex. Überdies zwingt sie den Programmierer Daten abzuspeichern (in Form des Schleifenzählers `i` sowie der implizit angeforderten unbenannten Speicherstelle zur Ablage der Elementanzahl), die nur zum Zweck der Mengentraversierung benötigt werden.

Um die Umsetzung dieser häufig auftretenden Standardsituation zu vereinfachen bietet die Standard-API die Schnittstelle `Iterator` an. Sie entbindent den Programmierer von der expliziten Ermittlung der Elementanzahl, sowie der indexgebundenen Traversierung.

Das nachfolgende Beispiel zeigt die Integration des `Iterator`-basierten Ansatzes für die bekannte Mengentraversierungsaufgabe

```
(1)import java.util.Iterator;
(2)import java.util.Vector;
(3)
(4)public class IteratorTest {
(5)     public static void main(String[] args) {
(6)             Vector v = new Vector();
(7)             v.add(new String("Berta"));
(8)             v.add(new String("Anna"));
(9)             v.add(new String("Cäsar"));
(10)
(11)            for (Iterator i = v.iterator() ; i.hasNext() ;) {
(12)            System.out.println(i.next());
(13)            } //for
(14)   } //main()
(15)} //class IteratorTest
```

Beispiel 11: Iterator-basierte Traversierung einer Objektmenge IteratorTest.java

Die Lösung enthebt den Programmierer zwar vom Aufwand die Elementanzahl explizit zu ermitteln und einem Speicherplatz zuzuweisen, jedoch wird mit dem `Iterator`-Objekt immernoch benannter Speicher (in Form der Variable e) angefordert, der ausschließlich der schleifeninternen Logik dient.

Erweiterte Schleifen-Syntax

Zur Behebung des Übelstandes der unnötigen expliziten Informationsermittlung im vorigen Beispiel existiert seit Java Version 1.5 eine abkürzenden Schreibweise für Mengentraversierungen.

```
(1)import java.util.Enumeration;
(2)import java.util.Vector;
(3)
(4)public class NewIteration {
(5)     public static void main(String[] args) {
(6)             Vector v = new Vector();
(7)             v.add(new String("Berta"));
(8)             v.add(new String("Anna"));
(9)             v.add(new String("Cäsar"));
(10)
(11)            for (Object o : v) {
(12)            System.out.println( o );
(13)            } //for
(14)   } //main()
(15)} //class NewIteration
```

Beispiel 12: Traversierung einer Objektmenge NewIteration.java

Das Beispiel zeigt die alternative Schreibweise, welche keine unötigen, d.h. im Schleifenrumpf nicht benötigten, Daten ermittelt. Der Ausdruck innerhalb der for-Klammen typisiert die Elemente der Menge v als `Object` und weist sie temporär (konkret: für jeden Schleifendurchlauf einen Wert) der Variable o zu.

Die Entnahme aus der Objektmenge erfolgt unter Nutzung des allgemeinen Typs `Object` anstatt direkt auf den konkreten Typ `String` zurückzugreifen. Diese Asymmetrie erklärt sich aus Nutzung der Klasse `Vector` ohne Verwendung der in der Programmiersprache vorhandenen Generizitätsmechanismen. Nähere Ausführungen zu den damit verbundenen Möglichkeiten finden sich im Kapitel Parametrische Polymorphie/Generics.

2.3.3 Ausnahmen und ihre Behandlung -- Exception Handling

Die Ausnahmebehandlung von Java ist konzeptionell und syntaktisch eng and das Pendant des ANSI-C++-Standards angelehnt. (siehe Java Language Specification).

Drei generelle Vorteile ergeben sich durch Verwendung von Ausnahmen:

- Separierung von Code und Fehlerbehandlung (siehe Java Tutorial)
- Propagierung des Ausnahmezustandes; d.h. Behandlung muß nicht lokal in der fehlerauslösenden Routine erfolgen (siehe Java Tutorial
- Bildung von Fehlerklassen durch Gruppierung gleichartiger Ausnahmen (siehe Java Tutorial)

Die generelle Syntax kann wie folgt beschrieben werden:

```
try {
   //statements which may throw an exception
   //... or create and throw an exception object manually
} catch (exceptionFoo e) {
   //handling of exception of type exceptionFoo
   //exception object is named e
} catch (exceptionBar e) {
   //handling of exception of type exceptionBar
   //exception object is named e
} catch (Exception e) {
   //handling all exceptions not handled by specialized handlers yet
   //exception object is named e
} finally {
   //executed regardless the execution state of the previous try block
}//finally
```

```
(1)public class ExceptionHandlingTest1 {
(2)     public static void main(String[] args)  {
(3)             for (int i=2; i>=0; i--)
(4)                     System.out.println(42/i);
(5)     } //main()
(6)} //class ExceptionHandlingTest1
```

Beispiel 13: Exception auslösender Code ExceptionHandlingTest1.java

Das Beispiel illustriert das Auftreten einer *java.lang.ArithmeticException*, verursacht durch die Division durch Null.

Alle in einen `try`-Block eingeschlossenen Anweisungen werden „überwacht“ ausgeführt. Das bedeutet, es erfolgt kein Programmabbruch beim Auftreten einer Ausnahme, sondern der direkte Ansprung eines *exception handlers*. Exception Handler werden durch `catch`-Blöcke implementiert. Der Typ der zu behandelden Exception wird als übergabeparameter angegeben. Das Laufzeitsystem sorgt für Auswahl der korrekten (d.h. zuständigen) Behandlungsroutine.

Alle Ausnahmen erben von der Klasse `Exception`. Hierdurch kann auch ein Vorgabe-Exception-Handler installiert werden. Das folgende Beispiel zeigt diesen im Anschluß an die Behandlungsroutine der `ArithmeticException` Ausnahme. Eine übersicht der in der Java-API definierten Ausnahme findet sich in: (siehe Java API Specification)
Der Ausdruck `catch (Exception e)` übernimmt hierbei die Rolle des aus C++ bekannten `catch(...)`.

```
(1)public class ExceptionHandlingTest2 {
(2)     public static void main(String[] args) {
(3)             try {
(4)                     for (int i=2; i>=0; i--)
(5)                             System.out.println(42/i);
(6)             } catch (java.lang.ArithmeticException e) {
(7)                     System.out.println("an arithmetic exception was thrown -- aborting
program");
(8)                     System.out.println("Exception's message "+e.getMessage() );
(9)                     System.out.println("Stack trace: ");
(10)                    e.printStackTrace();
(11)
(12)            }
(13)            catch (Exception e) {
(14)                    System.out.println("an exception was thrown -- aborting program");
(15)            } finally {
(16)                    System.out.println("finished try block");
(17)            } //finally
(18)    } //main()
(19)} //ExceptionhandlingTest2
```

Beispiel 14: Ausnahmebehandlung ExceptionHandlingTest2.java

Das Programm liefert die Ausgabe:

```
$java ExceptionHandlingTest1
21
42
```

```
an arithmetic exception was thrown
finished try block
```

Der optional angebbare `finally`-Block wird immer ausgeführt, unabhängig davon ob der von `try` umschlossene Anweisungsteil erfolgreich (d.h. fehlerfrei) oder durch Exception (oder auch sonstige Sprungmechanismen wie `break`) verlassen wurde. Er bietet die Gelegenheit nach erfolgter Ausnahmebehandlung „Aufräumarbeiten“, wie das Schließen möglicherweise noch geöffneter Dateien, durchzuführen.

Nützliche Zusatzinformationen über die Art des aufgetretenen Ausnahmeereignisses können durch die Methoden `getMessage()` und `printStackTrace()` abgefragt werden. Beide Methoden sind von `java.lang.Throwable`, der Superklasse von `java.lang.Exception`, ererbt und stehen daher auf allen Ausnahmeobjekten zur Verfügung.

Ausnahmen die als Subklassen von **`RuntimeException`** realisiert sind müssen nicht explizit deklariert oder aufgefangen werden, da sie von Instruktionen der virtuellen Maschine erzeugt werden. Dies stellt einen Widerspruch zur erhobenen Forderung dar, daß alle Ausnahmen, die Subklassen von `Throwable` sind, explizit zu deklarieren und aufzufangen sind!
Eine korrekte Deklaration wäre jedoch unter praktischen Gesichtspunkten nicht praktikabel, da beispielweise ein Fehler des Typs `InternalError` potentiell durch jede Methode erzeugt werden kann.
Abgesehen davon verhalten sich jedoch Laufzeit-Ausnahmen jedoch wie „normale“ Exceptions. Begründet durch Abwesenheit einer expliziten Deklaration sieht man auch -- außer in raren und begründeten Ausnahmefällen -- vom Auffangen und Behandeln durch den Programmierer ab.

Erzeugen eigener Ausnahmeereignisse
Neben durch das Laufzeitsystem generierten Ausnahmeereignissen können auch innerhalb des Programmcodes gezielt Exceptions durch den Anwender generiert werden. Hierfür existiert das `throw`-Statement.

```
(1)public class OwnException1 {
(2)     public static void main(String[] args) {
(3)             try {
(4)                     System.out.println("throwing an arithmetic exception...");
(5)                     throw new ArithmeticException();
(6)                     //never gets here
(7)             } //try
(8)             catch (Exception e) {
(9)                     System.out.println(e.toString() + " exception caught");
(10)            } //catch
(11)    } //main()
(12)} //class OwnException1
```

Beispiel 15: Anwenderdefiniert ausgelöste Ausnahme OwnException1.java

Das Codebeispiel zeigt das manuelle Erzeugen einer `ArithmeticException`.
Anmerkung: Nach `throw` angegebene Anweisungen können niemals erreicht werden, und führen bereits zum Übersetzungszeitpunkt zu einer entsprechenden Fehlermeldung.

Das bestehende vorgegebene System der Exceptions kann durch den Programmierer jederzeit um eigendefinierte Ausnahmen erweitert werden. Voraussetzung hierfür ist die Definition einer neuen Ausnahmeklasse; dies geschieht (im einfachsten Falle) durch Erben von `java.lang.Exception`.
Das nachfolgende Beispiel illustriert dies:

```
(1)public class OwnException2 {
(2)     public static void main(String[] args) {
(3)             try {
(4)                     System.out.println("throwing myException...");
(5)                     throw new myException();
(6)             } catch (Exception e) {
(7)                     System.out.println(e.toString() + " exception caught");
(8)             } //catch
(9)     } //main()
(10)} //class OwnException2
(11)
(12)class myException extends Exception {
(13)} //class myException
```

Beispiel 16: Eigendefinierte Ausnahme OwnException2.java

Für alle Ausnahmen die nicht lokal behandelt werden, kann durch Angabe der `throws`-Liste die Ausnahmenbehandlung an den Aufrufer weitergereicht werden.

```
(1)public class OwnException3 {
(2)     public static void main(String[] args) {
(3)             try {
(4)                     exceptionProne();
(5)             } catch (myException myE) {
(6)                     System.out.println("Exception of type myException caught!");
(7)             } //catch
(8)     } //main()
(9)
(10)    public static void exceptionProne() throws myException {
(11)            throw new myException();
(12)    } //exceptionProne()
(13)} //class OwnException3
(14)
(15)class myException extends Exception {
(16)} //class myException
```

Beispiel 17: Propagierung der Ausnahmebehandlung OwnException3.java

Im Beispiel wird die eigendefinierte Ausnahme `myException` nicht innerhalb der Methode `exceptionProne` behandelt, sondern an den Aufrufer -- in diesem Beispiel die `main`-Methode -- zur Behandlung weitergereicht. Der übersetzter prüft hier das Vorhandensein eines `try-catch`-Blockes innerhalb von `main` ab.
Das folgende Beispiel zeigt eine Erweiterung des vorhergehenden, dergestalt, daß auch die `main`-Methode mit einem `throws` versehen ist. In diesem Fall wird das Ausnahmeereignis an das Laufzeitsystem weitergereicht; welches in der Konsequenz die Ausführung terminiert.

Generell gilt in Java die *catch or throw*-Regel, die besagt, daß ein Ausnahmeereignis entweder aufgefangen und behandelt (*catch*) oder weitergereicht (*throw*) werden muß.

```
(1)public class OwnException4 {
(2)     public static void main(String[] args) throws myException       {
(3)             throw new myException();
(4)     } //main()
(5)} //class OwnException4
(6)
(7)class myException extends Exception {
(8)} //class myException
```

Beispiel 18: Propagierung eines Ausnahmeereignisses an das Laufzeitsystem OwnException4.java

Auch die Kombination der beiden vorgestellten Ansätze ist möglich ...
Im abschließenden Codebeispiel wird neben der lokalen Ausnahmebehandlung (`catch`-Block innerhalb `exceptionProne()` auch eine Behandlung innerhalb der aufrufenden Methode (`main`) durchgeführt. Hierzu wird das aufgefangene Ausnahmeereignis innerhalb der Behandlungsroutine erneut mittels `throw` ausgelöst.

```
(1)public class OwnException5 {
(2)     public static void main(String[] args) {
(3)             try {
(4)                     exceptionProne();
(5)             } catch (myException e) {
(6)                     System.out.println("exception myException catched within main method");
(7)             } //catch
(8)     } //main()
(9)
(10)    public static void exceptionProne() throws myException {
(11)            try {
(12)                    throw new myException();
(13)            } catch (myException e) {
(14)                    System.out.println("exception myException catched within method
exceptionProne");
(15)                    System.out.println("re-throwing...");
(16)                    throw e;
(17)            } //catch
(18)    } //exceptionProne()
(19)} //class OwnException5
(20)
(21)class myException extends Exception {
(22)} //class myException
```

Beispiel 19: Behandlung und Weiterreichung einer Ausnahme OwnException5.java

▲ **Zwangsbedingungen**

Zur Steigerung der Qualität des entstehenden Systems gestattet Java die Definition von *Zwangsbedingungen* (*Assertion*), deren Einhaltung während der Ausführung geprüft werden kann. Ist eine solche Bedingung nicht erfüllt, so erfolgt ein Programmabbruch.

Im Unterschied zur Möglichkeit durch Selektionsausdrücke die Rückgabewerte von Methodenaufrufen auszuwerten oder durch Ausnahmebehandlung gezielt und benutzerdefiniert auf Fehlersituationen zu reagieren bieten die Zwangsbedingungen sowohl eine gegenüber den beiden genannten Ansätzen signifikant kompaktifizierte Syntax an, die gleichzeitig weniger Freiheitsgrade in der Behandlung der Fehlersituation bietet.
Zusätzlich kann die Prüfung von Zwangsbedingungen zur Laufzeit statisch durch einen Schalter der Ausführungsumgebung gesteuert werden. Auf dieser Basis eignen sie sich gut für die Formulierung verschiedenster Konsistenzprüfungen, die später im Produktivbetrieb zur Steigerung der Ausführungsgeschwindigkeit deaktiviert werden können.

Die allgemeine Syntax einer Zwangsbedingung lautet:

```
assert Boole'scher-Ausdruck (: Ausdruck)opt
```

Generell werden Zwangsbedingungen durch das Schlüsselwort `assert` eingeleitet. Darauf folgt ein Boole'scher Ausdruck, der erfüllt sein muß. Liefert die Auswertung dieses Ausdrucks den Wahrheitswert `false`, so erfolgt der Programmabbruch. Ist zusätzlich, nach dem separierenden Doppelpunkt, ein Ausdruck angegeben, so wird dieser zur Konstruktion eines `AssertionError`-Objekts herangezogen.

Aus den zugelassenen Parametertypen eines solchen Objekts ergeben sich die möglichen angebbaren Ausdruckstypen als: `boolean`, `char`, `double`, `float`, `int`, `long` und `Object`.

Das nachfolgende Beispiel zeigt den Einsatz einer einfachen Zwangsbedingung, deren Fehlschlag ohne anwenderdefinierte Konstruktion `AssertionError`-Objekts behandelt wird:

```
(1)public class AssTest1 {
(2)     public static void main(String[] args) {
(3)             Object o = null;
(4)             //...
(5)             assert o != null;
(6)     } //main()
(7)} //class AssTest1
```

Beispiel 20: Einfache Zwangsbedingung AssTest1.java

Das Beispiel deklariert eine Variable o als Ausprägung der Klasse `Object` und initialisiert diese mit `null`. Im späteren Verlauf der Ausführung soll geprüft werden, ob zwischenzeitlich eine Initialisierung erfolgt ist. Ist dies nicht der Fall, so ist eine Programmfortsetzung nicht sinnvoll. Diese Prüfung, verbunden mit der impliziten Forderung den Ablauf bei ihrer Nichterfüllung zu terminieren, ist als Zwangsbedingung realisiert.

Zur Verwendung der Zwangsbedingungen ist die Übersetzung mindestens konform zur Sprachversion 1.4 notwendig. Aktiviert wird diese Quellcodeinterpretation durch Übergabe des Wertes „1.4“, oder höher, als Parameter des Übersetzerschalters `-source`. Wird ein Wert kleiner als `1.4` gewählt, so wird das Schlüsselwort `assert` nicht als solches interpretiert, sondern kann als Identifier zur Benennung von Klassen, Attributen oder Methoden benutzt werden.
Im selben Sinne ist es ebenso zwingend erforderlich die Prüfung von Zwangsbedingung innerhalb der Laufzeitumgebung durch den Schalter `-enableassertions` zu aktivieren. Andernfalls werden alle `assert`-Anweisungen ungeprüft ignoriert. Aus dieser Forderung ergibt sich, daß mindestens Version 1.4 der Laufzeitumgebung benötigt wird um die Interpretation von Zwangsbedingungen zu aktivieren.

Die Ausführung des Beispiels liefert daher, bei aktiviertem Schalter die Ausgabe:

```
Exception in thread "main" java.lang.AssertionError
        at AssTest1.main(AssTest1.java:5)
```

Zur Einschränkung der Zwangsbedingungsauswertung gestattet die Laufzeitumgebung die Spezifikation derjenigen Klassen, für die diese Bedingungen ausgewertet werden sollen. Hierzu werden nach dem Schlüsselwort `enableassertions`, abgetrennt durch einen Doppelpunkt, die Namen der zu berücksichtigenden Klassen angegeben werden. Für das obenstehende Beispiel sind daher die Aufrufe `java -enableassertions AssTest1` und `java -enableassertions:AssTest1 AssTest1` äquivalent.

Ebenfalls durch Kommandozeilenschalter kann die (De-)Aktivierung der Prüfung der in den Standard-API-Klassen formulierten Zwangsbedingungen gesteuert werden. Hierfür stehen die Schalter `-enablesystemassertions` bzw. `-disablesystemassertions`. Die Möglichkeiten der klassengenauen Steuerung stehen jedoch in diesem Falle nicht zur Verfügung.

Die Möglichkeit der Übergabe von Parametern an das automatisiert durch das Laufzeitsystem erzeugte `AssertionError`-Objekt bietet sich insbesondere zur Dokumentation der Abbruchursache an. So zeigt das nachfolgende Beispiel prüft durch Aufruf der Methode `exists` der Klasse `File` ob die Datei *test* im aktuellen Dateisystemkatalog angelegt ist. Ist dies nicht erfüllt, so wird eine festgelegte Zeichenkette dem Konstruktor von `AssertionError` übergeben. Die Zeichenkette kann, als Ausprägung der Klasse `String` zur Konstruktion eines Fehlerobjektes verwendet werden, da sie gemäß der Typrestriktion eine gültige Instanz von `Object` repräsentiert. Gleichzeitig kann die Methode `exists` als Ausdruck nach dem `assert`-Schlüsselwort angegeben werden, da die Ausführung der Methode einen Rückgabewert vom Typ `boolean` liefert und damit der Gesamtausdruck als

Wahrheitswert ausgewertet werden kann.

```
(1)import java.io.File;
(2)
(3)public class AssTest2 {
(4)     public static void main(String[] args) {
(5)             //...
(6)             assert (new File("test")).exists() : "File 'test' does not exist";
(7)     } //main()
(8)} //class AssTest2
```

Beispiel 21: Zwangsbedingung mit anwenderdefinierter Fehlerobjekt AssTest2.java

Zwar sollte nach Nichterfüllung einer Zwangsbedingung die Applikationsausführung beendet werden, wie es auch im Standardfalle durch das Laufzeitsystem geschieht. Jedoch kann dieses Verhalten mit Mitteln des Ausnahmebehandlung unterbunden werden. Hierzu muß eine Zwangsbedingungsauswertung in einen `try`-Block einbettet werden. Findet sich im zugeordneten `catch`-Bereich eine Klausel für den `AssertionError` so wird diese ausgeführt und anschließend die Programmverarbeitung normal fortgesetzt. Das nachfolgende Beispiel zeigt diese Anwendung.

```
(1)import java.io.File;
(2)
(3)public class AssTest3 {
(4)     public static void main(String[] args) {
(5)             //...
(6)             try {
(7)                     assert false : "this assertion fails definitely";
(8)             } catch (AssertionError ae) {
(9)                     System.out.println("caught assertion error");
(10)            } //catch
(11)            System.out.println("continuing after error ");
(12)    } //main()
(13)} //class AssTest3
```

Beispiel 22: Abfangen einer fehlschlagenden Zwangsbedingung AssTest3.java

Das im Beispiel gezeigte Verhalten ist zwar syntaktisch korrekt und wird auch im angestrebten Sinne ausgeführt, sollte jedoch nur mit Bedacht angewandt werden, da es die Semantik einer Zwangsbedingung aufweicht. Insgesamt empfiehlt sich die Verwendung von Zwangsbedingungen lediglich für eine engumrissene Klasse von Fehlern, wie Nachbedingungen von Methoden, die einen internen Systemzustand dokumentieren, der durch keine Modifikation in einen konsistenten Zustand überführt werden könnte, er eine Ausführungsfortsetzung sinnvoll erscheinen läßt. Insbesondere solche, die durch Anwenderinteraktion begründet sind (wie fehlerhafte Parameterübergabe, etc.) sollten durch Ausnahmen behandelt werden.

2.4 Von komplexen zu objektorientierten Datenstrukturen

2.4.1 Arrays

Wie fast alle prozeduralen Programmiersprachen unterstützt auch Java die Zusammenfassung beliebiger gleichartiger Elemente zu geordneten, nicht zwingend dupplikatfreien, Mengen; die sog. *Feldern, Arrays* oder *Vektoren*.

Arrays in Java sind nicht Bestandteil des built-in Typsystems, sondern bereits als Klasse innerhalb der Java API realisiert.
Wie in anderen Programmiersprachen üblich stellen Javaarrays einen zusammenhängen kontinuierlichen Speicherbereich dar.

Bereits innerhalb der Standardsignatur der `main`-Methode wird ein Array verwendet:
`public static void main(`**`String[] args`**`)`.

Einige Charakteristika:

- Statische Größenfestlegung.
  Arraygrenzen können nach Definition nicht mehr verändert werden.
  Vorgriff: Dies ist mit der API Klasse *Vector* möglich.
- Arrays sind (*first class*) Objekte.
  => Sie müssen durch den Programmierer explizit mit new erzeugt werden.
- Die Inhaltselemente sind immer beginnend mit 0 nummeriert.
- Zugriffe über die Arraygrenzen hinaus werden durch die `ArrayIndexOutOfBoundsException` abgefangen.
- Definition anonymer (d.h. namenloser) Arrays möglich.

- Definition „mehrdimensionaler“ Arrays möglich.
  ...dazu später mehr.
- Array können bereits während der Definition initialisiert werden.
  Hierdurch können auch dynamisch allokierte Arrays realisiert werden.
- Zeichenketten (Strings) sind *nicht* als *Array of Character*, sondern als eigene API-Klasse, realisiert.
- Der Zugriff auf einzelne Arrayelemente erfolgt über Index (=berechenbare Adresse).
  `a[i]` greift auf das i+1-te Element des Arrays a zu.
- Arraykomponenten werden bei der Erzeugung mit `new` initialisiert.
  Im Falle primitiver Datentypen wird als Vorgabewert `0` für alle numerischen Datentypen, `false` für `boolean` und Unicode `\u0000` für `char` gesetzt. Objektwertige Typen werden durch ihren Konstruktor initialisiert.

Syntax:

```
ArrayCreationExpression:
   new PrimitiveType DimExprs Dims_optional
   new TypeName DimExprs Dims_optional
   new PrimitiveType Dims_optional ArrayInitializer
   new TypeName Dims ArrayInitializer

DimExprs:
   DimExpr
   DimExprs DimExpr

DimExpr:
   [ Expression ]

Dims:
   [ ]
   Dims [ ]
```

```
(1)public class Array1 {
(2)     public static void main(String[] args) {
(3)             int                     firstArray[]    = new int[10];
(4)             int[]           secondArray     = new int[10];
(5)             double[]                thirdArray              = {3.14, 2+5, 42.0};
(6)
(7)             System.out.println("length of firstArray="+ firstArray.length );
(8)             System.out.println("length of secondArray="+ secondArray.length );
(9)             System.out.println("length or thirdArray="+ thirdArray.length );
(10)            for (int i=0; i<thirdArray.length; i++)
(11)                    System.out.println("thirdArray["+i+"]="+thirdArray[i]);
(12)
(13)            System.out.println("lenght of anonymous array="+ (new byte[] {1,2,3}).length);
(14)    } //main()
(15)} //class Array1
```

Beispiel 23: Arrayerzeugung und Initialisierung Array1.java

Der Code aus Beispiel 16 definiert zunächst zwei `int`-Array der Größe 10. Die beiden Syntaxvarianten sind (wie aus C/C++ bekannt) äquivalent.
`thirdArray` wird bereits während der Definition mit double-Werten initialisiert. Die Größe muß nicht explizit angegeben werden, die errechnet sich automatisch aus der Anzahl der angegeben Ausdrücke.
Die letzte Definition eines Arrays im Beispiel erfolgt anonym. Der so erzeugte Array steht nach Verlassen des `System.out.println`-Ausdrucks nicht mehr zur Verfügung.

Arrays deren Elemente eigendefinierte Typen sind werden entsprechend mit *`eigenerTyp`*`[] `*`arrayName`*` ...` definiert.

Mehrdimensionale Arrays ...
werden nicht explizit unterstützt, sondern als Arrays von Arrays behandelt.

```
(1)public class Array2 {
(2)     public static void main(String[] args) {
(3)             int[][] multiDimensional = new int[2][3];
(4)
(5)             int dimR = multiDimensional.length;
(6)             int dimC = multiDimensional[0].length;
(7)
(8)             System.out.println("Lenght of multiDimensional="+ dimR);
(9)             System.out.println("Lenght of multiDimensional[0]="+dimC );
(10)
(11)            for(int i=0; i<dimR; i++)
(12)                    for(int j=0; j<dimC; j++)
(13)                            multiDimensional[i][j] = i*dimC + j+1;
(14)
(15)            System.out.println("Content of multiDimensionl:");
(16)            for(int i=0; i<dimR; i++) {
(17)                    for(int j=0; j<dimC; j++) {
(18)                            System.out.print(multiDimensional[i][j]+ " ");
(19)                    } //for
```

```
(20)                    System.out.println();
(21)            } //for
(22)    } //main()
(23)} //class Array2
```

Beispiel 24: Mehrdimensionale Arrays Array2.java

Bildschirmausgabe:

```
$java Array2
Lenght of multiDimensional=2
Lenght of multiDimensional[0]=3
Content of multiDimensionl:
1 2 3
4 5 6
```

Die Angabe mehrer Dimensionsgrößen bei der Definition stellt eine Kurzform für die verschachtelte Variante dar. So ist das nachfolgende Codefragment äquivalent zum vorhergehenden Beispiel:

```
(1)public class Array3 {
(2)     public static void main(String[] args) {
(3)             int[][] multiDimensional = new int[2][];
(4)             for (int i=0; i<multiDimensional.length; i++)
(5)                     multiDimensional[i] = new int[3];
(6)
(7)             int dimR = multiDimensional.length;
(8)             int dimC = multiDimensional[0].length;
(9)
(10)            System.out.println("Lenght of multiDimensional="+ dimR);
(11)            System.out.println("Lenght of multiDimensional[0]="+dimC );
(12)
(13)            for(int i=0; i<dimR; i++)
(14)                    for(int j=0; j<dimC; j++)
(15)                            multiDimensional[i][j] = i*dimC + j+1;
(16)
(17)            System.out.println("Content of multiDimensionl:");
(18)            for(int i=0; i<dimR; i++) {
(19)                    for(int j=0; j<dimC; j++) {
(20)                            System.out.print(multiDimensional[i][j]+ " ");
(21)                    } //for
(22)                    System.out.println();
(23)            } //for
(24)    } //main()
(25)} //class Array3
```

Beispiel 25: verschachtelte mehrdimensionale Arraydefinition Array3.java

Die Definition
`int[][] multiDimensional = { {1,2,3}, {100,200,300} };`
Definiert einen Array der zwei Array, jeweils der Länge 3, enthält, und initialisiert diese mit den angegebenen Werten.
Unterscheiden sich die beiden implizit spezifizierten Arrays in der Größe, so definiert die Anzahl des ersten angegebenen Arrays die Elementzahl (Zeilenlänge) für alle folgenden (Zeileneinträge).

Duplizieren von Arrays:
Für die häufig benötigte Anwendung den vollständigen Inhalt eines Arrays in einen zweiten Array gleicher Dimension (en) zu kopieren exisitert die `clone`-Methode.

```
(1)public class Array4 {
(2)     public static void main(String[] args) {
(3)             int[] ia = {1,2,3,4,5};
(4)             int[] ib = new int[ia.length];
(5)
(6)             System.out.println("ia==ib = "+  (ia==ib));
(7)
(8)             System.out.println("content of ia:");
(9)             for(int i=0; i<ia.length; i++)
(10)                    System.out.print(ia[i]+" ");
(11)            System.out.println("");
(12)
(13)            System.out.println("content of ib:");
(14)            for(int i=0; i<ib.length; i++)
(15)                    System.out.print(ib[i]+" ");
(16)            System.out.println("");
(17)
(18)
(19)            ib = (int[]) ia.clone();
(20)            System.out.println("content of ib after cloning:");
(21)            for(int i=0; i<ib.length; i++)
```

```
(22)                    System.out.print(ib[i]+" ");
(23)            System.out.println("");
(24)
(25)            System.out.println("ia==ib = "+  (ia==ib));
(26)
(27)            System.out.println("Setting ia=ib...");
(28)            ia=ib;
(29)            System.out.println("ia==ib = "+  (ia==ib));
(30)    } //main()
(31)} //class Array4
```

Beispiel 26: Duplizierung eines Arrayinhaltes mit der clone-Methode Array4.java

Bildschirmausgabe:

```
$java Array4
ia==ib = false
content of ia:
1 2 3 4 5
content of ib:
0 0 0 0 0
content of ib after cloning:
1 2 3 4 5
ia==ib = false
```

Das Beispiel definiert zunächst den Array `ia` per expliziter Initialisierung. Ein zweiter Array, `ib` wird mit derselben Länge wie `ia` definiert.
Wie zu erwarten sind die Arrayobjekte verschieden, d.h. sie belegen unterschiedliche Speicherplätze.
Die `clone`-Methode dupliziert den Inhalt des Arrays `ia` und weist in `ib` zu.
In den letzten Zeilen wird der Variable `ia` der Wert von `ib` zugewiesen. Mithin der Array `ia` durch `ib` überschrieben. Das ursprüngliche `ia` kann damit nicht mehr durch den Programmierer referenziert werden, und wird für den Garbage Collector als freizugeben markiert.

Die **ArrayStoreException** wird ausgeworfen, sobald ein Typkonflikt beim Einfügen eines Arrayelementes zur Laufzeit auftritt; statisch erkennbare Typkonflikte werden bereits durch den übersetzer gemeldet.

```
(1)public class Array5 {
(2)     public static void main(String[] args) {
(3)             Test2[] ta = new Test2[5];
(4)             Test1[] tb = ta;
(5)
(6)             try {
(7)                     tb[0] = new Test1();
(8)             } //try
(9)             catch (ArrayStoreException e) {
(10)                    System.out.println(e);
(11)            } //catch
(12)    } //main()
(13)} //class Array5
(14)
(15)class Test1 {
(16)} //class Test1
(17)
(18)class Test2 extends Test1 {
(19)} //class Test2
```

Beispiel 27: Typkonflikt beim Einfügen, der zur Auslösung einer ArrayStoreException führt Array5.java

Obwohl der Typ der Arraykomponenten von `tb` als `test1` deklariert war, führt die Zuweisung innerhalb des `try`-Blockes zu einer `ArrayStoreException`.
Ursache: Durch die Initialisierung mit Elementen des Typs `Test2`, der eine Spezialisierung von `Test1` bildet, wird auch der erwartete Inhaltstyp von `Test1` verändert (konkret: spezialisiert).

2.4.2 Klassen

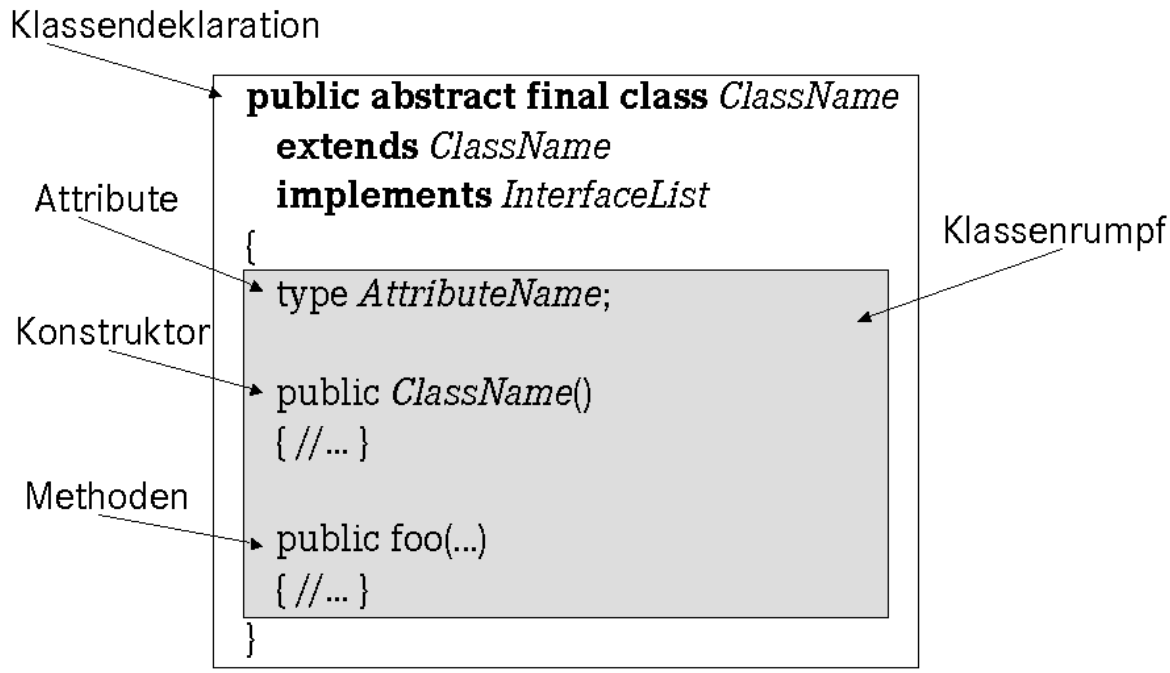
Klassen und Objekte stellen das namensgebende Hauptabstraktionsmittel in der objektorienterten Programmierung dar.
Prinzipiell lassen sich aus Sicht der OO-Programmierung drei Arten von Sprachen unterscheiden:

- Klassenlose Sprachen, wie C, Pascal, etc.
- Klassenbasierte hybride Sprachen, die Objekten und Vererbung unterstützen -- jedoch ihre Verwendung nicht auf allen Ebenen durchgängig erzwingen, wie C++ und Java
- (rein) objektorientierte Sprachen, wie SmallTalk, Eiffel, etc.

Gemäß dieser Einordnung ist Java als hybride objektorientierte Sprache anzusehen. Dies rührt hauptsächlich von der Umsetzung des build-in Typsystems als einfache Werte -- anstatt first class objects -- her.
In der Praxis zieht dies jedoch zumeist keine allzugrossen Einschränkungen nach sich.

Die Verwendung von Klassen ist in Java, im scharfen Gegensatz zu C++, zwingend. So ist es nicht möglich ein Programm ohne zumindest eine (Haupt-)Klasse zu schreiben.

Jede Klasse bildet einen abgeschlossenen Sichtbarkeitsbereich (auch: Scope) für die darin definierten Attribute und Methoden.

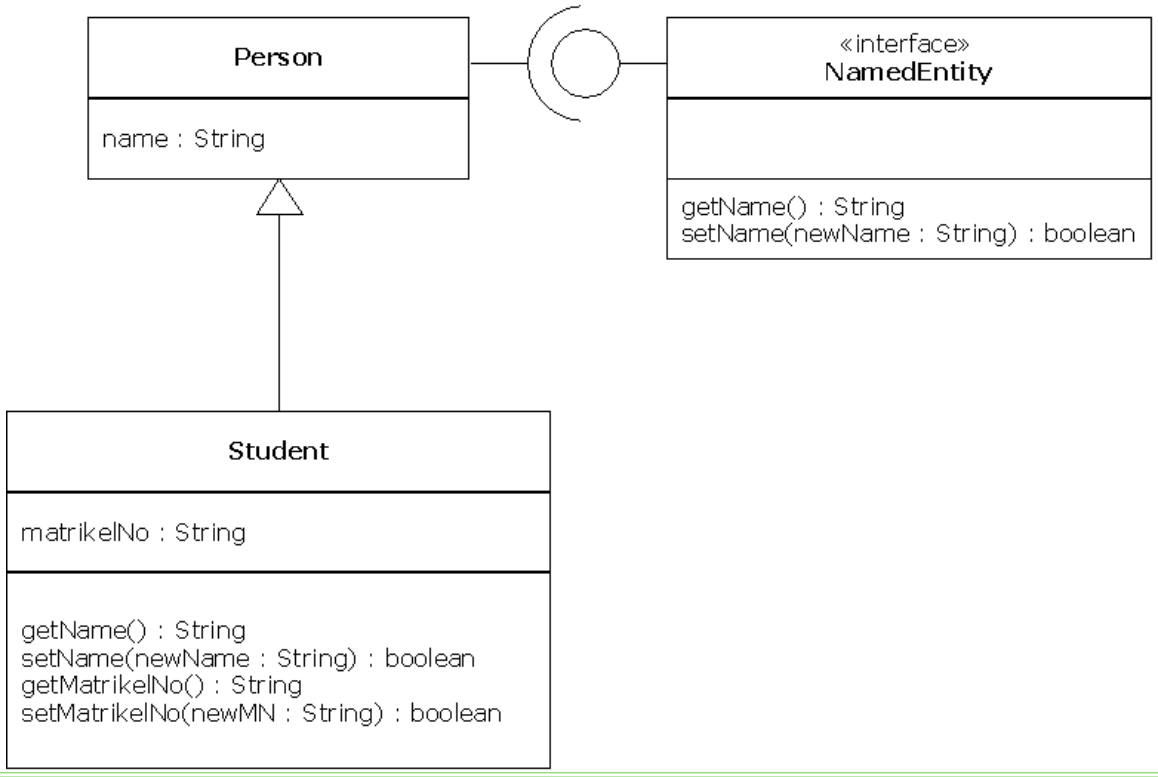


Eine Java-Klasse besteht aus den in der Abbildung dargestellten Teilen:
Klassendeklaration: Sie benennt die Klasse eindeutig, und legt die allgemeinen Charakteristika fest. Das Aussehen der Deklaration kann wie folgt beschrieben werden:

Syntaxelement	Semantik
`public`	Die Klasse ist allgemein sichtbar. Vorgabegemäß kann eine Klasse nur von Klassen desselben Packages genutzt werden. Durch das Schlüsselwort `public` wir sie auch außerhalb des umgebenden Packages sicht- und zugreifbar. Die namensgebende Klasse einer Java-Quellcode-Datei, die auch die `main`-Methode enthalten kann, muß zwingend als public erklärt sein.
`abstract`	es können keine Objekte dieser Klasse erzeugt werden. Abstrakte Klassen dienen zur Strukturierung des Entwurfs, um gemeinsame Merkmale verschiedener Klassen zentralisiert ausdrücken zu können.
`final`	von dieser Klasse kann nicht geerbt werden. Die Entscheidung eine Klasse als `final` zu deklarieren ist designgetrieben. In der Verwendung der Klasse ergeben sich keine Unterschiede. Jedoch können so Vererbungsäste als abgeschlossen gekennzeichnet werden, um auszudrücken, daß an dieser Stelle keine Weiterentwicklung erfolgen soll. Das Schlüsselwort verhindert einen entsprechenden Versuch durch einen Fehler zum Übersetzungszeitpunkt. Beispiele aus der Java-API: java.io.FileDescriptor, die Wrappertypen: Boolean, Integer, etc.
`class` *`ClassName`*	Name der Klasse
`extends` *`ClassName`*	Die Klasse erbt von der Klasse *ClassName* In Java ist nur einfache Vererbung zugelassen, daher kann an dieser Stelle auch maximal eine Superklasse spezifiziert werden.
`implements` *`InterfaceList`*	Die Klasse implementiert die in der *InterfaceList* aufgeführten Schnittstellen. Die Schnittstellennamen werden durch Kommata voneinander abgetrenn.

`{`
`ClassBody`
`}`
	Ausprogrammierter Klassenrumpf.

Ein umfangreicheres Beispiel:




```
(1)public class Student extends Person implements NamedEntity {
(2)     private String matrikelNo;
(3)
(4)     public Student(String matrikelNo) {
(5)             this.matrikelNo = matrikelNo;
(6)     } //constructor
(7)
(8)     public String getMatrikelNo() {
(9)             return matrikelNo;
(10)    } //getMatrikelNo;
(11)
(12)    public boolean setMatrikelNo(String matrikelNo) {
(13)            if (matrikelNo.compareTo("")==0) {
(14)                    this.matrikelNo = matrikelNo;
(15)                    return true;
(16)            } else
(17)                    return false;
(18)    } //setMatrikelNo()
(19)
(20)    public String getName() {
(21)            return name;
(22)    } //getName()
(23)
(24)    public boolean setName(String newName) {
(25)            if (newName.compareTo("")!=0) {
(26)                    name = newName;
(27)                    return true;
(28)            } else
(29)                    return false;
(30)    } //setName()
(31)
(32)    public String toString() {
(33)            return ("Name: "+this.getName()+"\nMatrikelnummer: "+this.getMatrikelNo() );
(34)    } //toString()
(35)
(36)    public static void main(String[] args) {
(37)            Student mario = new Student("0793022");
(38)            System.out.println( mario.getMatrikelNo() );
(39)            System.out.println("mario.toString() returns:\n"+mario.toString() );
(40)            mario.setName("Mario Jeckle");
(41)            System.out.println("mario.toString() returns:\n"+mario.toString() );
(42)    } //main()
(43)} //class Student
(44)
(45)class Person {
(46)    String  name;
(47)} //class Person
(48)
(49)interface NamedEntity {
```

```
(50)    String getName();
(51)    boolean setName(String newName);
(52)} //interface NamedEntity
```

Beispiel 28: Beispiel einer ausprogrammierten Klasse Student.java

Bildschirmausgabe:

```
0793022
mario.toString() returns:
Name: null
Matrikelnummer: 0793022
mario.toString() returns:
Name: Mario Jeckle
Matrikelnummer: 0793022
```

Innere Klassen

Seit Java v1.1 besteht auch die Möglichkeit Klassen innerhalb von Klassen zu definieren.
Merkmale:

- Objekte der inneren Klasse können auf private Daten des erstellenden Objektes zugreifen.
- Innere Klassen unterliegen dem Sichtbarkeitsbereich der umgebenden Klasse, d.h. sie sind vor Zugriffen anderer Klassen geschützt.
- Stark ereignisbezogene Anwendungen wie GUI-Bibliotheken nutzen diese Definitionsvariante sehr stark aus.
- Für inner classes wird eine eigenständige class-Datei mit der Namenskonvention *UmgebendeKlasse*$*innereKlasse* erzeugt.
  Innere Klassen bilden daher kein originäres Sprachmerkmal, sondern werden durch den Compiler in „normale“ Klassen umgesetzt; für die virtuelle Maschine ist diese Mimik (nahezu) transparent.

(Naive) Erweiterung des vorhergehenden Beispiels: Die Matrikelnummer ist als eigenständige Klasse innerhalb von Student2 realisiert:

```
(1)public class Student2 extends Person implements NamedEntity {
(2)     public Student2(String newMatrikelNo) {
(3)             matrikelNo = new MatrikelNo(newMatrikelNo);
(4)     } //constructor
(5)
(6)     class MatrikelNo {
(7)             private String  matrikelNo;
(8)
(9)             public MatrikelNo(String newMatrikelNo) {
(10)                    matrikelNo = newMatrikelNo;
(11)                    if (this.checkMatrikelNo() != true)
(12)                            System.out.println("illegal MatrikelNo");
(13)            } //constructor
(14)            public boolean checkMatrikelNo() {
(15)                    return( this.matrikelNo.length() == 7 ? true : false);
(16)            } //end checkMatrikelNo()
(17)
(18)            public String getMatrikelNo() {
(19)                    return matrikelNo;
(20)            } //getMatrikelNo()
(21)
(22)            public boolean setMatrikelNo(String newMatrikelNo) {
(23)                    matrikelNo = newMatrikelNo;
(24)                    if (this.checkMatrikelNo() == true)
(25)                            return true;
(26)                    else
(27)                            return false;
(28)            } //setMatrikelNo()
(29)    } //MatrikelNo;
(30)
(31)
(32)    private MatrikelNo matrikelNo;
(33)
(34)    public String getMatrikelNo() {
(35)            return matrikelNo.getMatrikelNo();
(36)    } //getMatrikelNo;
(37)
(38)    public boolean setMatrikelNo(String matrikelNo) {
(39)            return true;
(40)    } //setMatrikelNo()
(41)
(42)    public String getName() {
(43)            return name;
(44)    } //getName()
(45)
(46)    public boolean setName(String newName) {
```

```
(47)            if (newName.compareTo("")!=0) {
(48)                    name = newName;
(49)                    return true;
(50)            } else
(51)                    return false;
(52)    } //setName()
(53)
(54)    public String toString() {
(55)            return ("Name: "+this.getName()+"\nMatrikelnummer: "+this.getMatrikelNo() );
(56)    } //toString()
(57)
(58)    public static void main(String[] args) {
(59)            Student2 mario = new Student2("0793022");
(60)            System.out.println( mario.getMatrikelNo() );
(61)
(62)            Student2 hans = new Student2("1234567X");
(63)    } //main()
(64)} //class Student2
(65)
(66)class Person {
(67)    String  name;
(68)} //class Person
(69)
(70)interface NamedEntity {
(71)    String getName();
(72)    boolean setName(String newName);
(73)} //interface NamedEntity
```

Beispiel 29: Realisierung der Matrikelnummer als innere Klasse Student2.java

Anonyme Klassen:
Als weitere Besonderheit besteht die Möglichkeit die eingebettete Klasse anonym zu definieren. (siehe Java Language Specification)
Syntaktisch folgt die gesamte Klassendefinition auf eine mit new eingeleitete Objekterzeugung.
Unbenannte Klassen verfügen über keinen Konstruktor, da dieser konsequenterweise auch namenlos sein müßte.

Syntax:

```
new BaseClass (Parameters) {
   //inner class methods and data
};
```

Hinweis: Man beachte das (zwingende) Semikolon am Ende des Ausdrucks!

```
(1)public class Anonymous {
(2)     private TestClass test() {
(3)             return new TestClass() {
(4)                     public void hello() {
(5)                             System.out.println("overridden test!");
(6)                     } //hello()
(7)             }; //class TestClass
(8)     } //test()
(9)
(10)    public static void main(String[] args) {
(11)            TestClass myRV;
(12)            myRV = (new Anonymous()).test();
(13)            myRV.hello();
(14)
(15)            System.out.println( myRV instanceof TestClass);
(16)            System.out.println( myRV.getClass().getName() );
(17)    } //end main()
(18)} //class Anonymous
(19)
(20)class TestClass {
(21)    public void hello() {
(22)            System.out.println("original");
(23)    } //hello()
(24)} //class TestClass
```

Beispiel 30: Anonyme innere Klasse Anonymous.java

Bildschirmausgabe:

```
$java Anonymous
overridden test!
true
Anonymous$1
```

In der durch die Methode `test` retournierten Ausprägung der Klasse `TestClass` ist die Methode `hello` überschrieben. Da jede innere anonyme Klasse eine bestehende Klasse aus Ausgangsbasis besitzen muß kann die zurückgegebene Ausprägung einer Speicherzelle dieses Typs zugewiesen werden. Entsprechend lifert der `instanceof`-Operator auch `true` für den Typtest zurück.
Durch die Redefinition der Methode `hello` wird innerhalb von `main` nicht die ursprüngliche, sondern die überschreibende der anonymen Klasse aufgerufen.
Die letzte Programmzeile in `main` bildet einen Vorgriff auf die noch zu behandelnde Reflection API, welche die Gewinnung von Modellinformationen zur Laufzeit erlaubt. Konkret ermittelt die abgebildete Zeile den Namen der Klasse des in `myRV` gespeicherten Objekts. Java setzt für innere Klassen einen Namen aus der umgebenden Klasse und dem Namen der inneren Klasse, abgetrennt durch das Dollarsymbol, zusammen. Anonyme Klassen werden aufsteigend nummeriert. Daher im Beispiel der Surrogatname `Anonymous$1` für die erste anonyme Klasse innerhalb der Klasse `anonymous`.

Im Grunde genommen handelt es sich bei anonymen inneren Klassen de facto um eine besondere Art der Vererbung, welche bereits bekannte Methoden überschreibt. Das überladen existierender Methoden, sowie die Erweiterung der Superklasse um zusätzliche Attribute oder Methoden ist nicht möglich.

Vorwärtsverweis: Ein reales Beispiel für die Nutzung dieses Sprachmechanismus wird in Kapitel 3 vorgestellt.
Im Kontext des in Kapitel 3.2.7 vorgestellten Abstract Windowing Toolkit (AWT) ergeben sich gute reale Anwendungsfälle für anonyme innere Klassen.

Abschlußbemerkung:
Die weiteren Spielarten innerer Klassendefinitionen werden wegen der geringen praktischen Relevanz -- außerhalb des AWT --, und tendenziellen Unübersichtlichkeit des entstehenden Entwurfs nicht diskuiert. Eine gute Referenz findet sich im für Ausbildungszwecke (als HTML) kostenfrei verfügbaren Buch GoTo Java2.

Mit `strictfp` existiert seit Java v1.2 ein neues Schlüsselwort zur Modifikation des Klassenverhaltens. Es deaktiviert die erweiterte Gleitpunktdarstellung nach IEEE 754-1985. In diesem Modus werden `float`-Datentypen statt mit 32 mit 43-Bit, bzw. `double`-Datentypen mit 79 statt 64-Bit behandelt.
Durch die bessere Nutzung der vorhandenen Zielhardware ergibt sich neben Performancegewinnen auch eine erhöhte Berechnungsgenauigkeit. Als Resultat kann derselbe Code auf verschiedenen realen Maschinen, je nach Auslegung der Gleitkommaarithmetik, zu verschiedenen Berechnungsergebnissen führen.
Durch Angabe von `strictfp` wird somit wieder portables Verhalten der Fließkommaoperationen erzwungen.

Das Beispiel (nach Kazuyuki SHUDO) zeigt die Verwendung dieses Schlüsselwortes:

```
(1)class StrictfpTest {
(2)  private static double defaultDmul(double a, double b) {
(3)    return a * b;
(4)  }
(5)
(6)  private static strictfp double strictDmul(double a, double b) {
(7)    return a * b;
(8)  }
(9)
(10)  private static double defaultDdiv(double a, double b) {
(11)    return a / b;
(12)  }
(13)
(14)  private static strictfp double strictDdiv(double a, double b) {
(15)    return a / b;
(16)  }
(17)
(18)  public static void main(String[] args) {
(19)    double a, b, c;
(20)
(21)    /* multiplication */
(22)    a = Double.longBitsToDouble(0x0008008000000000L);
(23)    b = Double.longBitsToDouble(0x3ff0000000000001L);
(24)
(25)    System.out.println(a + " (0x0008008000000000)");
(26)    System.out.println("  * " + b + " (0x3ff0000000000001)");
(27)
(28)    c = defaultDmul(a, b);
(29)    System.out.println("default : " + c +
(30)            " (0x" + Long.toHexString(Double.doubleToLongBits(c)) + ")");
(31)
(32)    c = strictDmul(a, b);
(33)    System.out.println("strictfp: " + c +
(34)            " (0x" + Long.toHexString(Double.doubleToLongBits(c)) + ")");
(35)
(36)    System.out.println();
(37)
(38)    /* division */
(39)    a = Double.longBitsToDouble(0x000fffffffffffffL);
(40)    b = Double.longBitsToDouble(0x3feffffffffffffL);
(41)
(42)    System.out.println(a + " (0x000fffffffffffff)");
(43)    System.out.println("  / " + b + " (0x3feffffffffffff)");
(44)
```

```
(45)    c = defaultDdiv(a, b);
(46)    System.out.println("default : " + c +
(47)            " (0x" + Long.toHexString(Double.doubleToLongBits(c)) + ")");
(48)
(49)    c = strictDdiv(a, b);
(50)    System.out.println("strictfp: " + c +
(51)            " (0x" + Long.toHexString(Double.doubleToLongBits(c)) + ")");
(52)  }
(53)}
```

Beispiel 31: Verwendung des Schlüsselwortes strictfp StrictfpTest.java

Die Unterstützung durch die virtuelle Maschine vorausgesetzt, liefert die Ausführung des Programms folgende Ausgabe:

```
1.112808544714844E-308 (0x0008008000000000) * 1.0000000000000002 (0x3ff0000000000001)
default : 1.1128085447148447E-308 (0x8008000000000)
strictfp: 1.1128085447148447E-308 (0x8008000000001)

2.225073858507201E-308 (0x000fffffffffffff) / 0.9999999999999999 (0x3fefffffffffffff)
default : 2.2250738585072014E-308 (0x10000000000000)
strictfp: 2.225073858507201E-308 (0xfffffffffffff)
```

SUNs Referenzimplementierung der virtuellen Maschine unterstützen ab Version 1.3 auf den Intelplattformen Windows und Linux die korrekte Umsetzung des erweiterten Fließkommaformates.

Objekterzeugung:
Die Erzeung konkreter Ausprägungen (auch: Instanzen oder Objekte) einer Klasse geschieht üblicherweise durch den new-Operator.
Die Syntax lautet: *Type Variable* = new *Type*(*Parameters*)

Wird ein Objekt nicht mehr durch den Programmierer referenziert, so wird es durch den Garbage Collector aus dem Speicher entfernt. Dies kann u. U. auch erst verzögert geschehen, da der Garbage Collector als eigener asynchroner Thread der virtuellen Maschine realisiert ist. (Der Aufruf System.gc() schlägt der virtuellen Maschine vor den Garbage Collector bei Gelegenheit aufzurufen.)

Weiterführende Informationen zum Thema *innere Klassen*.

2.4.3 Attribute

Attribute stellen in der objektorientierten Programmierung den üblichen -- und bei strenger Betrachtung den einzigen -- Weg zur Ablage dynamischer Zustandsinformation eines Objekts dar. (Selbstverständlich können auch Variablen innerhalb von Methoden beliebige Informationen aufnehmen. Jedoch sind diese Inhalte im Allgemeinen nach Verlassen des Gültigkeitsbereichs verloren).

Abbildung 2 stellt die Plazierung der Attribute in UML-Notation dar.

Vereinfachte **Syntax** einer Attributdefinition:

(public, protected, private, static, final, transient, volatile)$_{\text{zero or more}}$ *Identifier* []$_{\text{optional}}$ (= *Initialization*)$_{\text{optional}}$

Die Syntaxkomponenten im Einzelnen:

Syntaxelement	Semantik
`public`	Das Attribut ist allgemein für alle anderen Klassen sichtbar.
`protected`	Das Attribut ist ausschließlich für die definierende Klasse sowie diejenigen sichtbar die von aktuellen erben, sowie alle Klassen desselben Packages.
`private`	Das Attribut ist nur innerhalb der deklarierenden Klasse sichtbar.
`static`	Gültigkeitsbereich des Attributs ist die gesamte Klasse. D.h. alle Objekte dieser Klasse teilen sich *dasselbe* Attribut (= derselbe Speicherplatz). Ein so deklariertes Attribut hat in allen Ausprägungen denselben Wert; änderungen finden automatisch synchronisiert in allen Objekten statt. Ohne Angabe dieses Schlüsselwortes ist der Scope auf die Objektebene festgelegt.
`final`	Das Attribut ist konstant, und kann nach seiner (zwingend anzugebenden!) Initialisierung nicht mehr verändert werden.

`transient`	So ausgezeichnete Attribute sind nicht Teil des persistenten Objektzustandes, und werden bei einer Ablage auf Sekundärspeicher (File, Datenbank, etc.) ignoriert.
`volatile`	Kennzeichnet ein Attribut als bei Optimierungen nicht zu berücksichtigen. Die Semantik und Anwendung ist ähnlich zur aus C/C++ bekannten Mimik.

```
(1)public class Attributes {
(2)     int             testAttribute1;
(3)     public  short           testAttriubte2 = 99;
(4)     private         long            testAttribute3 = 5L;
(5)     protected boolean       testAttribute4;
(6)     volatile        private char testAttribute5;
(7)     public transient long testAttribute6 = 0x42;
(8)
(9)     static final double pi = 3.141592654;
(10)
(11)    TestClass testAttributeC1;
(12)    TestClass testAttributeC2 = new TestClass();
(13)} //class Attributes
(14)
(15)class TestClass {
(16)} //class TestClass
```

Beispiel 32: Einige Attributdefinitionen Attributes.java

Abschlußbemerkungen:

- Aus Gründen der Speicherplatzgründen empfiehlt sich die Definition von Konstanten (wie `pi` im Beispiel) als `static final`.
- Java erlaubt, anders als C++, keinerlei Datenstrukturen außerhalb von Klassen. Globale Variablen, sowie wiederverwendbare „Allzweckvariablen“ -- beispielsweise als Schleifenzähler -- können daher nicht in der gewohnten (Un-)Sitte realisiert werden.

2.4.4 Operationen und Methoden

Operationen und Methoden bilden das dynamische Verhalten eines Objekts ab. Während der Begriff *Operation* nur die Signatur, bestehend aus Rückgabetyp, Operationsnamen und der Parameterliste, bezeichnet, deckt *Methode* auch die programmiersprachliche Umsetzung ab.
Java erlaubt ausschließlich die Definition von Methoden innerhalb von Klassen. Globale Funktionen sind ebenso wie globale Variablen nicht möglich!

Vereinfachte **Syntax** einer Operation:

(`public`, `protected`, `private`, `abstract`, `static`, `final`, `synchronized`, `native`, `strictfp`)$_{\text{zero or more}}$ *ResultType* *Identifier* (*ParameterList*) (`throws` *ExceptionList*)$_{\text{optional}}$

Die Syntaxkomponenten im Einzelnen:

Syntaxelement	Semantik
`public`	Die Methode ist allgemein für alle anderen Klassen sichtbar.
`protected`	Die Methode ist ausschließlich für Klassen sichtbar die von der aktuellen erben, sowie alle Klassen desselben Packages.
`private`	Die Methode ist nur innerhalb der deklarierenden Klasse sichtbar.
`abstract`	Definiert analog zum Schlüsselwort auf Klassenebene, daß eine so gekennzeichnete Operation keine Implementierung bereitstellt; mithin keine Methode implementiert.
`static`	Gültigkeitsbereich dieser Methode ist die Klasse. In der Anwendung bedeutet dies, daß eine so deklarierte Methode sowohl auf der Klasse selbst, als auch einem Objekt dieser Klasse aufgerufen werden kann. Beispiel
`final`	Die Methode darf nicht in einer erbenden Klasse überschrieben werden.
`synchronized`	Auf einer so deklarierten Methode wird durch das Laufzeitsysteme ein wechselseitiger Ausschluß realisiert. Das bedeutet, daß sich nur jeweils ein Thread innerhalb einer Methode des Objektes befinden darf.

`native`	Eine so gekennzeichnete Methode ist nicht in Java realisiert, sondern wird als nativer Code einer anderen Programmiersprache (z.B. C/C++) realisiert. Das JDK bietet für die beiden genannten Sprachen Unterstützung in Form automatisch generierter Headerdateien an.
`strictfp`	Deaktiviert Nutzung plattformspezifischer Gleitkommahardware. analog strictfp auf Klassenebene

Aufgrund der strengen Typisierung der Programmiersprache ist jede Javaoperation über ihren Rückgabewert typisiert. Als **Rückgabetypen** stehen alle primitiven built-in Datentypen, alle Klassen und Schnittstellen, sowie der explizite *Nichttyp*`void` zur Verfügung.
Wie aus C/C++ bekannt geben `void`-Methoden keine Werte an den Aufrufer zurück, und können daher nicht innerhalb von Ausdrücken verwendet werden.
Ebenso analog der bekannten Mimik wird die explizite Rückgabe eines Wertes durch das Schlüsselwort `return` eingeleitet. Wie in (neuen) C/C++-übersetzern üblich, prüft auch Java die Existenz einer erreichbaren typkompatiblen Return-Anweisung innerhalb des Methodenrumpfes.

Die auf den Operationsnamen folgende **Parameterliste** setzt sich aus einer Folge von Typen und Parameternamen zusammen, kann jedoch auch leer sein.
Verursacht durch das Fehlen expliziter Referenztypen (wie Zeiger) werden alle Parameter, die primitve build-in Typen sind **by value** und alle objektartigen Parameter per Vorgabe **by reference** übergeben.
Hinweis: Es existiert kein Mechanismus dieses Verhalten zu überschreiben oder abzuändern! Lediglich für die by reference Interpretation der Primitivtypen ist mit den Wrapper Typen eine Standardmethodik vorgesehen.

Genaugenommen kommt auch für Objekte, die als Parameter einer Methode verwendet werden, eine Wertübergabe zum Einsatz. Jedoch wird in diesem Falle nicht der Wert des Objektes übergeben, d.h. es wird keine Kopie des Objektes erzeugt und an die aufzurufende Methode übergeben. Vielmehr wird eine Kopie der Referenz auf das Objekt übergeben.
Dieser Umstand führt dazu, daß auch objektwertige Parameter innerhalb eines Methodenrumpfes nicht direkt modifiziert werden können, sondern hierfür auf Methoden des zu verändernden Objekts zurückgegriffen werden muß.
Mehr Informationen hierzu.

Die **Signatur einer Javamethode** wird aus Operationsname und der Parameterliste, unter Berücksichtigung der Parameterreihenfolge, gebildet.
Sichtbarkeitsattribute, ebenso wie der Rückgabetyp und weitere Eigenschaften gehen nicht in die Signaturbildung ein.

Innerhalb des **Methodenrumpfs** sind alle aus 2.3 bekannten Kontrollstrukturen zugelassen. Darüberhinaus kann an beliebiger Stelle die Deklaration lokaler Variablen erfolgen. Im Gegensatz zu C/C++ ist die Lebensdauer lokaler Variablen strikt an die des umgebenden Blocks gebunden. Daher steht das Schlüsselwort `static` für die Variablendefinition innerhalb von Methoden nicht zur verfügung.

Der Aufruf von Methoden erfolgt in der bekannten *Punktnotation*, die in allgemeiner Form als oder `Objekt.Methode(Parameterliste)` oder entsprechend `Klasse.Methode(Parameterliste)` bei Klassenmethoden beschrieben werden kann.

Eine Sonderrolle spielt das **Schlüsselwort `this` im Methodenrumpf**. Es steht als impliziter Parameter in allen nicht-statischen Methoden zur Verfügung. `this` referenziert immer das aktuelle Objekt. Die lokale Variable `this` ist dabei immer von Typ `final` und erlaubt keine Neuzuweisungen.
Üblicherweise ist die Verwendung von `this` nicht explizit erforderlich. Zur Behebung von Namenskonflikten zwischen Übergabeparametern und lokalen Variablen leistet es jedoch wertvolle Dienste. Weitere Anwendung findet das Schlüsselwort zur Aufruf von Konstruktoren.

```
(1)public class ThisDemo {
(2)     private int i = 42;
(3)
(4)     public void setI(int i) {
(5)             this.i = i;
(6)     } //setI()
(7)
(8)     public int getI() {
(9)             return this.i;
(10)    } //getI()
(11)
(12)    public static void main(String[] args) {
(13)            ThisDemo td = new ThisDemo();
(14)            System.out.println("value of i="+td.getI() );
(15)            td.setI(45);
(16)            System.out.println("value of i="+td.getI() );
(17)    } //main()
(18)} //class ThisDemo
```

Beispiel 33: Zugriff auf Objekteigenschaften mit this ThisDemo.java

Bildschirmausgabe:

```
$java ThisDemo
value of i=42
value of i=45
```

Das Beispiel zeigt die Verwendung der Eigenobjektreferenz `this` innerhalb der Methoden `setI` und `getI`.
Während die Referenzierung von `i` innerhalb `getI` auch ohne vorstelltes explizites `this` eindeutig auflösbar ist, muß in `setI` das Schlüsselwort zwingend angegeben werden um dem Namenskonflikt zwischen dem Parameter `i` und dem gleichnamigen Attribut aufzulösen.

Analog zu den Klassenattributen werden durch das Schlüsselwort `static`**Klassenmethoden** definiert.
Auf sie kann unabhängig von der Existenz konkreter Objekte der Klasse zugegriffen werden.
Aufgrund ihrer instanzenunabhängigen Natur besitzen statische Methoden keine `this` Referenz, da ein so zu referenzierendes Objekt nicht exisitiert.

```
(1)public class StaticMethodTest {
(2)     public static void helloWorld() {
(3)             System.out.println("hello world");
(4)     } //helloWorld()
(5)
(6)     public static void main(String[] args) {
(7)             StaticMethodTest.helloWorld();
(8)             (new StaticMethodTest()).helloWorld();
(9)     } //main()
(10)} //class StaticMethodTest
```

Beispiel 34: Statische Methoden StaticMethodTest.java

***Besondere* Methoden:**
Konstruktoren
Identisch benannt zur beherbergenden Klasse

Syntaktisch ähneln die Konstruktoren den „normalen“ Operationsdeklarationen. Jedoch mit dem Unterschied, daß kein Rückgabetyp spezifiert werden kann.
Der Konstruktorenaufruf wird automatisch durch den übersetzer plaziert.
Existiert kein expliziter Konstruktor, so wird während des übersetzungsvorganges ein unparametrisierter Vorgabekonstruktor erzeugt. Dieser enthält keinerlei eigene Funktionalität, sondern ruft nur den entsprechenden parameterlosen Konstruktor der Superklasse auf. Verfügt eine Klasse hingegen ausschließlich über parametrisierte Konstruktoren, so wird kein unparametrisierter Defaultkonstruktor angelegt.
Aufrufe unter den verschiedenen Konstruktoren können durch das bekannte **Schlüsselwort `this`** vorgenommen werden. Hierbei werden die einzelnen Konstruktoren wie normale Methoden behandelt. Der Aufruf erfolgt durch `this`gefolgt von den Konstruktorenparametern in runden Klammern. Der kaskadierende Konstruktorenaufruf muß zwingend als erstes Statement des Anweisungsblockes plaziert werden.
Zwar verfügt die Konstruktormethode über keinen expliziten Rückgabetyp, wird aber intern als void-Typisiert behandelt.

```
(1)public class Construct {
(2)     public Construct() {
(3)             System.out.println("object created using explicitly stated parameterless
Constructur");
(4)     } //constructor
(5)
(6)     public Construct(int i) {
(7)             System.out.println("object Constructed by parametrized Constructor, parameter
i="+i);
(8)     } //constructor
(9)
(10)    public static void main(String args[]) {
(11)            new Construct();
(12)            new Construct(1);
(13)            new TestClass1();
(14)    } //main()
(15)} //class Construct
(16)
(17)class TestClass1 extends TestClass2 {
(18)    //nothing!
(19)} //TestClass1
(20)
(21)class TestClass2 {
(22)    TestClass2() {
(23)            this(1);
(24)            System.out.println("Constructor of TestClass2 called");
(25)    } //constructor
(26)
(27)    TestClass2(int i) {
(28)            this ((short) 2,(byte) 3);
(29)            System.out.println("Constructor of TestClass2 called paramter i="+i);
(30)    } //constructor
(31)
(32)    TestClass2(short s, byte b) {
(33)            System.out.println("Constructor of TestClass2 called paramter s="+s+" parameter
b="+b);
(34)    } //constructor
(35)} //class TestClass2
```

Beispiel 35: Verschiedene Konstruktoren Construct.java

Nicht-öffentliche Konstruktoren und Fabriken:
Häufig anzutreffen sind Klassen, die zwar einen explizit definierten Konstruktor bieten, diesen jedoch `private` deklarieren. In der Konsequenz ist eine Objekterzeugung per `new` nicht möglich.
Zumeist wird dieser Ansatz angewandt, wenn die Objekterzeugung aufwendig ist und nicht dem Anwender überlassen werden kann oder soll.
Um dennoch Objekte der betreffenden Klasse erzeugen zu können wird eine statische *Fabrik*-Methode eingeführt, welche die Objekterzeugung übernimmt und implizit `new` aufruft.

```
(1)public class Construct2 {
(2)     private Construct2() {
(3)             System.out.println("hello world");
(4)     } //constructor
(5)
(6)     public static void main(String[] args) {
(7)             new Construct2();
(8)             /* new OtherClass(); would fail since constructor is declared to be private */
(9)             OtherClass oc = OtherClass.OtherClassFactory(); //object creation using simple
factory approach
(10)    } //main()
(11)} //class Construct2
(12)
(13)class OtherClass {
(14)    public static OtherClass OtherClassFactory() {
(15)            return new OtherClass();
(16)    } //constructor
(17)
(18)    private OtherClass() {
(19)            System.out.println("other class created");
(20)    } //constructor
(21)} //class OtherClass
```

Beispiel 36: Nicht-öffentliche Konstruktoren und Fabrikmethoden zur Objekterzeugung Construct2.java

Bildschirmausgabe:

```
$java Construct2
hello world
other class created
```

Der Konstruktorenaufruf innerhalb der Klasse `construct2` kann bei Absetzen des `new` ausgeführt werden, da aus der Klasse heraus (konkret: innerhalb einer der statischen Methode `main`) ein Objekt dieser Klasse erzeugt wird -- der private Konstruktor ist somit zugreifbar.
Hingegen ist die Erzeugung eines Objekts von `otherClass` per `new` nicht möglich, da der dort definierte private Konstruktor nicht aus `construct2` heraus referenzierbar ist. Der entsprechende Fehler wird bereits zum Übersetzungszeitpunkt erkannt.
Die Erzeugung von Objekten der Klasse `otherClass` ist ausschließlich über die statische Methode `otherClassFactory` möglich. Sie ruft intern den privaten Konstruktor auf, der an dieser Stelle sicht- und zugreifbar ist.
(siehe Java API Specification, siehe Java Language Specification)

Ein Beispiel für eine Klasse, die weder über eine Factorymethode, noch über öffentliche Konstruktoren verfügt ist die Standard-API-Klasse `Void`.
Ihre Implementierung ist in der API wie folgt festgelegt:

```
public final
class Void {
    public static final Class TYPE = Class.getPrimitiveClass("void");

    private Void() {}
}
```

Noch vor dem Konstruktor werden die **statischen Initialisierungen** aufgerufen. Aufgrund der Reihenfolge in der Methodenabarbeitung existiert zum Ablaufzeitpunkt der statischen Initialisierungen das zu erzeugende Objekt noch nicht; der `static`-Block wird quasi zum Ladezeitpunkt der Klasse ausgeführt. Daher können zu diesem Zeitpunkt keine Zugriffe auf nichtstatische Speicherobjekte (Methoden und Attribute) erfolgen. Zugriffe auf statische Attribute und Methoden sind jedoch möglich.
Der Initialisierungsblock darf nicht durch Unterbrechungsanweisungen wie `break` oder `return` vorzeitig verlassen werden. Hierunter fallen auch Exceptions, die zum vorzeitigen Verlassen des Anweisungsblockes führen würden.

```
(1)public class StaticInit {
(2)     static int i=42;
(3)
(4)     static {
(5)             System.out.println("value of i="+i);
(6)             helloWorld();
(7)             i = 43;
(8)     } //static
(9)
(10)    public static void helloWorld() {
(11)            System.out.println("hello world");
(12)    } //helloWorld()
(13)
(14)    public static void main(String[] args) {
(15)            StaticInit myObj = new StaticInit();
(16)    } //main()
(17)
(18)    public StaticInit() {
(19)            System.out.println("value of i="+i);
(20)    } //constructor
(21)} //class StaticInit
```

Beispiel 37: Statische Initialisierung StaticInit.java

Bildschirmausgabe:

```
$java StaticInit
value of i=42
hello world
value of i=43
```

Das Beispiel zeigt zunächst den Zugriff auf das statische Attribut (Klassenattribut) `i` direkt nach dessen Definition und Initialisierung. Zu diesem Zeitpunkt ist der Initialisierungswert 42 gesetzt.
Der Aufruf von `helloWorld` demonstriert die Möglichkeit vor der Objekterzeugung statische Methoden auszuführen.
Nach Abarbeitung der statischen Initialisierung wird der Konstruktor bearbeitet. In ihm steht der Wert von `i` nur noch in der durch die statische Initialisierung modifizierten Form zur Verfügung.

Ausführungsreihenfolge der konstruierenden Codesequenzen:
(als Pseudocode):

```
for-each (Superclass s) {
   s.AttributeInitialization()
   s.StaticBlock()
} //for-each
for-each (Superclass s) {
   s.Constructor()
}
```

Beginnend mit der hierachiehöchsten Superklasse werden zunächst die Initialisierungen der Attribute, im Anschluß daran (falls vorhanden) der `static`-Block dieser Klasse, abgearbeitet.
In einem zweiten Durchlauf über die Klassenhierarchie werden die Konstruktoren der Superklassen in derselben Reihenfolge zur Ausführung gebracht.

```
(1)public class SuperClassConstruct extends Class2 {
(2)     static int i=6;
(3)
(4)     static {
(5)             System.out.println("i="+i--);
(6)             System.out.println("static initialization of class SuperClassConstruct");
(7)             System.out.println("i="+i);
(8)     } //static
(9)
(10)    public SuperClassConstruct() {
(11)            System.out.println("object of class SuperClassConstruct constructed");
(12)    } //constructor
(13)
(14)    public static void main(String[] args) {
(15)            new SuperClassConstruct();
(16)    } //main()
(17)} //class SuperClassConstruct
(18)
(19)class Class2 extends Class1 {
(20)    static int i=4;
(21)
(22)    static {
(23)            System.out.println("i="+i--);
(24)            System.out.println("static initialization of class Class2");
(25)            System.out.println("i="+i);
(26)    } //static
```

```
(27)
(28)    public Class2() {
(29)            System.out.println("object of class Class2 constructed");
(30)    } //constructor
(31)} //class Class2
(32)
(33)class Class1 {
(34)    static int i=2;
(35)    static {
(36)            System.out.println("i="+i--);
(37)            System.out.println("static initialization of class Class1");
(38)            System.out.println("i="+i);
(39)    } //static
(40)
(41)    public Class1() {
(42)            System.out.println("object of class Class1 constructed");
(43)    } //constructor
(44)} //class Class2
```

Beispiel 38: Initialisierungsreihenfolge SuperClassConstruct.java

Bildschirmausgabe:

```
$java SuperClassConstruct
i=2
static initialization of class class1
i=1
i=4
static initialization of class class2
i=3
i=6
static initialization of class superClassConstruct
i=5
object of class class1 constructed
object of class class2 constructed
object of class superClassConstruct constructed
```

Destruktoren --finalize
Zwar können in Java, anders als in C++, Objekte nicht durch Destruktorenaufruf zerstört werden, sondern nur duch Null-Zuweisung als nicht mehr benötigt markiert werden, Destruktoren werden jedoch weiterhin explizit zur Verfügung gestellt.
Destruktoren werden nicht durch den Anwender aufgerufen, sondern implizit erst zum Zerstörungszeitpunkt eines Objekts. Genaugenommen ist die Ausführung des Destruktors nicht garantiert. Terminiert die Applikation vor dem Ablauf des Garbage Collectors, so werden die Objekte implizit durch das Betriebssystem aus dem Speicher entfernt, ohne das die Java-Laufzeitumgebung zunächst die finalize-Methoden aufruft.
Die Sichtbarkeitseinschränkung von finalize muß mindestens protected sein. Dies rührt von der impliziten Überschreibung der von Object ererbten Methode finalize her. (vgl. java.lang.Object:finalize)

```
(1)public class DestruktorTest {
(2)     public static void main(String[] args) {
(3)             Test t1 = new Test();
(4)             t1 = null;
(5)             System.gc();
(6)     } //main()
(7)} //class DestruktorTest
(8)
(9)class Test {
(10)    public void finalize()  {
(11)            System.out.println("destructor of class Test called");
(12)    } //finalize()
(13)} //class Test
```

Beispiel 39: Destruktorenaufruf durch Garbage Collector DestruktorTest.java

Im Beispiel wird das Objekt t1 zunächst durch Nullsetzung zum Löschen markiert, und im anschließenden Garbage Collector-Aufruf gelöscht. Während des Entfernens aus dem Speicher wird die darstellte Nachricht am Bildschirm ausgegeben.

Hauptmethode -- main
Pro Java-Applikation kann maximal eine Methode der Signatur main(String[]) existieren. Sie wird beim Startvorgang innerhalb der öffentlichen Klasse gesucht, die namensgebend für die Klassendatei ist.
Applets verfügen hingegen über keine automatisch aufgerufene main-Methode, sondern werden durch init() Initialisiert und durch das im Anschluß darauf aufgerufene start() ausgeführt.

Die Methode toString
ist auf der Superklasse Object aller Javaklassen definiert. Sie wird von allen Klassen der Standard-API implementiert. Durch sie wird immer eine Zeichenkettenrepräsentation des aktuellen Objekts zurückgegeben. Diese Methode wird standardmäßig bei der Ausgabe per System.out.println aufgerufen.

Zum Abschluß: Sichtbarkeit von Attributen und Methoden im Überblick:

	definierende Klasse	Abgeleitete Klasse	Package	Alle anderen
default (keine explizite Festlegung)	✓			
`public`	✓	✓	✓	✓
`private`	✓			
`protected`	✓	✓ Zugriff auf Attribute und Methoden der Subklasse	✓	

Anmerkung: Der Zugriff auf als `protected` deklarierte Attribute und Methoden ist auch über Objektreferenzen möglich, die denselben Typ haben wie die definierende Klasse.

2.4.5 Aufzählungstypen

Ab Version 1.5 führt Java mit dem Schlüsselwort `enum` einen eigenständigen und expliziten Mechanismus zur Definition von Aufzählungstypen ein.

Konzeptionell sind Aufzählungstypen identisch zu Variablen, die innerhalb des Rumpfes einer Methode deklarierten werden oder Variablen und Attributen einer Klasse gleichgestellt. Aus diesem Grunde ähnelt die Definitionssyntax auch den bekannten Darstellungsformen:

Vereinfachte **Syntax** der Enum-Definition:

(public, protected, private)$_{\text{one of}}$	static$_{\text{opt}}$	final$_{\text{opt}}$	`enum`	*Identifier*	{*value*}

Im einfachsten Anwendungsfall besteht die Definition eines Aufzählungstypen aus der durch Kommata voneinander separierten vollständigen Aufzählung aller Werte.
Das Beispiel zeigt die Bildung des Aufzählungstyps `season`, der durch die zulässigen Werte `winter`, `spring`, `summer` und `fall` konstituiert wird.

```
(1)public class EnumTest1 {
(2)     public static void main(String[] args) {
(3)             enum season { winter, spring, summer, fall; }
(4)
(5)             season s1 = season.winter;
(6)
(7)             if (s1 == season.winter)
(8)                     System.out.println("It's "+s1);
(9)
(10)            season s2 = season.winter;
(11)    } //main()
(12)} //class EnumTest1
```

Beispiel 40: Definition eines einfachen Aufzählungstyps EnumTest1.java

Die Verwendung von Aufzählungstypen entspricht der von primitiven Typen. Aus diesem Grund ist keine Anforderung von Speicherplatz durch das new-Schlüsselwort notwendig. Der Übersetzer verhindert sogar aktiv die Instanziierung eines Aufzählungstyps via new und bricht beim Versuch mit der Fehlermeldung `enum types may not be instantiated` ab.
Im Gegensatz zu Ausprägungen der Primivtypen besitzen Aufzählungsinstanzen jedoch keinen Vorgabewert mit dem sie standardmäßig initialisiert werden. Stattdessen führt die Verwendung einer nichtinitialisierten Ausprägung eines Aufzählungstypen zu einem Übersetzungsfehler (Fehlermeldung: `variable ... might not have been initialized`).

Ansonsten bietet die Umsetzung die für Primitivtypen bekannten Eigenschaften. So liefert die Ausgabe diejenige Zeichenkette (d.h. den Wert) mit dem Aufzählungsinstanz belegt wurde.
Ebenfalls identisch zu den Primitivtypen besitzen Aufzählungsinstanzen keine Identität. Dies äußert sich darin, daß zwei mit demselben Wert belegte Aufzählungen als identisch betrachtet werden.

Nicht identisch sind hingegen Ausprägungen verschiedener Aufzählungstypen, die vermeintlich denselben Wert enthalten, d.h. in deren zur Definition verwendeten Werteliste sich lexikalisch dieselben Einträge finden.
So würde die nachfolgende Zuweisung bereits durch den Übersetzer (mit der Fehlermeldung `incompatible types`)

abgelehnt

```
enum seasonE { winter, spring, summer, fall; };
enum seasonG { winter, frühling, sommer, herbst; };
seasonE s = seasonG.winter;
```

Dasselbe gilt auch für den Versuch des Vergleichs der Inhalte zweiter Aufzählungsausprägungen, wie sie durch das nachstehende Codefragment versucht wird.

```
enum seasonE { winter, spring, summer, fall; };
enum seasonG { winter, frühling, sommer, herbst; };
seasonE s1 = seasonE.winter;
seasonG s2 = seasonG.winter;
if (s1==s2) ...
```

Auch in diesem Fall wird bereits zum Übersetzungszeitpunkt durch die Fehlermeldung `incomparable types: seasonG and seasonE` auf den Fehler hingewiesen.

In Erweiterung der bisher vorgestellten Syntax lassen sich Aufzählungstypen sogar „klassenartig" ausbauen. Diese Erweiterung erlaubt es die Elemente des Aufzählungstypen wahlfrei an selbstdefinierte Eigenschaften zu binden. Konzeptionell werden diese Eigenschaften dabei als Attribute des Aufzählungstypen aufgefaßt, die durch einen durch den Programmierer bereitzustellenden Konstruktor zugewiesen werden. Der Konstuktorenaufruf erfolgt dabei automatisch durch das Laufzeitsystem zum Definitionszeitpunkt eines Aufzählungstypen für alle konstituierenden Inhaltselemente. Das nachfolgende Beispiel zeigt eine Verwendung des erweiterten Konzepts:

```
(1)public class EnumTest2 {
(2)     public enum Coin {
(3)             penny(1), nickel(5), dime(10), quarter(25);
(4)     private int value;
(5)             Coin(int value) {
(6)                     System.out.println("creating: "+value);
(7)                     this.value = value;
(8)             } //constructor
(9)     public int value() {
(10)                    return value;
(11)            } //value()
(12)    } //enum Coin
(13)
(14)    public static void main(String[] args) {
(15)            Coin c1 = Coin.dime;
(16)            System.out.println( c1.value() );
(17)            } //main
(18)} //class EnumTest2
```

Beispiel 41: Definition eines klassenartigen Aufzählungstyps EnumTest2.java

Das Beispiel definiert den Typ `Coin` mit seinen Inhaltstypen `penny`, `nickel`, `dime` und `quarter`. Den Inhaltstypen wird durch Konstruktoraufruf jeweils ein Inhaltswert zugeordnet. Dieser Wert wird der definierten privaten Variable `value` zugewiesen.
Durch die Methode `value` kann der dem jeweiligen Inhaltstyp zugewiesene Wert ausgelesen werden.
Methoden, die den Inhalt der definierten Variable schreiben können zwar definiert werden, jedoch werden Wertänderungen nicht auf die vordefinierten Inhaltstypen synchronisiert.

Die Anzahl der intern mit einem Wert verbundenen Festwerte ist hier bei nicht beschränkt. Das abschließende Beispiel zeigt die Zuweisung von zwei Einzelwerten im Konstruktor:

```
(1)public class EnumTest3 {
(2)     public enum Time {
(3)             highNoon(12,00), sunSet(20,30), sunRise(5,30), teaTime(17,00);
(4)     private int hour;
(5)             private int minute;
(6)
(7)             Time(int hour, int minute) {
(8)                     this.hour = hour;
(9)                     this.minute = minute;
(10)            } //constructor
(11)            public void printTime() {
(12)                    System.out.print(hour+":"+minute);
(13)            } //printTime()
(14)    } //enum Time
(15)
(16)    public static void main(String[] args) {
(17)            Time t1 = Time.sunSet;
(18)            System.out.print( "Sunset is at: ");
(19)            t1.printTime();
(20)            } //end main
(21)} //class EnumTest3
```

Beispiel 42: Definition eines klassenartigen Aufzählungstyps EnumTest3.java

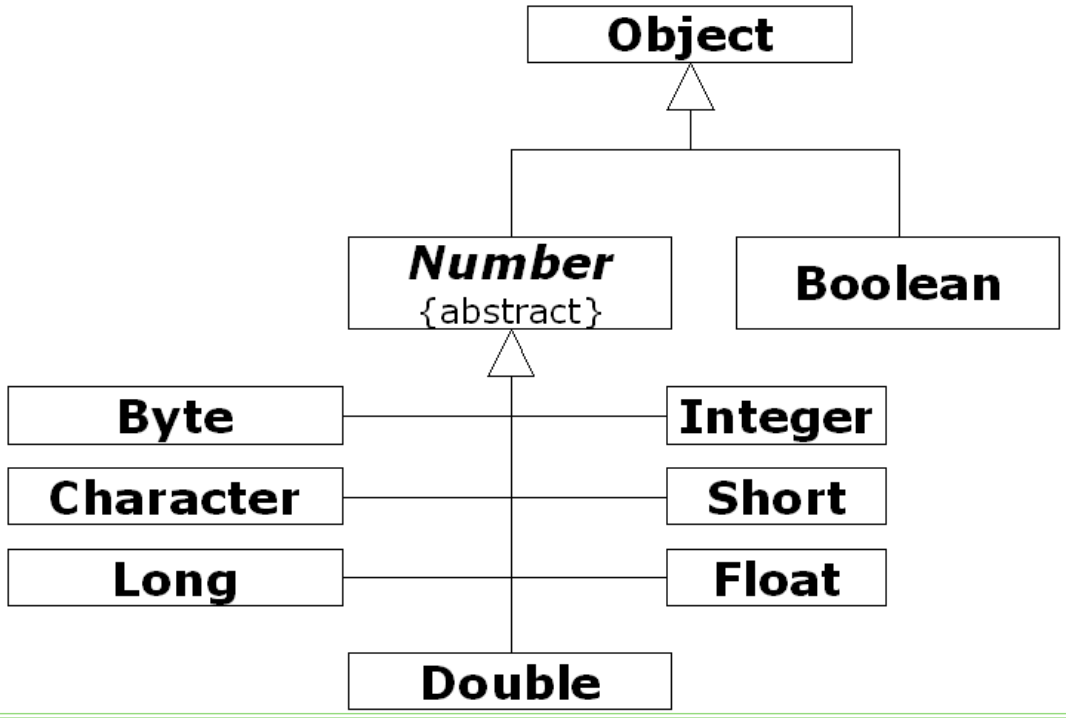
2.4.6 Wrapper-Typen

Korrespondierend zu jedem primitiven Datentypen in Java gibt es einen *Wrapper Typen*.
Sie kapselt den zugrundeleigenden Primitivtyp in einer eigenen Klasse, und stellt einige Servicemethoden bereit.

Objekte aller Wrappertypklassen können nur bei ihrer Erzeugung mit Werten versehen werden, die über die gesamte Lebensdauer nicht mehr verändert werden können.

Primitivtyp	Wrapper Typ
`boolean`	`Boolean`
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`void` Dieser Typ ist nicht als Datentyp für Variablen und Attribute verfügbar, sondern kann nur als Rückgabetyp von Operationen spezifiziert werden.	`Void` Kann nicht instanziert werden.

Die Abbildung 9 zeigt die Organisation der Wrapperklassen innerhalb der Standard-API im Überblick:




```
(1)public class CBRCBV {
(2)     public static void main(String[] args) {
(3)             TestClass testObj = new TestClass();
(4)             int testVar;
(5)             Byte testWrapperType;
(6)
(7)             testObj.setS((short) 42);
(8)             testWrapperType = new Byte((byte) 12);
(9)             testVar = 50;
(10)
(11)            System.out.println("values before method run");
(12)            System.out.println("testVar = "+testVar);
(13)            System.out.println("testWrapperType = "+testWrapperType.byteValue() );
(14)            System.out.println("testObj.s = "+testObj.getS() );
(15)
(16)            aMethod(testVar, testWrapperType, testObj);
(17)
(18)            System.out.println("values after method run");
(19)            System.out.println("testVar = "+testVar);
```

```
(20)            System.out.println("testWrapperType = "+testWrapperType.byteValue() );
(21)            System.out.println("testObj.s = "+testObj.getS() );
(22)    } //main()
(23)
(24)    private static void aMethod(int i, Byte b, TestClass tc)        {
(25)            i++;
(26)            b = new Byte((byte) 0);
(27)            tc.setS( (short) (tc.getS()+1) );
(28)
(29)            System.out.println("values within method after modification:");
(30)            System.out.println("testVar = "+i);
(31)            System.out.println("testWrapperType = "+b.byteValue() );
(32)            System.out.println("testObj.s = "+tc.getS() );
(33)    } //aMethod()
(34)} //class CBRCBV
(35)
(36)class TestClass {
(37)    private short s;
(38)
(39)    public short getS() {
(40)            return s;
(41)    } //getS
(42)
(43)    public void setS(short s) {
(44)            this.s = s;
(45)    } //getS()
(46)} //TestClass
```

Beispiel 43: Verwendung von primitiven, Wrapper und objektwertigen Typen CBRCBV.java

Bildschirmausgabe:

```
values before method run
testVar = 50
testWrapperType = 12
testObj.s = 42
values within method after modification:
testVar = 51
testWrapperType = 0
testObj.s = 43
values after method run
testVar = 50
testWrapperType = 12
testObj.s = 43
```

Im Beispiel werden zunächst Exemplare der drei verschiedenen Typfamilien -- Primitivtyp, Wrapper Typ und Objekt -- erzeugt und mit Werten versehen.
Innerhalb der Methode `aMethod`, die alle drei Variablen als Übergabeparameter erhält, werden die Inhalte lokal geändert. Während die `int`-Variable direkt beschrieben werden kann, wird die Änderung des objektwertigen Parameters `tc` durch eine Methode der Klasse `TestClass` realisiert. Auf dem Wrapper Typen ist durch die Java-API keine Modifikationsroutine vorgesehen. Daher wird der übergebenen Referenz ein neues Objekt zugewiesen.
Nach Ausführung der Methode `aMethod` werden die Werte nochmals ausgegeben. Hier zeigt sich, daß auch für Objekte **kein echtes Call-by-Reference** zur Verfügung steht, sondern lediglich eine Kopie auf die Referenz übergeben wurde. Aus diesem Grunde sind die Änderungen sowohl am übergebenen Primitivtypen als auch an der Variable des Typs `Byte` verloren.
Als einzige Möglichkeit zur Realisierung von Modifikationen an einem Objekt erweisen sich die Methoden dieses Objekts.

Mehr Information hierzu:

- Understanding that parameters are passed by value and not by reference
- Pass-by-value semantics in Java applications

Alle Wrappertypen bieten bestimmte Servicemethoden an, die in der Praxis häufig auftretende Anforderungen erfüllten. Darunter fallen neben der Ausgabe des Wrappertypeninhaltes (d.h. des gekapselten Primtivwertes) auch Umsetzungen der auf Objekten nicht zur Verfügung stehenden Typumwandlungen (casts), sowie Methoden zur Erzeugung der Primitivrepräsentationen aus anderen als den Wrappertypendarstellungen (z.B. aus beliebigen Zeichenketten).

Wichtige und gebräuchliche Servicemethoden, sowie weitere Charakteristika der Wrappertypen:

- Alle Wrapperklassen sind `final` deklariert, und können daher nicht weiter spezialisiert werden.
- **`...Value`**
  Beispiele: `booleanValue`, `byteValue`, `shortValue`, `intValue`, `longValue`, `floatValue`, `doubleValue`
  Die abstrakte Signatur lautet in allen Fällen: `public` *`T`* *`T`*`Value()`, wobei *`T`* der jeweilige Primitivtyp ist (entsprechend für den Wrappertypen um int: `public int intValue()`.
  Gibt den intern gekapselten Wert unverändert zurück.
  Darüberhinaus existieren Methoden zur Ausgabe eines numerischen Wrappertypen in einen beliebigen numerischen Primitivtypen. Diese ersetzen somit die explizite Typumwandlung.
- **Konstruktoren**

Erlauben die Erzeugung des Wrappertypen aus einem Ausdruck vom Typ des gekapselten Typen oder aus einem String. (Beispiele: `Boolean (boolean value)`, `Boolean (String s)`, `Integer (int value)`, `Integer (String s)`) Ist dies nicht möglich so wird eine NumberFormatException ausgelöst. Dies ist der liegt vor, wennder aus der Zeichenkette extrahierte Wert den zulässigen Gültigkeitsbereich (d.h. er liegt unter `MIN_VALUE` oder oberhalb `MAX_VALUE`) überschreitet. Ebenso wenn die Zeichenkette Symbole enthält, die nich in eine numerische Darstellung überführt werden können.

- **`compareTo`**
  Vergleicht den gekapselten Wert zweier Wrappertyp-Objekte.
  Im Gegensatz hierzu vergleicht `equals` die Objekte direkt.

```
(1)public class WrapperComparison {
(2)     public static void main(String[] args) {
(3)             Integer i1 = new Integer ( 42 );
(4)             Integer i2 = new Integer ( 42 );
(5)
(6)             System.out.println("i1==i2="+ (i1==i2) );
(7)             System.out.println("i1.equals(i2)="+ (i1.equals(i2) ));
(8)             System.out.println("i1.compareTo(i2)="+ (i1.compareTo(i2)));
(9)     } //main()
(10)} //class WrapperComparison
```

Beispiel 44: Vergleich von Wrapperobjekten WrapperComparison.java

Bildschirmausgabe:

```
i1==i2=false
i1.equals(i2)=true
i1.compareTo(i2)=0
```

Hinweis:
Obwohl die Semantik der Operation `equals` für Objekte der Klasse `Object` und deren Subklassen als *größtmögliche unterscheidende Äquivalenzrelation* (im Original) festgelegt ist weicht z.B. die Implementierung in der Standard-API-Klasse `String` von dieser Maßgabe ab. Sie liefert bereits `true` wenn die beiden Zeichenketten inhaltsgleich sind, jedoch verschiedene Objekte darstellen.

- **`parse...`**-Methode
  Die Fähigkeit der Konstruktoren, Wrapperobjekte -- nach vorheriger Gültigkeitsprüfung -- aus Zeichenketten zu erzeugen steht auch lösgelöst von der Objektkonstruktion zur Verfügung.
  Auf allen numerischen Wrappertypen werden `parseT`-Methoden angeboten, wobei *T* den konkreten Primitivtypen bezeichne. Diese liefern, im Falle der Gültigkeit, aus einer Zeichenkette den jeweiligen Primitvtypen zurück.
  Beispiele: `parseByte (String s)`, `parseByte(String s, int radix)`, `parseInt (String s)`, `parseInt (String s, int radix)`, etc.)
- **`valueOf`**
  Für alle Wrappertypen integraler Primitivtypen ist mit der `valueOf`-Methode eine Factorymethode zur Erzeugung eines neuen Wrapperobjektes aus einer Zeichenkette definiert.
- Veränderbare Wrapperklassen für die primitiven Typen stehen mit den *CORBA Holdern* zur Verfügung.
  (siehe `BooleanHolder`, `byteHolder`, `ShortHolder`, `IntHolder`, `LongHolder`, `FloatHolder`, `DoubleHolder`).
- Die üblichen arithmetischen Operationen stehen zur Verknüpfung von Wrapperobjekten nicht zur Verfügung (kein Operator overloading!).

2.4.7 Boxing/Unboxing

Zwar realisiert Java eine grundlegende Trennung zwischen Primitivtypen und Klassen, die Wertausprägungen nicht als Objekte auffaßt. Dennoch wird diese Maßgabe durch das in der Sprachversion 1.5 eingeführte dynamische Umwandlung zwischen primitiven Werten und den zugehörigen Wrapperklassen aufgeweicht. Die Integration dieses als *Boxing/Unboxing* bezeichnete Verfahren trägt den gesammelten Anwendererfahrungen Rechnung, die eine Vereinfachung der aufwendigen Wrapper-Objekterzeugung bzw. Wertextraktion aus bestehenden Wrapperobjekten fordern. Das namensgebende Begriffspaar bezeichnet hierbei diese beiden Schritte, d.h. *Boxing* die automatische Wrapper-Objekterzeugung bzw. *Unboxing* die Wandlung eines objektgekapselten Wertes in einen Primitivwert.

Die automatische Typwandlung wird in Java ausschließlich für Übergabeparameter angeboten, wie das nachfolgende Beispiel zeigt:

```
(1)public class BUBTest1 {
(2)     public static void main(String[] args) {
(3)             BUBTest1 obj = new BUBTest1();
(4)             obj.boxIt( new Integer(42) ); //nothing new
(5)             obj.boxIt( 42 ); //dynamic boxing
(6)
(7)             obj.unBoxIt( 42 ); //nothing new
(8)             obj.unBoxIt( new Integer(42) ); //dynamic unboxing
(9)     } //main()
(10)    public void boxIt(Integer i) {
(11)            System.out.println("value="+i);
(12)    } //boxIt
(13)    public void unBoxIt(int i) {
```

```
(14)            System.out.println("value="+i);
(15)    } //unBoxIt
(16)} //class BUBTest1
```

Beispiel 45: Dynamisches Boxing/Unboxing BUBTest1.java

Das Beispiel zeigt neben den jeweils singnaturkonformen Aufrufen auch die Verwendung der dynamischen Typwandlung; für die Methode `boxIt` die automatische Erzeugung eines innerhalb des Methodenrumpfes referenzierten Wrapper-Objekts bzw. für `unBoxIt` die Extraktion des im übergebenen Wrapper-Objekt enthaltenen `int` Primitivwertes.

Der Boxingvorgang kann außer für die Wrapper-Typen auch für Objekte der Klasse `Object` durchgeführt werden, da diese als Superklasse aller Wrapper-Typen angelegt sind ist diese Konversion stets typkonform. Das nachfolgende Beispiel zeigt die Anwendung:

```
(1)public class BUBTest2 {
(2)     public static void main(String[] args) {
(3)             BUBTest2 obj = new BUBTest2();
(4)             obj.boxIt( new Integer(42) ); //nothing new
(5)             obj.boxIt( 42 ); //dynamic boxing
(6)
(7)     } //main()
(8)     public void boxIt(Object i) {
(9)             System.out.println("value="+i);
(10)    } //boxIt
(11)} //class BUBTest2
```

Beispiel 46: Dynamisches Boxing für den Typ Object BUBTest2.java

Das Beispiel zeigt die bisher schon mögliche Verwendung einer Ausprägung des Typs `Integer` an einer Stelle, an der eine Ausprägung von `Object` erwartet wird, was unter Nutzung der Subklassenpolymorphie möglich ist. Ausgehend von diesem Sachstand ist die Realisierung des dynamischen Boxings im Beispiel zu verstehen. Das Auftreten der `int`-Zahl wird automatisch in eine Ausprägung der Klasse `Integer` konvertiert, die dann typkonform anstelle der erwarteten `Object`-Instanz verwendet werden kann.

Die umgekehrte Nutzung, d.h. die Verwendung einer Ausprägung von `Object` an einer Stelle, an der eine konkrete `int`-Zahl erwartet wird, ist --- nach Maßgabe der nicht typsicher möglichen Konversion entgegen der Vererbungsrichtung --- nicht möglich.

2.4.8 Vererbung

Wie in objektorientierten Sprachen üblich, untersütützt auch Java das Konzept der Vererbung. Die wesentlichen Hintergründe und Anwendungsgebiete, sowie Designaspekte sind aus früheren Lehrveranstaltungen bekannt, und werden daher an dieser Stelle nicht mehr wiederholt.

Charakteristika:

- Anders als C++ unterstützt Java **nur Einfachvererbung**!
- Syntaktisch wird Vererbung durch das Schlüsselwort `extends` ausgedrückt.
- Zur **Schnittstellenvererbung** (in C++ üblicherweise als *pure virtual class* realisiert) exisitert mit den Interfaces ein explizites Sprachkonzept
- Subklasse erbt nicht private Methoden und Attribute der Superklasse.
- Vererbung wird implizit als **Spezialisierung** aufgefaßt.
  Jede Ausprägung einer spezialisierten Klasse kann daher an jeder Stelle im Programm verwendet werden, an der eine Ausprägung der Superklasse erwartet wird (Liskov'sches Substitutionsprinzip). Hierbei erfolgt eine implizite Typumwandlung (*up cast*).
  Die originale Typinformation bleibt dessen ungeachtet erhalten.
- Umwandlung einer Subklassenausprägung in eine der Superklasse bedarf eines expliziten Cast-Statements.
  Schlägt dieses fehl, wird eine `ClassCastException` ausgelöst (C++ retourniert hier ein `null` Objekt).

```
(1)public class InheritanceDemo {
(2)     public static void main(String[] args) {
(3)             C2 myC2 = new C2();
(4)
(5)             System.out.println("attrib2="+myC2.attrib2);
(6)             System.out.println("inherited attrib1="+myC2.attrib1);
(7)
(8)             C1 myC1 = myC2; //implicit type cast (up cast!)
(9)
(10)            C1 myC11 = new C1();
(11)
(12)            C2 myC22 = (C2) myC11;  //explicit type cast needed (down cast!)
(13)                                                                    //will throw a
ClassCastException
(14)
```

```
(15)    } //main()
(16)} //class InheritanceDemo
(17)
(18)class C1 {
(19)    public int attrib1=1;
(20)} //class C1
(21)
(22)class C2 extends C1 {
(23)    public int attrib2=2;
(24)} //class C2
```

Beispiel 47: Erzeugung einer ClassCastException InheritanceDemo.java

Die Umwandlung des C2 Objektes in eines vom Typ C1 erfolgt implizit und automatisch.
Beim Versuch ein Objekt der Klasse c1 in ein Objekt der Klasse C2 umzuwandeln (*down cast* -- Subklassenobjekt wird in Superklassenobjekt gewandelt) wird eine ClassCastException erzeugt.

- Dynamische Bindung von nicht statischen Methoden (analog virtueller Funktionen in C++).
  Dieses Verhalten kann durch Kennzeichnung der Methode als `final` unterbunden werden.
  Klassenmethoden sind ebenso von diesem Verhalten ausgenommen.

```
(1)public class DynamicBinding {
(2)     public static void main(String[] args) {
(3)             C1 myC1 = new C1();
(4)             myC1.hello();
(5)
(6)             myC1 = new C2();
(7)             myC1.hello();
(8)
(9)             System.out.println("invoking sHello...");
(10)            ((C2) myC1).sHello();
(11)    } //main()
(12)} //class DynamicBinding
(13)
(14)class C1 {
(15)    public void hello() {
(16)            System.out.println("Hello from class C1");
(17)    } //hello()
(18)} //class C1
(19)
(20)class C2 extends C1 {
(21)    public void hello() {
(22)            System.out.println("Hello from class C2");
(23)    } //hello()
(24)
(25)    public void sHello() {
(26)            super.hello();
(27)            hello();
(28)    } //superHello()
(29)} //class C2
```

Beispiel 48: Dynamische Bindung von Methoden DynamicBinding.java

Bildschirmausgabe:

```
$java DynamicBinding
Hello from class C1
Hello from class C2
invoking sHello...
Hello from class C1
Hello from class C2
```

(Relevanter Teil der Ausgabe: oberhalb von *invoking sHello*) Trotz der Typisierung der Variable myC1 als C1-Ausprägung wird die Methode hello von C2 ausgeführt, wenn der Inhalt der Variable auf ein solches Objekt verweist.

- Das Schlüsselwort super dient als Platzhalter zur dynamischen Referenzierung des Superklassenobjekts.
  Das vorhergehende Beispiel zeigt die Verwendung in der Methode sHello, die zunächst auf dem Superklassenobjekt die Methode hello ausführt, und im Anschluß die gleichnamige auf dem aktuellen Objekt.
  Anmerkung: ein Zugriff auf die *Großelternklasse*, d.h. über mehrere Hierarchiebene hinweg -- etwa durch Konkatenation von super zu super.super.foo() --, ist nicht möglich.
- Innerhalb von Konstruktoren, und dort nur als erstes Statement, steht die Surrogatmethode super(...) zur Verfügung, durch welche der entsprechende Superklassenkonstruktor aufgerufen werden kann. Wird der Konstruktor der Superklasse nicht explizit aufgerufen, so erfolgt autmatisch ein Aufruf des parameterlosen Standardkonstruktors der Superklasse.
- Destruktorenaufrufe werden (analog C++) nicht an die Superklasse propagiert.

```
(1)public class ConstDestProp {
(2)     public static void main(String[] args) {
(3)             System.out.println("first creation...");
(4)             new C2();
(5)             System.gc();
(6)
(7)             System.out.println("second creation...");
(8)             new C2(42);
(9)
(10)            System.out.println("third creation...");
(11)            new C2(3.14);
(12)    } //main()
(13)} //class ConstDestProp
(14)
(15)class C1 {
(16)    public C1(int i) {
(17)            System.out.println("constructor of C1 exectued with param i="+i);
(18)    } //C2(int)
(19)
(20)    public C1() {
(21)            System.out.println("constructor of C1 exectued");
(22)    } //C1()
(23)
(24)    public void finalize() {
(25)            System.out.println("destructor of C1 executed");
(26)    } //finalize()
(27)} //class C1
(28)
(29)class C2 extends C1 {
(30)    public C2() {
(31)            System.out.println("constructor of C2 exectued");
(32)    } //constructor
(33)
(34)    public C2(int i) {
(35)            super(i++);
(36)            System.out.println("constructor of C2 exectued with param i="+i);
(37)    } //constructor
(38)
(39)    public C2(double d) {
(40)            System.out.println("constructor of C2 exectued with param d="+d);
(41)    } //constructor
(42)
(43)    public void finalize() {
(44)            System.out.println("destructor of C2 executed");
(45)    } //finalize()
(46)} //class C1
```

Beispiel 49: Propagierung von Konstruktoren- und Destrukturenaufrufen ConstDestProp.java

Bildschirmausgabe:

```
$java ConstDestProp
first creation...
constructor of c1 exectued
constructor of c2 exectued
destructor of c2 executed
second creation...
constructor of c1 exectued with param i=42
constructor of c2 exectued with param i=43
third creation...
constructor of c1 exectued
constructor of c2 exectued with param d=3.14
```

Zunächst wird ein Objekt der Klasse `C2` über den Aufruf des parameterlosen Konstrukturs erzeugt. Vor Ausführung des Konstruktors von `C2` wird durch den Übersetzer ein Aufruf an den Superklassenkonstruktor von `c1` generiert.
Bei Entfernung des `C2`-Objektes aus dem Speicher wird dessen Destruktor, nicht jedoch der der Superklasse, aufgerufen.
Die zweite Objekterzeugung nutzt den expliziten parametrisierten Konstruktor `C2(int)`. Innerhalb dieser Methode wird zunächst explizit der Konstruktur identischer Parameterliste der Superklasse explizit per `super`-Schlüsselwort aufgerufen.
Die dritte Sequenz nutzt den zweiten parametrisierten Konstruktor in `C2`. In dessen Rumpf ist jedoch kein expliziter Aufruf eines Superklassenkonstruktors plaziert. Daher wird automatisch ein Aufruf an den parameterlosen Konstruktor der Superklasse erzeugt.

- Methoden identischer Signatur überschreiben das Pendant der Superklasse.
- Sichtbarkeitseinschränkungen an Methoden und Attribute können in der Subklasse gegenüber der Superklasse erweitert werden.
  Beispiel: Superklasse: `protected`, Subklasse: `private` ist möglich.
- **Abstrakte Klassen** werden durch Angabe des Schlüsselwortes `abstract` als nicht instanziierbar gekennzeichnet.
- Anders als in C++ besitzt *jede* Javaklasse eine Superklasse. Ist eine solche nicht explizit angegeben, so setzt der

Compiler, außer für die Klasse `Object` selbst, die API-Klasse `Object` an diese Stelle.
Anmerkung: Gewisse generische Möglichkeiten (z.B. Objektsammlungen oder Collections) werden hierdurch erst eröffnet.

- Die Vererbungssemantik gilt auch für Arrays!
  So kann jeder beliebige Array *T*`[]`in `Object[]` umgewandelt werden (*up cast*).

2.4.10 Schnittstellen

In C++ nicht explizit präsent. Dort wird die Schnittstellensemantik durch *pure virtual classes* (abstrakte Klasse die nur abstrakte Methoden besitzt) nachgebildet.
Hinweis: Trotz der teilweise weit gefaßten Interpretation des Begriffes *Schnittstelle* wurde diese Übersetzung hier für das originalsprachliche *Interface* verwendet.

Java-Schnittstellen versammeln Operationen, d.h. Methodensignaturen ohne eine Implementierung vorzugeben, sowie Konstante.
Zusätzlich können auch Konstanten definiert werden.

Syntax:

```
InterfaceDeclaration:
   InterfaceModifiers_opt interface Identifier
      ExtendsInterfaces_opt InterfaceBody
```

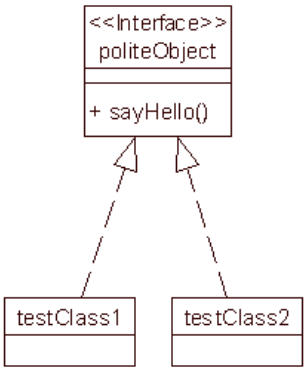
Auch für Schnittstellen stehen die bekannten Modifier `public, protected, private, abstract, static` und `strictfp` zur Verfügung.

Die Sichtbarkeitseinschränkungen sind genauso definiert wie die gleichnamigen Pendants für Klassen.
Konsequenterweise muß ein `public` Interface auch in einer gleichnamigen Quellcodedatei untergebracht werden.
Zwar definiert die Sprachspezifikation (aus Konsistenzgründen zur Definition der Klasse) den **Modifier `abstract`**, dieser ist aber für alle Schnittstellen implizit. Die Sprachspezifikation rät sogar explizit von seiner (verwirrenden) Verwendung ab.

Weiterführende Information: Java Language Specification

Charakteristika:

- Verhalten identisch zu abstrakten Klassen (Vererbung, Substitution, nicht instanziierbarkeit).
- Während des Übersetzerlaufes wird geprüft, ob eine Klasse alle Operationen der Schnittstelle implementiert. Ist dies nicht der Fall, so wird die Deklaration der Klasse als abstrakt erzwungen.
- Auf Ebene der ist Mehrfachvererbung (genauer: Mehrfachimplementierung) zugelassen. Mehrfach geerbte signaturgleiche Operationen werden zu einer einzigen vereinigt.
  Verursacht durch die (möglicherweise) intendierte unterschiedliche Semantik der verschiedenen implementierten Schnittstellen kann dies zu ernsthaften Problemen führen!
- Durch Schnittstellenimplementierung erhält die Klasse als zusätzlichen Typ den (Namen) der Schnittstelle.
- Attribute einer Schnittstelle sind implizit `final static`, und damit als Konstanten, deklariert.
- Implementiert eine Klasse eine Schnittstelle, so vererbt sich dies an alle Subklassen.
- Operationen einer Schnittstelle sind implizit als `abstract`, und damit -- innerhalb der Schnittstelle als -- nicht implementierbar, deklariert.
- Schnittstellen können andere Schnittstellen beerben. Auch hier gilt Einfachvererbung.
- Analog zu den anonymen inneren Klassen können auch anonyme innere Interfaces erstellt werden. Dasselbe gilt für benannte innere Klassen.



```
(1)public class Interface1 {
(2)     public static void main(String[] args) {
(3)             TestClass1 tco1 = new TestClass1();
(4)             TestClass2 tco2 = new TestClass2();
(5)             tco1.sayHello();
(6)             tco2.sayHello();
(7)
(8)             if (tco1 instanceof TestClass1)
(9)                     System.out.println("tco1 is instance of TestClass1");
(10)
(11)            if (tco1 instanceof PoliteObject)
(12)                    System.out.println("tco1 is instance of PoliteObject");
(13)
(14)            PoliteObject po;
(15)            po = tco1;
(16)            po.sayHello();
(17)            po = tco2;
(18)            po.sayHello();
(19)    } //main()
(20)} //class Interface1
(21)
(22)interface PoliteObject {
(23)    public void sayHello();
(24)} //interface PoliteObject
(25)
(26)class TestClass1 implements PoliteObject {
(27)    public void sayHello() {
(28)            System.out.println("Hello World");
(29)    } //sayHello()
(30)} //class TestClass1
(31)
(32)class TestClass2 implements PoliteObject {
(33)    public void sayHello() {
(34)            System.out.println("Guten Tag");
(35)    } //sayHello()
(36)} //class TestClass2
```

Beispiel 50: Verwendung von Schnittstellen Interface1.java

Bildschirmausgabe:

```
$java Interface1
Hello World
Guten Tag
tco1 is instance of TestClass1
tco1 is instance of politeObject
Hello World
Guten Tag
```

Das Beispiel definiert die Schnittstelle `PoliteObject` welche die Operation `sayHello` anbietet. Die beiden Klassen `TestClass1` und `TestClass2` implementieren jeweils Methoden für die durch die Schnittstelle definierten Operationen.
Der erste Anweisungsblock zeigt die (simple) Ausführung der genannten Methode.
Die darauffolgende Codesequenz hebt den Aspekt der Mehrfachtypisierung (Objekt von Schnittstellen-implementierender Klasse ist sowohl Ausprägung der Klasse selbst (im Beispiel: `TestClass1`, als auch der Schnittstelle (`PoliteObject`).
Abschließend wird eine Variable vom Typ der Schnittstelle deklariert, und zunächst mit der Referenz auf ein Objekt der Klasse `TestClass1` belegt. Der (statisch typsicher mögliche) Aufruf `sayHello` ist auch hier möglich. Gleiche Bedingungen herrschen nach der Zuweisung eines Objektes vom Typ `TestClass2` an dieselbe Variable.

Einige **bekannte Schnittstellenverwendungen**:

- Copareable -- wird u.a. von allen numerischen Wrappertypen implementiert.
- Serializable -- dient zur Speicherung (Serialisierung) von Objekten.

2.4.10 Pakete

Pakete stellen eine Sammlung von Klassen und Schnittstellen dar, die Zugriffsschutz und Namensraumverwaltung bietet.
siehe Java Language Specification

Aus dieser Definition heraus sind die Java-Pakete mit den aus C/C++ bekannten Header- und Includedateien nur schwer vergleichbar. Anders als in C/C++ werden diese externen Quellcodesequenzen nicht physisch in den zu übersetzenden Strom eingebunden, sondern existieren weiter und unabhängig. Technisch stellen Pakete ein eigenständiges Sprachmittel dar.

Insbesondere dienen sie nicht zur Separierung von Schnittstellendefinition (.h-Datei) und deren Implementierung.

Syntax einer Paketdefinition:
package *packageName*; am Anfang der Quellcodedatei.
Syntax einer Paketdefinition:
import *packageName.subPackage.subSubPackage...className*;
Hinweis: Im Gegensatz zur C/C++-Präprozessoranweisung include stellt die import-Definition einen syntaktisch korrekten Javaausdruck dar, der durch ein Semikolon abgeschlossen werden muß.
Paketbenennung: Um Pakete weltweit eindeutig identifizieren und unterscheiden zu können hat sich als Konvention ein an die URL-Notation (IETF RFC 1738) anglehntes Namensschema durchgesetzt. Hierbei werden die URL-Komponenten ausgehend von der Toplevel-Domain von rechts nach links angegeben.
Beispiel: Ein Paket testPackageA im Namensraum myCompany.germany.org würde als org.germany.myCompany.testPackageA abgelegt.

Die einzelnen Hierarchiebene der Paketidentifikation werden im JDK durch Kataloge im Dateisystem nachgebildet in denen die Javadateien des entsprechenden Paketes abgelegt werden müssen. So würde die Datei testPackageA.class im Katalog /com/germany/myCompany plaziert (Durch Substitution der Punkte im vollqualifizierten Klassennamen durch den Verzeichnistrenner / ergibt sich der vollqualifizierte Dateipfad.
Das JDK erfodert es, alle Quellcodedateien einer Paketebene gemeinsam zu übersetzen. Compileraufruf javac File1.java File2.java File3.java ...
Weiterführende Information

Charakteristika:

- Eine Javadatei kann maximal ein Paket beinhalten.
- Jede Quellcodedatei kann beliebig viele Pakete importieren.
- Pakete können geschachtelt werden; d.h. ein Paket kann neben Klassen und Schnittstellen auch weitere Pakete beinhalten.
- Klassen und Schnittstellen in Quellcodedateien ohne explizite einleitende Paketdefiniton werden einem unbenannten Standardpaket (*default package*) zugeordnet.
- Alternativ zur Ablage der Paketstruktur im Dateisystem kann der entsprechende Dateibaumast auch zu einer einzigen JAR-Datei (intern ZIP-Komprimierung) zusammengefaßt werden.
- Import kann verschiedengranular angegeben werden:
  - Vollqualifiziert bis auf Klassenebene -- importiert wird (ausschließlich) die spezifizierte Klasse. Der Importvorgang erfolgt erst beim tatsächlichen Zugriff (*import on demand*)
  - Auf beliebiger Ebene endend, mit Stern * terminiert -- importiert werden alle Unterpakete einschließlich der dort definierten Klassen.
- Für Pakete gelten besondere Sichtbarkeitseinschränkungen.
  Besonders hervorzuheben ist die Zugriffsbehandlung nicht public deklarierter Attribute oder Methoden innerhalb eines Paketes. Hier werden alle Elemente desselben Paketes als vertrauenswürdig (engl. *trusted*) angesehen, und erhalten daher auch Zugriff auf als protected deklarierte Information.
- Auf Klassen und Schnittstellen importierter Pakete kann ohne Qualifizierung zugegriffen werden.
- Zum Übersetzungszeitpunkt wird die Namenseindeutigkeit der importierten Elemente geprüft. Liegen mehrdeutige Namensdeklarationen vor, so wird dies durch den Compiler gemeldet.

Hinweise:

- Das Paket java.lang wird durch den Übersetzer automatisch importiert. Daher stehen Methoden wie System.out.println, System.gc ohne vorhergehende import-Anweisung zur Verfügung.
- Inhalte des *Java Extension* Paketes javax sind nicht Bestandteil des offiziellen JDKs und stehen daher oftmals nicht auf allen unterstützten Plattformen zur Verfügung.
- Die Umgebungsvariable CLASSPATH muß zwingend zumindest Dateisystemkataloge der benutzerdefinierten Pakte umfassen. In älteren JDK-Versionen (< v1.3) zusätzlich noch die Systempakete, bzw. die Datei classes.zip.

Beispiele aus der Java-API:

- Übersicht der Pakte
- java.lang, enthält u.a. die Wrappertypen
- java.applet, Grundpaket der Applet-Programmierung
- java.io, Ein-/Ausgabebehandlung
- java.util, u.a. Collection-API

Beispiel:
Quellcodedateien:

```
(1)package testPackage;
(2)
(3)public class TestClass {
(4)     public static void sayHello() {
(5)             System.out.println("hello from TestClass contained in testPackage");
(6)     } //sayHello()
(7)} //class TestClass
```

Beispiel 51: Klasse TestClass innerhalb des Paketes testPackage TestClass.java

```
(1)import testPackage.*;
(2)
(3)public class PackageUser {
(4)     public static void main(String[] args) {
(5)             testClass.sayHello();
(6)     } //main()
(7)} //class PackageUser
```

Beispiel 52: Paketverwendende Datei PackageUser.java PackageUser.java

Dateiplazierung:
`TestClass.java` im Verzeichnis `testPackage`.
`packageUser.java` im logisch direkt übergeordneten Verzeichnis.

Anmerkung:
Der allgemeine Import aller in `testPackage` enthaltenen Klassen, Schnittstellen und Pakete könnte auch per `import testPackage.TestClass;` auf die gewünschte Klasse `TestClass` eingeschänkt werden.
Ebenso würde der vollqualifizierte Methodenaufruf mit `testPackage.TestClass.sayHello();` ohne `import`-Deklaration auskommen.

Statische Imports

Mit der Überarbeitung zur Sprachversion 1.5 wurde durch die *statischen Importe* eine abkürzende Schreibweise für importierte statische Methoden etabliert.
Durch die zusätzliche Angabe des Schlüsselwortes `static` vor der vollqualifizierten Klassenidentifikation stehen als Resultat alle als statisch deklarierten Methoden zum direkten Aufruf, d.h. ohne den Zwang die deklarierende Klasse zu explizieren, zur Verfügung.
Semantisch entspricht die neu geschaffene Importvariante jedoch der bisher vorstellten Mimik. Sie ergänzt diese lediglich um eine Schreibweise die syntaktisch an den Aufruf globaler Funktionen in C/C++ erinnert.
Das nachfolgende Beispiel zeigt die Anwendung beim Aufruf der Methode `sin` der Standardklasse `Math`:

```
(1)import static java.lang.Math.*;
(2)
(3)public class SITest {
(4)     public static void main(String[] atrgs) {
(5)             System.out.println("sin(42)="+ sin(42) );
(6)     } //main()
(7)} //class SITest
```

Beispiel 53: Statischer Import SITest.java

3 Die Java-Plattform

3.1 Die Laufzeitumgebung

3.1.1 Garbage Collection

Wie bereits im Einführungskapitel angeschnitten, verfügt die Javalaufzeitumgebung über eine Speicherverwaltung automatischer Speicherbereinigung (engl. *garbage collection*) des dynamisch verwalteten Heaps.
Generell kann Speicher durch den Programmierer nur konsumiert, nicht jedoch wieder freigegeben werden. (*Zur Erinnerung*: Die explizite Nullzuweisung an eine Objektvariable markiert den dadurch referenzierten Speicherplatz als nicht mehr benötigt, gibt ihn jedoch nicht direkt frei.)
Speicherbereiche werden in Java durch das Java-Schlüsselwort `new`, für Java-Objekte, bzw. durch die explizite Definition von primitiven Datentypen, reserviert. Konkret geschieht die Speicherreservierung auf dem Heap ausschließlich durch die Byte-Code-Instruktionen `new`, `newarray`, `anewarray` und `multianewarray`.
Explizite Freigabemöglichkeiten, wie durch `delete` in C++, existieren nicht.

Zum Begriff *garbage collection*: Die Wortwahl impliziert, daß nicht mehr benötigter Speicher als Abfall behandelt wird, der wegzuwerfen ist. Jedoch implementiert die automatische Speicherbereinigung nicht das intuitiv damit verknüpfte Bild des weggebens...
Vielmehr führt die garbage collection ein *Speicher Recycling* durch, in dessen Verlauf wiederverwendbare Speicherbereiche erkannt, und einer neuen Nutzung zugeführt werden.

Unabhängig von der tatsächlichen Implementierung vollzieht sich die Speicherbereinigung in zwei Schritten:

- Identifikation von nicht mehr erreichbaren Speicherobjekten.
- Freigabe der ermittelten Speicherobjekte.

Als zusätzliche Implementierungsrestriktion tritt unter praktischen Gesichtspunktennoch hinzu, daß der Garbage-Collector-Lauf möglichst wenig (im Idealfalle: keinen) zusätzlichen Speicher beansprucht. Nur so kann das funktionieren auch bei knappem Speicher noch gewährleistet werden.

Für die Aufgabenstellung der automatischen Freispeichergewinnung werden daher sog. *mark and sweep*-Algorithmen eingesetzt. Die zugrundeliegende Vorgehensweise kann wie folgt beschrieben werden: während seiner periodischen Durchläufe markiert der Speicherbereinigungsprozeß alle erreichbaren Speicherobjekte (*mark*-Phase). Nach Analyse aller möglichen Zugriffspfade werden die unmarkieren (da unreferenzierten) Speicherobjekte automatisch freigegeben (*sweep*-Phase).

Während der Mark-Phase werden die Speicherreferenzen verändert, daher wird die Programmausführung zu diesem Zeitpunkt unterbrochen. Als Konsequenz hat die Speicherbereinigung meßbare Auswirkung auf die Ausführungszeit. Die Hauptzeit wird jedoch während der Sweep-Phase verbracht, deren Dauer letztlich von der Größe des zur Verfügung stehenden Arbeitsspeichers abhängt.

Die durch das Mark-and-Sweep-Verfahren bedingte zeitweilige Unterbrechung der Programmausführung ist für interaktive- und Realzeitanwendungen denkbar schlecht geeignet. Daher kommen in der Praxis, so auch in Java, zumeist modifizierte -- aber aufwendigere -- Verfahren zum Einsatz, welche einzelne Speicherobjekte dynamisch sperren. Die Modifikationen setzen an der Erkenntnis an, daß sowohl die Markierungs- als auch die Löschphase inkrementell organisiert sind, d.h. sie betreffen nicht die Gesamtheit der speicherresidenten Objekte. Daher eignen sich beide zu einer kontrolliert parallelen Ausführung, die „lediglich“ dafür Sorge tragen muß, daß die aktuell durch den Garbage Collector im Zugriff befindlichen Speicherstrukturen entsprechend gesperrt sind.

Dieses Verfahren ist der naiven Referenzzählung (separate Tabelle enthält Markierung für jedes Objekt, ob Referenzen darauf existieren) deutlich überlegen, da auch zirkulär verkettete nicht erreichbare Speicherstrukturen als unerreichbar erkannt werden können.

Das nachfolgende Beispiel legt sukzessive verschiedene Speicherstrukturen an, im Folgenden wird der Ablauf des *mark and sweep*-Algorithmus verdeutlicht:

Anmerkung: Im Vorgriff auf die Behandlung der Collection API verwendet die Implementierung der Klasse node die Klasse `HashSet` und die Schnittstelle `Iterator`, auf die in Kapitel 3.2.4 näher eingegangen wird.

```
(1)import java.util.HashSet;
(2)import java.util.Iterator;
(3)
(4)public class GCTest4 {
(5)     static int m1;
(6)
(7)     void aMethod(Node paramNode) {
(8)             Node m2 = new Node("n1");
(9)             Node tmpNode;
(10)
(11)            tmpNode = m2.createChild("n11");
(12)            tmpNode.createChild("n111");
(13)            tmpNode = m2.createChild("n12");
(14)            tmpNode.createChild("n121");
(15)            tmpNode.createChild("n122");
(16)            tmpNode.createChild("n123");
(17)
(18)            Node m3 = new Node("n2");
(19)            tmpNode = m3.createChild("n21");
(20)            tmpNode.appendChild( m3 );
(21)
(22)            System.out.println("output just for testing reasons...");
(23)            System.out.println("name of parameter Node: "+paramNode.getName() );
(24)            System.out.println("name of Node referenced by m2: "+m2.getName() );
(25)            System.out.println("child's names:");
(26)            m2.getChildsNames();
(27)
(28)            System.out.println("name of Node referenced by m3: "+m3.getName() );
(29)            System.out.println("is n21 child of n2? "+m3.isChild("n21") );
(30)            System.out.println("is n2 child of n21? "+tmpNode.isChild("n2") );
(31)
(32)            tmpNode = new Node("n31");
(33)
(34)            ((tmpNode.createChild("n32")).createChild("n33")).appendChild(tmpNode);
(35)
(36)            tmpNode = null;
(37)            System.gc();
(38)    } //aMethod()
(39)
(40)    public static void main(String[] args) {
(41)            (new GCTest4()).aMethod(new Node("n4"));
(42)    } //end main()
(43)} //class GCTest4
```

```
(44)
(45)class Node {
(46)    HashSet childs = new HashSet();
(47)    String name;
(48)
(49)    public Node(String name) {
(50)            this.name = name;
(51)            System.out.println("created Node object named "+this.name);
(52)    } //Node(String)
(53)
(54)    public void appendChild(Node child) {
(55)            childs.add(child);
(56)    } //appendChild(Node)
(57)
(58)    public Node createChild(String name) {
(59)            Node newChild = new Node( name );
(60)            childs.add(newChild);
(61)            return newChild;
(62)    } //createChild(String)
(63)
(64)    public String getName() {
(65)            return name;
(66)    } //getName()
(67)
(68)    public boolean isChild(String name) {
(69)            Iterator childIt = childs.iterator();
(70)            while (childIt.hasNext()) {
(71)                    if ( ((Node) childIt.next()).getName() == name)
(72)                            return true;
(73)            } //while
(74)            return false;
(75)    } //isChild(String)
(76)
(77)    public void getChildsNames() {
(78)            Node child;
(79)            Iterator childIt = childs.iterator();
(80)            while (childIt.hasNext()) {
(81)                    child = (Node) childIt.next(); //explicit (down) type cast needed since
HashMap contains only Object instances
(82)                    System.out.println( child.getName() );
(83)                    child.getChildsNames();
(84)            } //while
(85)    } //getChildsNames()
(86)    protected void finalize() {
(87)            System.out.println("Node object named "+name+" freed!");
(88)    } //finalize()
(89)} //class Node
```

Beispiel 54: verschiedene Speicherstrukturen GCTest4.java

Bildschirmausgabe:

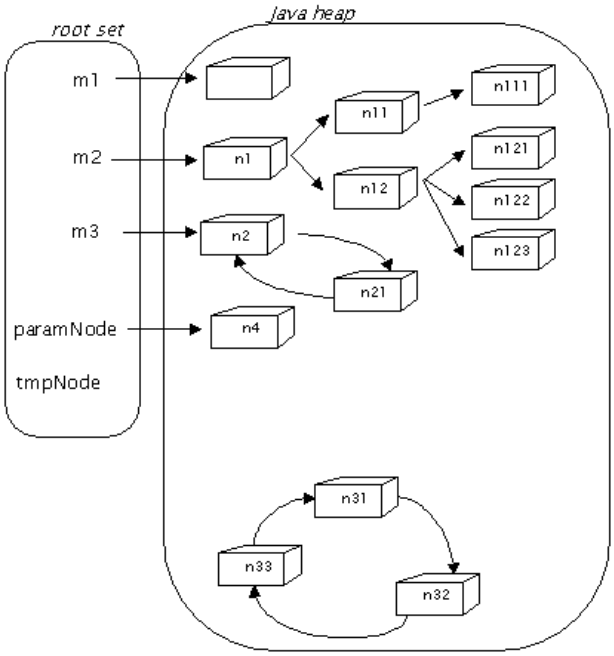
```
$java GCTest4
created node object named n4
created node object named n1
created node object named n11
created node object named n111
created node object named n12
created node object named n121
created node object named n122
created node object named n123
created node object named n2
created node object named n21
output just for testing reasons...
name of parameter node: n4
name of node referenced by m2: n1
child's names:
n11
n111
n12
n121
n122
n123
name of node referenced by m3: n2
is n21 child of n2? true
is n2 child of n21? true
created node object named n31
created node object named n32
created node object named n33
node object named n31 freed!
```

```
node object named n32 freed!
node object named n33 freed!
```

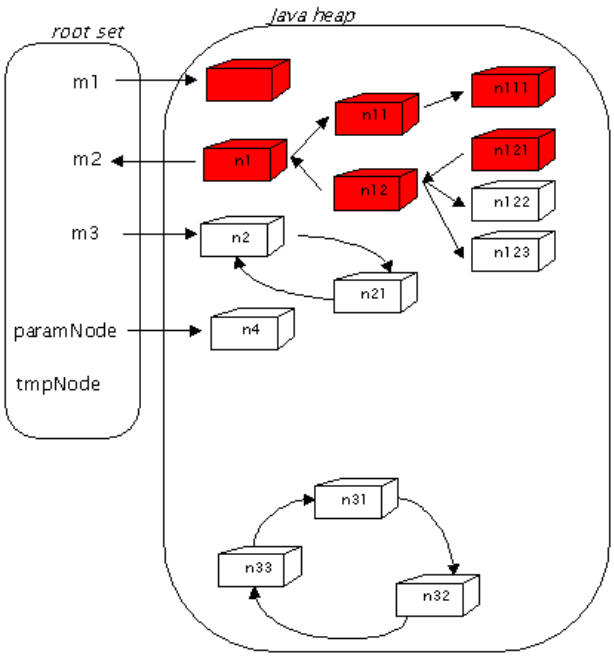
Ablauf des *mark and sweep*-Verfahrens:
Zunächst werden die Referenzen aller im Gültigkeitsbereich der aktuell ausgeführten Methode verfolgt. Im Einzelnen sind dies:

- Übergabe-Argumente
- lokale Variablen
- statische Variablen (der geladenen Klassen)
- Referenzen die durch das Java Native Interface (JNI) registriert sind

Die Menge dieser Speicherreferenzen wird als *root set* bezeichnet.



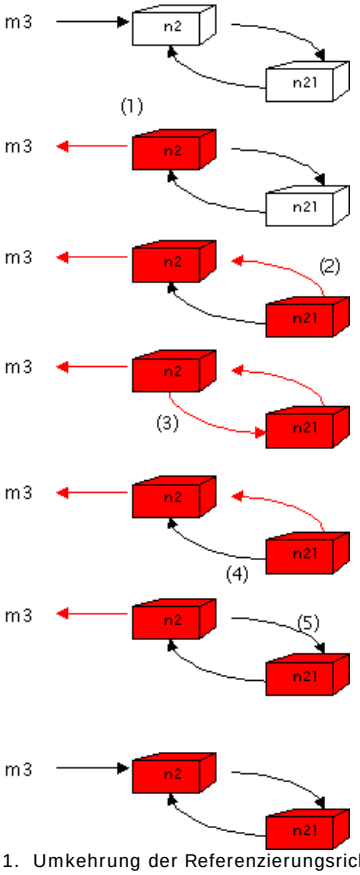
Mark-Phase:
Der Speicherbereinigungsprozeß durchläuft ausgehend von den Elementen des *root set*s , mit dem Ziel die erreichbaren zu markieren. Die sich zunächst intuitiv aufdrängende Vorgehensweise der rekursiven Baumtraversierung verbietet sich, wenn die Restriktion daß der Speicherbereinigungsprozeß nur minimale eigene Speicheranforderungen stellt berücksichtigt werden soll. (Darüberhinaus ergibt sich noch ein weit schwerwiegenderes Problem bei zyklischen Strukturen, wie wir in der Folge noch sehen werden...).
Daher wird die verzeigerte Speicherstruktur iterativ, unter Abhänderung der Zeigerstruktur, durchlaufen. Da der Rekursionsstack als *Gedächtnis* des genommenen Weges durch den Baum nicht zur Verfügung steht, werden die *Vorwärts*-Zeiger sukzessive zu *Rückkehr*-Zeigern modifizert. Dies vollzieht sich in zwei Schritten. Zunächst wird der referenzierte Knoten ermittelt und zwischengespeichert. Dann wird dieser besucht und markiert; und die Zeigerrichtung invertiert. Ist ein Blattknoten erreicht und markiert, so werden die veränderten Zeigerstrukturen zur Rückkehr benutzt (sie zeigen jetzt auf den jeweils hierarchisch übergeordneten Knoten). Während des Aufsteigens werden die Zeicherstrukturen nochmals invertiert, d.h. wieder in ihren ursprünglichen Zustand (zurück) versetzt.



Einen Sonderfall, der bei rekursiver Implementierung aufwendig zu behandeln wäre, stellen zirkuläre Strukturen dar. Hierbei handelt es sich allgemein um Zykel beliebiger Länge. Das bedeutet, nach einer gewissen Anzahl von

Speicherknoten existiert ein Verweis (zurück) auf einen bereits traversierten Knoten. Als praktisches Beispiel solcher Strukturen seien zyklisch verkettete Listen angeführt.
Solche Strukturen sind in zweierlei Hinsicht bemerkenswert. Zum Einen, stellen sie hinsichtlich effizienter Traversierung eine Herausforderung dar, zum Anderen, da unabhängige Zykel (siehe n31, n32 und n33 im Beispiel) nicht erreichbare, aber gültig referenzierte Strukturen sind.

Die Graphik zeigt schrittweise die Traversierung und Markierung der einzelnen Knoten eines Zykels der Länge 2.



1. Umkehrung der Referenzierungsrichtung zwischen der Anwendervariable `m3` (Element des *root set*) und dem als `n2` benannten Speicherobjekt.
   `n2` wird markiert.
2. Weiternavigation, durch Verfolgung der Referenz von `n2` zum Speicherobjekt `n21`.
   Umkehrung der Referenz und Markierung des Knotens `n21`.
3. Verfolgung der Referenz von `n21` zurück zu `n2`.
   Neuer Knoten (`n2`) ist bereits markiert. (Implizit ist ein Zyklus erkannt worden!)
   Damit sind wir zwangsläufig am Ende eines Traversierungsastes angelangt, da auf einen markierten Knoten ausschließlich markierte folgen, sofern die Hierarchie absteigent durchlaufen wird.
4. Verfolgung der Referenz zurück zum Ursprungsknoten.
   Anschließend: Invertierung der Verzeigerung; stellt Ursprungszustand wieder her.
5. dto.
6. dto.

Eine Abwandlung des vorhergehenden Falles zyklischer Strukturen stellt die auf `n31`, `n32` und `n33` gebildete Struktur dar.
Obwohl auf alle drei erzeugten Objekte gültige Referenzen existieren (`n31` zeigt auf `n32`, `n32` auf `n33`, das wiederum auf `n31` verweist) sind die Objekte nach Neuzuweisung (null-Setzung) an `tmpNode` allesamt nicht mehr erreichbar.

Sweep-Phase:
Sie dominiert den Speicherbereinigungslauf zeitlich. Während die Markierungen vergleichsweise schnell und effizient -- d.h. unter Vermeidung unnötiger Besuchsschritte -- angebracht werden können, wird in der zweiten Phase der gesamte Heap durchlaufen. Dabei wird jedes Speicherobjekt betrachtet, unabhängig davon ob es markiert ist oder nicht.
Nichtmarkierte Speicherobjekte werden in den Freispeicher eingereiht.

Hinweis: Die implementierungsspezifischen Aussagen beziehen sich auf SUNs Java2 (JDK v1.3) Umsetzung.

Technisch ist der Java Garbage Collector eigenständiger niederpriorer Thread innerhalb der virtuellen Maschine ausgelegt. Situations- und plattformabhängig ist dieser Thread synchron oder asynchron realisiert. Im Falle knappen Speichers, oder expliziter Anforderung durch das Programm, läuft er synchron.
Vor der Freigabe des Speicherplatzes eines Objekts wird dessen Destruktormethode aufgerufen.
Inkrementelle Speicherplatzbereinigung (auf Klassenebene) kann für die virtuelle Maschine der Fa. SUN durch die Kommandozeilenoption `-Xincgc` erzwungen werden.

Der Garbage Collector kann nicht explizit aufgerufen werden. Jedoch erwirkt der Aufruf System.gc() den Versuch zur Speicherbereinigung. Nach Rückkehr der Methode muß eine Speicherbereinigung nicht zwingend erfolgt sein.
Für Objekte die bereits zur Entfernung aus dem Speicher ausgewählt wurden, jedoch die Abarbeitung ihrer Destruktoren noch aussteht, kann der Destruktlauf mit System.runFinalization() angestoßen werden.

Die Javalaufzeitumgebung von SUN java erlaubt den Eingriff in die Standardgarbagecollection über folgende Kommandozeilenschalter:

- verbose:gc
  Schaltet die Protokollierung der Garbage Collector-Aktivitäten auf Stdout ein.
- Xnoclassgc
  Unterbindet Speicherbereinigung für Klassen.
- Xincgc
  Aktiviert die inkrementelle Garbage Collection. Hierdurch werden zunächst die (durch Anhalten der Ausführung) spürbaren Speicherbereinigungsläufe zugunsten einer permanenten Ausführung unterbunden. Jedoch geht die Applikationsperformanz um ca. 10% zurück.

```
(1)public class GCTest1 {
(2)     public static void main(String[] args) {
(3)             System.out.println("memory before: "+Runtime.getRuntime().freeMemory() );
(4)             Test1 t1Obj = new Test1();
(5)             System.out.println("memory after: "+Runtime.getRuntime().freeMemory() );
(6)             t1Obj = null;
(7)             System.gc();
(8)             System.out.println("memory after garbage collection: "+ Runtime.getRuntime().
freeMemory() );
(9)     } //main()
(10)} //class GCTest1
(11)
(12)class Test1 {
(13)    double testArray[] = new double[100];
(14)    public void finalize() {
(15)            System.out.println("object of class Test1 freed");
(16)    } //finalize()
(17)} //class Test1
```

Beispiel 55: Speicherverbrauch vor und nach Garbage Collection GCTest1.java

Bildschirmausgabe:

```
$java -verbose:gc GCTest1
memory before: 1814608
memory after: 1809824
[Full GC 216K->112K(1984K), 0.0344306 secs]
object of class test1 freed
memory after garbage collection: 1915920
```

Zunächst wird der aktuelle freie Speicher innhalb der virtuellen Maschine mittels freeMemory() abgefragt (genaugenommen liefert die Methode eine Schätzung des freien Speichers in Byte). Das erzeugte Objekt initialisiert einen double-Array mit 100 Werten. Hieraus errechnet sich ein minimaler Speicherplatzbedarf von 800 Byte für das Objekt t1Obj. Der im Beispiel ermittelte Wert von 4520 Byte wird durch weitere Effekte wie Verwaltungsstrukturen und sonstige Laufzeitinformation verursacht.
Nach Nullzuweisung und explizitem Aufruf der Garbage Collectors mit System.gc() vergrößert sich der freie Speicher wieder. Er nimmt sogar über den Startwert zu. Dies ist der Freigabe von Speicherobjekten geschuldet, die nicht durch den Anwender, sondern durch die virtuelle Maschine selbst erzeugt wurden. Einen Eindruck der, üblicherweise verdeckt, automatisch geladenen Kompontenten liefert der Kommandozeilenschalter verbose:class der der Java-Ausführungsumgebung übergeben werden kann.

Die Realisierung des Garbage Collectors von SUN erhebt nicht den Anspruch in jedem Falle Speichplatz-optimal vorzugehen, d.h. alle potentiell unreferenzierten Objekte zu sofort entdecken und freizugeben. Dies läßt sich in einer Modifikation des obigen Beispiels zeigen:

```
(1)public class GCTest2 {
(2)     public static void main(String[] args) {
(3)             System.out.println("memory before: "+Runtime.getRuntime().freeMemory() );
(4)             Test1 t1Obj = new Test1();
(5)             t1Obj = null;
(6)
(7)             for (int i=0; i<10; i++) {
(8)                     System.gc();
(9)                     System.out.println("memory after garbage collection: "+ Runtime.getRuntime
().freeMemory() );
(10)            } //for
(11)    } //main()
(12)} //class GCTest2
(13)
(14)class Test1 {
(15)    double testArray[] = new double[100];
(16)    public void finalize()  {
(17)            System.out.println("object of class Test1 freed");
(18)    } //finalize()
(19)} //class Test1
```

Beispiel 56: Mehrmaliger Garbage Collectorlauf GCTest2.java

Bildschirmausgabe:

```
$java -verbose:gc gcTest2
memory before: 1814640
[Full GC 216K->112K(1984K), 0.0336478 secs]
object of class test1 freed
memory after garbage collection: 1915920
[Full GC 113K->112K(1984K), 0.0382289 secs]
memory after garbage collection: 1915952
[Full GC 113K->112K(1984K), 0.0306936 secs]
memory after garbage collection: 1915952
[Full GC 113K->111K(1984K), 0.0392094 secs]
memory after garbage collection: 1917008
[Full GC 112K->111K(1984K), 0.0315576 secs]
memory after garbage collection: 1917008
[Full GC 112K->111K(1984K), 0.0305033 secs]
memory after garbage collection: 1917008
[Full GC 112K->111K(1984K), 0.0313825 secs]
memory after garbage collection: 1917008
[Full GC 112K->111K(1984K), 0.0304195 secs]
memory after garbage collection: 1917008
[Full GC 112K->111K(1984K), 0.0319465 secs]
memory after garbage collection: 1917008
[Full GC 112K->111K(1984K), 0.0361018 secs]
memory after garbage collection: 1917008
```

So wird ein weiterer Speicherblock, trotz der bereits erfolgten Entfernung des Objektes aus dem Speicher (siehe Ausgabe des Destruktors), der Größe 193 Bytes erst durch den vierten Garbage Collector Lauf freigegeben.

Auswirkungen auf die Programmierung:
A priori hat das Vorhandensein eines automatischen Speicherbereinigungsmechanismus keine Auswirkungen auf die Algorithmenentwicklung oder -umsetzung. Jedoch können Maßnahmen zur expliziten Freigabe nicht mehr benötigter Speicherbereiche durch den Anwendungsprogrammierer (oftmals) unterbleiben, da das Laufzeitsystem sich um die Freigabe nicht mehr erreichbarer Speicherzellen kümmert. Dies kann beispielsweise bei der Implementierung verketteter Datenstrukturen (Listen, Bäume, etc.) hilfreich sein.
Die entstehenden Algorithmen sind jedoch dann nicht mehr adaptionsfrei auf Systeme ohne garbage collection übertragbar.

Abschlußbemerkungen:

- In Einzelfällen kann der berechtigte Wunsch entstehen, Objekte freizugeben, auf die noch Referenzen existieren; beispielsweise wenn ein Objekt vollständig aus dem Speicher entfernt werden soll, obwohl es noch innerhalb von Objektsammlungen (Listen, Arrays, etc.) referenziert wird. Hierfür wurden in Java v1.2 sog. *schwache Referenzen* geschaffen, die es erlauben ausschließlich schwach referenzierte Objekte freizugeben. Implementiert ist dieses Verhalten, das so auch in anderen Programmiersprachen, wie z.B. SmallTalk verwirklicht ist, im Paket `java.lang.ref`.
- Die technische Umsetzung der automatischen Speicherbereinigung (asynchroner Thread, sub-optimales Freigabeverhalten) bedingt insgesamt ein nichtdeterministisches Anwendungsverhalten hinsichtlich Speicherplatzbedarf der Applikation, gepaart mit einem nichtdeterministisches Laufzeitverhalten durch nicht-planbare und -vorhersehbare Aktivität des Garbage Collectors. Dies läßt die Verwendung von Java, in der Standardversion, für harte Realzeitanwendungen nicht zu. Daher haben sich in der zwischenzeit einige Aktivitäten zur Modifikation von Java für ebendiese Anwendungsdomänen formiert.

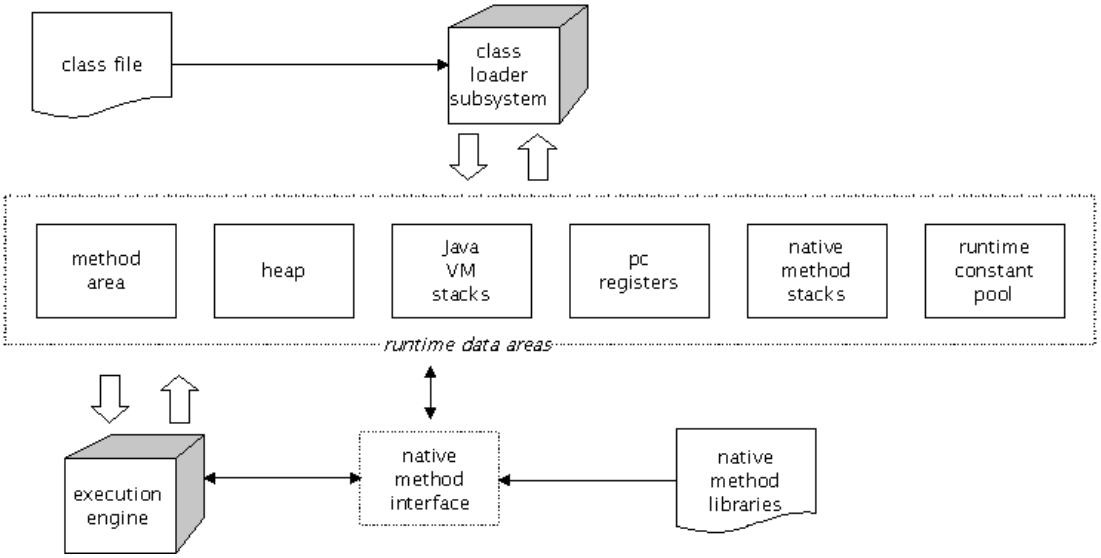
3.1.2 Virtuelle Maschine

Kern der oft apostrophierten Plattformunabhängigkeit der Programmiersprache Java ist die Generierung eines generischen Zwischenformates -- des Byte-Codes. Dieser wird von einer plattformabhängig implementierten Programmeinheit, der *Java Virtual Machine* (Abk. JVM) interpretativ zur Ausführung gebracht.
Jede Java-Applikation wird auf einer eigenen virtuellen Maschine zur Ausführung gebracht. Dies garantiert eine größtmögliche Abschottung, mit dem Ziel maximierter Sicherheit, der möglicherweise gleichzeitig auf einer realen Maschine ausgeführten Java-Programme voneinander.
Das Konzept der virtuellen Maschine, die als Programm auf einer realen Hardware abläuft, ermöglicht eine vergleichsweise einfache und schnelle Portierbarkeit auf neue Zielumgebungen, da lediglich die virtuelle Maschine an die veränderte reale Maschine angepaßt werden muß.

Der Gedanke virtueller Maschinen, die generischen Zwischencode -- oftmals auch als *P-Code* bezeichnet -- ausführen, ist nicht neu. Bereits USCD Pascal, E-BASIC und die verschiedenen SmallTalk-Implementierungen, setzt diesen praktisch um.
Die Realisierung der Befehlsfolgen (Opcodes) innerhalb der virtuellen Maschine von Java ähnelt teilweise frappant der Architektur der für die Züricher Pascal-Implementierung entwickelten (abstrakten) P-Maschine.

Inzwischen steht mit der zAAP (zSeries Application Assist Processor)-Hardware für die IBM-Mainframemaschinen *z890* und *z990* sogar eine vollständige Hardwareimplementierung der JVM zur Verfügung, welche den Charakter der *virtuellen* zu einer *realen* Maschine weiterentwickelt.

Ein Beispiel einer vollständig als Softwar realisierten „klassischen“ virtuellen Maschine ist `java`, die Bestandteil des Java-Development Toolkits von SUN ist.
Bekannte andere virtuelle Maschinen sind: Kaffe oder auch IBMs *Jikes*-Implementierung
Wie bereits bekannt wird eine Java-Applikation durch den Aufruf `java`, gefolgt vom Namen der Startklasse und etwaiger Kommandozeilenparameter ausgeführt. Technisch gesehen bewirkt der Aufruf zunächt die Erzeugung einer neuen Instanz der virtuellen Maschine, auf welcher die Programm-Abarbeitbung mit der `main`-Methode der Startklasse begonnen wird.
Eine Instanz einer virtuellen Maschine existiert, solange Programmfäden (engl. *Threads*) (genaugenommen: non-deamon Threads) ausgeführt werden, bzw. die virtuelle Maschine explizit beendet wird (mit dem API-Aufruf `System.exit()`) oder ein Fehler auftritt.



Die wesentlichen Bestandteile der JVM sind:

- *method area*
  Einmal vorhanden je virtueller Maschine, wird gemeinsam von allen Threads benutzt; enthält klassenspezifische Daten (Typinformation).
- *heap*
  Einmal vorhanden je virtueller Maschine, wird gemeinsam von allen Threads benutzt; enthält die dynamisch erzeugten Objekte.
- *Java VM stacks*
  Threadspezifisch. Enthält Zustand von nicht-nativen Methodenaufrufen, deren lokale Variablen, Übergabeparameter, den möglichen Rückgabewert sowie Methoden-interne Berechnungsergebnisse. (vgl. JVM-Spezifikation)
- *pc registers*
  Threadspezifisch; enthält für jeden Zeitpunkt der Ausführung einer nicht-nativen Methode die Adresse der derzeit ausgeführten Instruktion. (vgl. JVM-Spezifikation). Während der Abarbeitung nativer Methoden ist der Wert auf `undefined` gesetzt. Konkrete Größe des virtuellen pc-Registers hängt von der Adresslänge der realen Plattform ab.
- *native method stack*
  Implementierungsspezifischer Speicherbereich zur Behandlung von nativen Methodenaufrufen.
- *runtime constant pool*
  Klassen- oder Schnittstellenspezifisch. Enthält verschiedene Konstanten, die zur Übersetzungszeit feststehen und in der `constant_pool` Tabelle der class-Datei abgelegt sind. Die Funktion dieses Bereichs ähnelt dem einer konventionellen Symboltabelle. (vgl. JVM-Spezifikation)

Die Java-Stacks sind in *stack frames* organisert. Jedem Methodenaufruf ist ein Stack-Frame zugeordnet, der beim Aufruf erzeugt (push), und beim Verlassen (pop) vom Stack entnommen wird.
Innerhalb eines Frames befindet sich

- Array der lokalen Variablen
- Operanden-Stack
  Wichtigste Datenstruktur innerhalb der virtuellen Maschine. Der Operanden-Stack enthält alle Eingangs- und Ausgangsdaten der verschiedenen Berechnungen. Seine Einträge sind typisierten in 32-Bit Worten organisiert; die korrekte Typisierung der Eingangoperanden wird von jedem Opcode geprüft.
- Referenz auf runtime constant pool
- implementierungsspezifische- und debugging-Information

Den für den Anwendungsentwickler offensichtlichsten Bestandteil der virtuellen Maschine, bilden jedoch die JVM-Instruktionen -- die Maschinensprache der JVM.
Der Befehlssatz der JVM umfaßt ausschließlich genau ein Byte lange Opcodes.
Die JVM ist generell *stack-orientiert*. Dies bedeutet, daß Quell- und Zieloperanden der meisten Operationen werden vom Stack entnommen, und das Ergebnis dort abgelegt. Insbesondere existieren, abgesehen von vier Verwaltungsspeicherplätzen je Ausführungs-Thread, keine virtuellen Prozessorregister, um die Implementierungsanforderungen an die reale Plattform zu minimieren.

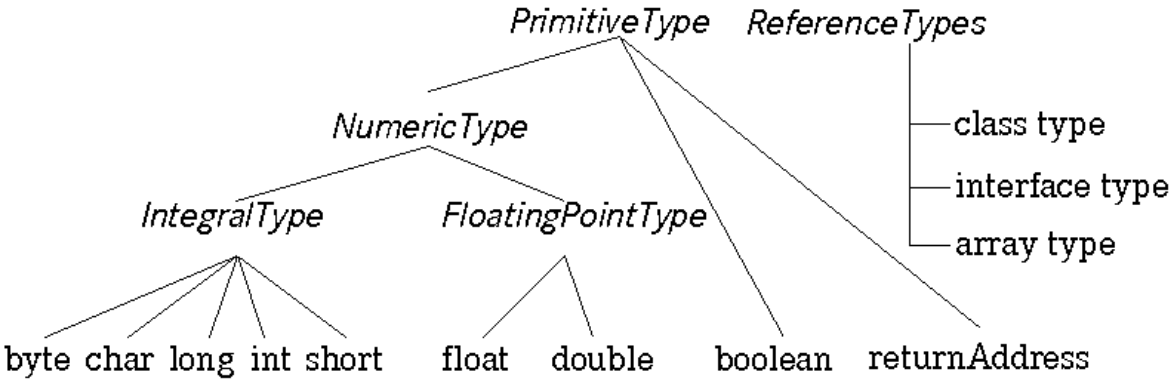
Als threadlokale Register stehen zur Verfügung:

- pc -- program counter
  Enthält die Adresse der gerade ausgeführten Instruktion.
  Handelt es sich um eine native-Methode, die in einer anderen Programmiersprache ausgeführt wird, so ist der Wert *undefined*.

- optop
  Verweist auf die Spitze des Operanden Stacks.
- frame
  Verweist auf die Ausführungsumgebung der derzeit aktiven Methode.
- vars
  Ein Zeiger auf die erste lokale Variable der aktuellen Methode.

Die Adresslänge innerhalb der JVM ist auf vier Byte (32 Bit) fixiert. Hieraus ergibt sich ein (theoretischer) Adressraum von 4 GB.
Die initiale und maximale Ausdehnung des Heaps kann durch die Kommandozeilenschalter `Xms` bzw. `Xmx` gesteuert werden (Beispiel: `java -Xms350M -Xmx500M HelloWorld` führt ein einfaches Hello-World-Beispiel mit einer anfänglichen Speicherausstattung von 350 MB aus, die im Verlaufe des Programmablaufs auf höchstens 500 MB anwachsen kann.)

Das **Typsystem der JVM** lehnt sich eng an das der Hochsprache an. (Zur Erinnerung: primitive Typen in Java)
Zusätzlich erweitert es die Primitivtypen um einen Adresstypen `returnAddress` und führt explizite Referenztypen auf die verschiedenen high-level Typen (Klassen, Schnittstellen, Arrays) ein.



Datentypen der JVM:

- byte
  Vorzeichenbehaftete 8-Bit Zahl in Zweierkomplementdarstellung.
- short
  Vorzeichenbehaftete 16-Bit Zahl in Zweierkomplementdarstellung.
- int
  Vorzeichenbehaftete 32-Bit Zahl in Zweierkomplementdarstellung.
- long
  Vorzeichenbehaftete 64-Bit Zahl in Zweierkomplementdarstellung.
- char
  Vorzeichenbehaftete 16-Bit Zahl zur Darstellung von Unicode-Zeichen.
- float
  32-Bit Binärdarstellung einer Fließkommazahl gemäß IEEE 754-1985.
- double
  64-Bit Binärdarstellung einer Fließkommazahl gemäß IEEE 754-1985.
- boolean
  Wird zwar durch die JVM definiert, jedoch existieren keine Opcodes, die Parameter dieses Typs erwarten. Der Compiler setzt Anweisungen die boolean-Typen enthalten als Operationen auf int-Typen um.
- returnAddress
  Für den Programmierer nicht zugänglicher Typ, der Sprungadressen aufnimmt.
- Referenztypen
  Verweisen auf Klassen, Arrays oder Schnittstellen.
  Der Wert kann auch null sein, wobei die JVM keine konkrete Darstellung dieses Wertes unterstellt.

Jede Bytecode-Instruktion besteht zunächst aus ihrem Opcode, optional gefolgt von den benötigten Operanden. Diese stehen jedoch nicht für sich, sondern sind eingebettet in den organisatorischen Rahmen der `class`-Datei, deren Format im Anschluß vorgestellt wird.

Die verschiedenen Maschineninstruktionen lassen sich in Klassen einteilen:

Instruktionen zum Zugriff auf lokale Variablen:

Instruktion	Funktion
ret	Rücksprung aus Subroutine, wird in der Implementierung von finally benutzt
aload	Lädt Referenz von spezifisch indizierter Position auf den Operanden-Stack
aload <n>	Lädt Referenz auf Operanden-Stack (Index im Opcode explizit angegeben)
astore	Speichert auf Operanden Stack liegenden Wert in lokale Variable
astore <n>	Legt Referenz in lokaler Variable auf Operanden-Stack ab (Index wird im Opcode explizit angegeben)
dload	Lädt double-Wert aus lokaler Variable auf den Operanden-Stack

dstore	Lädt double-Wert vom Operanden-Stack und legt ihn in lokaler Variable ab
fload	Lädt float-Wert aus lokaler Variable auf den Operanden-Stack
fstore	Lädt float-Wert vom Operanden-Stack und legt ihn in lokaler Variable ab
iload	Lädt int auf Operanden-Stack (Index im Opcode explizit angegeben)
iload_<*n*>	Lädt int-Wert aus lokaler Variable auf den Operanden-Stack
istore	Legt int-Wert von spezifisch indizierter Position in lokaler Variable auf Operanden-Stack ab
istore <*n*>	Lädt int-Wert vom Operanden-Stack und legt ihn in lokaler Variable ab
lload	Lädt long-Wert aus lokaler Variable auf den Operanden-Stack
lstore	Lädt long-Wert vom Operanden-Stack und legt ihn in lokaler Variable ab
iinc	Inkrementiert lokale Variable um fixe int-Zahl

Instruktionen zur expliziten Modifikation des Operanden-Stacks:

Instruktion	Funktion
bipush	Ablegen eines byte-Wertes auf dem Operanden-Stack
sipush	Ablegen eines short-Wertes auf dem Operanden-Stack
pop	Entnimmt und verwirft (de facto: löscht) obersten Operanden-Stack-Eintrag.
pop2	Entnimmt (de facto: löscht) obersten beiden Operanden-Stack-Einträge.
swap	Tauscht die beiden obersten Operanden-Stack-Einträge aus.
ldc	Ablegen eines Elements (referenziert über 16-Bit Index) des runtime constant pools auf dem Operanden-Stack
ldc w	Ablegen eines Elements (referenziert über 32-Bit Index) des runtime constant pools auf dem Operanden-Stack
aconst_null	Legt null auf dem Operanden-Stack ab.
dconst_<*d*>	Legt double Konstante auf dem Operanden-Stack ab. dconst_0 die Konstante 0.0, bzw. dconst_1 den Wert 1.0.
fconst_<*f*>	Legt float Konstante auf dem Operanden-Stack ab. fconst_0 die Konstante 0.0, bzw. fconst_1 den Wert 1.0.
iconst_<*i*>	Legt int-Konstante auf dem Operanden-Stack ab. Es existieren Opcodes für folgende Konstanten: iconst_m1 -- -1; iconst_0 -- 0; iconst_1 -- 1; iconst_2 -- 2; iconst_3 -- 3; iconst_4 -- 4; iconst_5 --5.
dup	Dupliziert oberstes Element des Operanden-Stacks. dup_x1 legt das neue Element als zweitunterstes auf dem Stack ab. dup_x2 legt das neue Element, abhängig vom Stack-Inhalt, als zweit- oder drittunterstes auf dem Stack ab.
dup2	Dupliziert die beiden obersten Elemente des Operanden-Stacks. dup2_x1 legt die neuen Elemente als zweitunterstes und folgendes auf dem Stack ab. dup2_x2 legt die neuen Elemente, abhängig vom Stack-Inhalt, als zweit- oder drittunterstes und folgendes auf dem Stack ab.
lconst_<*l*>	Legt long Konstante auf dem Operanden-Stack ab. Es existieren Opcodes für folgende Konstanten: iconst_0 -- 0; iconst_1 -- 1.

Instruktion zur Steuerung des Kontrollflußes:

Instruktion	Funktion
goto	Bedingungsloser Sprung innerhalb derselben Methode. Der anzugebende *branch offset* ist auf 16-Bit fixiert, woraus sich ableiten läßt, daß das Opcodesegment einer Methode niemals (in der JVM-Version 1.3) die Größe von 64KByte überschreiten darf. Näheres zu den Einschränkungen der virtuellen Maschine findet sich in im Abschnitt 4.10 der JVM-Spezifikation, sowie in der Diskussion des Class-File Formats.
goto_w	Bedingungsloser Sprung innerhalb derselben Methode (mit 32-Bit Offset)
if_acmp<*cond*>	Bedingter Sprung im Falle der Gültigkeit der Bedingung. Die zu vergleichenden Operanden werden als Referenzen übergeben. Als Bedingungen stehen Gleichheit (eq) und Ungleichheit (ne) zur Verfügung.
if_icmp<*cond*>	Bedingter Sprung im Falle der Gültigkeit der Bedingung. Die zu vergleichenden Operanden werden als int-Werte übergeben. Als Bedinungen stehen zur Verfügung: Gleichheit (eq), Ungleichheit (ne), Kleiner (lt), Kleiner oder Gleich (le), Größer (gt) und Größer oder Gleich (ge).
if<*cond*>	Bedingter Sprung, nach Vergleich des obersten Elements des Operanden-Stacks mit Null Für spezifische Vergleiche stehen folgende Opcodes zur Verfügung: Geleichheit (ifeq), Ungleichheit (ifne), Kleiner (iflt), Kleiner oder Gleich (ifle), Größer (ifgt), Größer oder Gleich (ifge).
ifnonnull	Bedingter Sprung -- im Falle der Ungleichheit -- nach Vergleich der auf dem Operanden-Stack befindlichen Referenz mit Null

ifnull	Bedingter Sprung -- im Falle der Gleichheit -- nach Vergleich der auf dem Operanden-Stack befindlichen Referenz mit Null
jsr	Unbedingter Sprung, unter Sicherung der Rücksprungadresse auf dem Operanden-Stack, zu 16-Bit Adresse.
jsr_w	Unbedingter Sprung, unter Sicherung der Rücksprungadresse auf dem Operanden-Stack, zu 32-Bit Adresse.
lookupswitch	Zugriff auf Sprungtabelle per Schlüssel und anschließende Verzweigung. Benutzt zur Implementierung des switch-Konstrukts
tableswitch	Indexbasierter Zugriff auf Sprungtabelle und anschließende Verzweigung.

Instruktionen zur Operation auf Klassen und Objekten:

Instruktion	Funktion
anewarray	Array-Erzeugung an definierter Stelle im Laufzeit-Konstanten-Pool.
checkcast	Typkompatibilitätsprüfung (siehe Beispiel)
instanceof	Prüft ob Objekt gegebenen Typ hat (d.h. Ausprägung der Klasse -- oder einer Subklasse -- ist; das Interface implementiert; Array-Kompatibel ist)
new	Erzeugt neues Objekt *Hinweis*: Die Punktnotation zur Trennung der Pakethierarchien wird hier durch Slash „/“ ersetzt.

Instruktionen zur Methodenausführung:

Instruktion	Funktion
invokespecial	Ruft Instanzenmethode auf; mit besonderer Behandlung bestimmter Umstände. *Hinweis*: Diese Instruktion wurde umbenannt, frühere JDK-Versionen benutzen invokeonvirtual.
invokestatic	Ruft statische Klassenmethode auf.
invokevirtual	Ruft Instanzenmethode auf.
invokeinterface	Ruft Schnittstellenmethode auf.
areturn	Retourniert Referenz auf Speicherobjekt nach Methodenausführung.
dreturn	Retourniert double-Wert nach Methodenausführung.
freturn	Retourniert float-Wert nach Methodenausführung.
ireturn	Retourniert int-Wert nach Methodenausführung.
lreturn	Retourniert long-Wert nach Methodenausführung.
return	Retourniert nach Methodenausführung ohne Rückgabewert (void-Methode).

Instruktionen zum Zugriff auf Attribute:

Instruktion	Funktion
getfield	Legt Attributinhalt eines Objekts auf Operanden-Stack ab. Die Klasse des Objekts, auf das der Zugriff erfolgen soll, wird über einen 16-Bit Offset auf dem runtime constant pool addressiert. Auch hier wird wieder die Limitierung der virtuellen Maschine auf 2^{16} Klassen deutlich.
getstatic	Legt Attributinhalt eines statischen Klassenattributes auf dem Operanden-Stack ab.
putfield	Setzt Wert eines Attributs.
putstatic	Setzt Wert eines statischen Klassenattributes.

Instruktionen zur Operation auf Arrays:

Instruktion	Funktion
newarray	Erzeugt einen neuen Array, und definiert die Komponententypen als einen der Primitvtypen. Die Implementierung von SUN verwendet für den Wahrheitswert je acht Bit. Anderen Umsetzungen ist es explizit Freigestellt hier mit optimierteren Speicherstrukturen zu operieren.
anewarray	Erzeugt einen neuen Array von Referenztypen zur Aufnahme beliebiger Objekte.
multianewarray	Erzeugt einen mehrdimensionalen Array.
aaload	Lädt Referenz von Arrayposition.
aastore	Speicher Objekt an spezifischer Arrayposition.
arraylength	Liefert Elementanzahl (Kardinalität) eines Arrays.

baload	Lädt byte oder boolean aus Arrayposition.
bastore	Legt byte oder boolean an Arrayposition ab.
saload	Lädt short aus Arrayposition.
sastore	Legt short an Arrayposition ab.
caload	Lädt char aus Arrayposition.
castore	Legt char an Arrayposition ab.
daload	Lädt double aus Arrayposition.
dastore	Legt double an Arrayposition ab.
faload	Lädt float aus Arrayposition.
fastore	Legt float an Arrayposition ab.
iaload	Lädt int aus Arrayposition.
iastore	Legt int an Arrayposition ab.
laload	Lädt long aus Arrayposition.
lastore	Legt long an Arrayposition ab.

Instruktionen zur Typkonversion:

Instruktion	Funktion
i2b	Konvertiert int zu byte.
i2c	Konvertiert int zu char.
i2d	Konvertiert int zu double.
i2f	Konvertiert int zu float.
i2l	Konvertiert int zu long.
i2s	Konvertiert int zu short.
d2f	Konvertiert double zu float.
l2d	Konvertiert long zu double.
l2f	Konvertiert long zu float.
l2i	Konvertiert long zu int.
d2f	Konvertiert double zu float.
f2d	Konvertiert float zu double.
f2i	Konvertiert float zu int.
f2l	Konvertiert float zu long.
d2f	Konvertiert double zu float.
d2f	Konvertiert double zu float.
d2i	Konvertiert double zu int.
d2l	Konvertiert double zu long.

Instruktionen zur Durchführung arithmetischer Operationen:
Eingangsperanden werden vom Stack entnommen und das Berechnungsergebnis ebenda abgelegt.

Instruktion	Funktion
dadd	Addiert zwei double-Werte.
dsub	Subtrahiert zwei double-Werte.
ddiv	Dividiert zwei double-Werte.
dmul	Multipliziert zwei double-Werte.
dneg	Negiert double-Wert durch Zweierkomplementbildung.
drem	Divisionsrest bei Division zweier double-Werte.
dcmp<*op*>	Vergleicht zwei double Werte. Ist einer der beiden Operanden NaN, so legt dcmpg1, dcmpl hingegen -1 als Ergebnis auf dem Stack ab.

fadd	Addiert zwei float-Werte.
fsub	Subtrahiert zwei float-Werte.
fmul	Multipliziert zwei float-Werte.
fdiv	Dividiert zwei float-Werte.
iadd	Addiert zwei int-Werte.
isub	Subtrahiert zwei int-Werte.
imul	Multipliziert zwei int-Werte.
idiv	Dividiert zwei int-Werte.
iand	Boole'sche UND-Verknüpfung zweier int-Werte.
ior	Boole'sche ODER-Verknüpfung zweier int-Werte.
ixor	Exklusive Boole'sche ODER-Verknüpfung zweier int-Werte.
ineg	Negiert int-Wert durch Zweierkomplementbildung.
irem	Divisionsrest bei Division zweier int-Werte.
ishl	Linksshift eines int-Wertes.
ishr	Rechtsshift eines int-Wertes.
iushr	Rechtsshift eines int-Wertes unter Nulleinfügung und Vorzeichenlösung.
fneg	Negiert float-Wert durch Zweierkomplementbildung.
frem	Divisionsrest bei Division zweier float-Werte.
fcmp<*op*>	Vergleicht zwei floatWerte. Ist einer der beiden Operanden NaN, so legt fcmpg1, fcmpl hingegen -1 als Ergebnis auf dem Stack ab.
ladd	Addiert zwei long-Werte.
land	Boole'sche UND-Verknüpfung zweier long-Werte.
ldcmp	Vergleicht zwei longWerte.
ldiv	Dividiert zwei long-Werte.
lmul	Multipliziert zwei long-Werte.
lneg	Negiert long-Wert durch Zweierkomplementbildung.
lrem	Divisionsrest bei Division zweier long-Werte.
lor	Boole'sche ODER-Verknüpfung zweier long-Werte.
lshl	Linksshift eines long-Wertes.
lshr	Rechtsshift eines long-Wertes.
lsub	Subtrahiert zwei long-Werte.
lushr	Rechtsshift eines long-Wertes unter Nulleinfügung und Vorzeichenlösung.
lxor	Exklusive Boole'sche ODER-Verknüpfung zweier long-Werte.

Sonstige Instruktionen:

Instruktion	Funktion	
Ausnahmebehandlung	athrow	Wirft eine Exception.
Synchronisation -- Monitoroperationen	monitorenter	Der Zugriff auf jedes Objekt wird durch einen Monitor synchronisiert. Die Anweisung sperrt ein Objekt.
	monitorexit	Gibt ein gesperrtes Objekt frei.
Sonstige Opcodes	nop	Buchstäblich: *no operation*. Bewirkt nichts; keine Operatoren, keine Änderungen am Operanden-Stack.

Die beiden Opcodes mit den Ordnungsnummern 254 und 255 (0xfe und 0xff, mnemonic impdep1 und impdep2) sind durch SUN als reserviert gekennzeichnet. Sie können von durch den Hersteller der virtuellen Maschine mit eigendefinierter Funktionalität implementiert werden.
Darüberhinaus ist mit dem Opcode 202 (mnemonic breakpoint) ein Einstiegspunkt für Debugger definiert.

Alle Opcodes sind mit einem Byte codiert. Hieraus ergibt 256 als maximaler Befehlsumfang der virtuellen Maschine. Zur Verringerung der notwendigen verschiedenen Befehle sind nicht alle Opcodes für alle Typen der JVM implementiert. Üblicherweise existieren nur Opcodes für int, float, long und double sowie die Referenzen. Für alle anderen Typen stehen Konvertierungsmöglichkeiten in die genannten zur Verfügung.

Opcode	byte	short	int	long	float	double	char	Referenz
...ipush	bipush	sipush						
...const			iconst	lconst	fconst	dconst		aconst
...load			iload	lload	fload	dload		aload
...store			istore	lstore	fstore	dstore		astore
...inc			iinc					
...aload	baload	saload	iaload	laload	faload	daload	caload	aaload
...astore	bastore	sastore	iastore	lastore	fastore	dastore	castore	aastore
...add			iadd	ladd	fadd	dadd		
...sub			isubb	lsub	fsub	dsub		
...mul			imul	lmul	fmul	dmul		
...div			idiv	ldiv	fdiv	ddiv		
...rem			irem	lrem	frem	drem		
...neg			ineg	lneg	fneg	dneg		
...shl			ishl	lshl				
...shr			ishr	lshr				
...ushr			iushr	lushr				
...and			iand	land				
...or			ior	lor				
...xor			ixor	lxor				
i2...	i2b	i2s		i2l	i2f	i2d		
l2...			l2i		l2f	l2d		
f2...			f2i	f2l	f2d			
...cmp				lcmp				
...cmpl				fcmpl	dcmpl			
...cmpg				fcmpg	dcmpg			
if_...cmp*cond*			if_icmp*cond*					if_acmp*cond*
...return			ireturn	lreturn	freturn	dreturn		areturn

Treten bei der Ausführung der Opcodes Ausnahmen auf, so werden durch die virtuelle Maschine Laufzeit-Ausnahmeereignisse (engl. *runtime exception*) generiert. Wie bekannt werden Ausnahmen dieser Kategorie (üblicherweise) nicht aufgefangen und behandelt.
So wird die `ClassCastException` im Beispiel aus Kapitel 2 durch die versuchte explizite Typumwandlung ausgelöst.
Der erzeugte Bytecode für diese Anweisung lautet:

```
aload_3
checkcast 2
```

`aload_3` lädt die Referenz auf eine lokale Variable auf den Operanden-Stack. Die lokale Variable 3 entspricht im Beispiel `myC11`.
`checkcast` testet ob die auf dem Operanden-Stack befindliche Referenz kompatibel zum übergebenen Typen (hier die 2 als Referenz auf die zweite geladene Klasse; benannt mit `C2`) ist. Im Falle der Nichtkompatibilität wird durch die virtuelle Maschine eine `ClassCastException` erzeugt.

Beispiel 1: Einfache arithmetische und Ein-/Ausgabeoperationen

```
(1).class public examples/BC1
(2).super java/lang/Object
(3)
(4).method public <init>()V
(5)   aload_0
(6)   invokenonvirtual java/lang/Object/<init>()V
(7)   return
(8).end method
(9)
(10).method public static main([Ljava/lang/String;)V
(11)    .limit locals 5
(12)   .limit stack 10
(13)
(14)    iconst_2
(15)    istore_0
```

```
(16)
(17)    bipush 101
(18)    istore_1
(19)
(20)    bipush 99
(21)    istore_2
(22)
(23)    ;we will need this twice
(24)    getstatic java/lang/System/out Ljava/io/PrintStream;
(25)    astore_3
(26)
(27)    iload_1
(28)    iload_2
(29)    iadd
(30)    istore_1
(31)
(32)    ;convert int to string
(33)    iload_1
(34)    invokestatic java/lang/String/valueOf(I)Ljava/lang/String;
(35)    astore 4
(36)
(37)    ;Print a string
(38)    aload_3
(39)    aload 4
(40)    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
(41)
(42)    iload_1
(43)    iload_0
(44)    idiv
(45)
(46)    ;convert int to string
(47)    invokestatic java/lang/String/valueOf(I)Ljava/lang/String;
(48)    astore 4
(49)
(50)    ;Print a string
(51)    aload_3
(52)    aload 4
(53)    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
(54)
(55)    ;print a fixed string
(56)    aload_3
(57)    ldc "The End"
(58)    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
(59)   return
(60).end method
(61)
```

Download des Beispiels

Der Java-Assemblercode des Beispiels 1 zeigt die Verwendung einiger einfacher arithmetischer und Ein-/Ausgabeoperationen.
Zunächst zeigt das Beispiel den Aufbau einer Java-Assemblerdatei, wie sie vom Übersetzer *Jasmin* akzeptiert und in ausführbaren Java-Bytecode umgewandelt wird.
So legt die `.class`-Deklaration zunächst fest, daß es sich um die öffentlich zugängliche (d.h. als `public` deklarierte) Klasse `BC1`, im Paket `examples` handelt.
Die darauf folgende `.super`-Definition legt die Elternklasse der betrachteten Klasse fest. Im Falle keiner explizit definierten Elternklasse ist dies vorgabegemäß die Standardklasse `Class`.
Nach den einführenden Deklarationen definiert die Quellcodedatei den statischen Initialisierter.

Die Methode `main` stellt den Beginn der aktiven Verarbeitung dar. Ihre Signaturdefinition (`([Ljava/lang/String;)V`) läßt die JVM-interne Kurzschreibweise des Typsystems erkennen. So deutet die in den Klammern der Übergabewerte eingeschlossene eckige Klammern an, daß ein Array gleichtypisierter Ausprägungen übergeben wird. Diese Ausprägungen sind alle vom Standardtyp `java.lang.String` (die paket-separierenden Punkte werden JVM-intern zu Pfadseparatoren aufgelöst). Zusätzlich ist dem Klassenname ein einleitendes `L` vorangestellt, um auszudrücken, daß es sich nicht um einen Primitivtyp, sondern um eine Sprachkomponente (das „L“ deutet hierbei auf den Begriff *language* hin) handelt.
Nach der Klammer ist der Rückgabetyp --- im Falle von `main` vorgabegemäß `void` --- angegeben. Auch er wird unter Verwendung derselben Abkürzungskonvention dargestellt.

Zu Eingangs der Methode `main` allozieren die beiden Direktiven `.limit` zunächst Speicher für die lokalen Variablen (`.limit locals`) und die Tiefe des methodenintern verwendeten Operandenstacks (`.limit stack`).

Die (aktive durch den Programmierer gesteuerte) Verarbeitung beginnt im Beispiel mit dem Anweisung `iconst_2` welche den ganzzahligen Wert 2 auf dem Operandenstack ablegt. Anschließend wird dieser Wert, mittels der Anweisung `istore_0` vom Stack entnommen und in die erste lokale Variable gespeichert.
Mit `iconst_2` findet ein besonderer Befehl zur Ablage einer ganzzahligen Konstante auf dem Operanden Stack Verwendung, der es gestattet bestimmte (häufig benötigte) Konstantenablagen in nur genau einem Byte auszudrücken. Durch die JVM-Spezifikation vorgesehen sind hierbei Instruktionen für die Konstanten `-1, 0, 1, 2, 3, 4` oder `5`. Im Ergebnis ist die Nutzung der abkürzenden Befehlsschweibweise äquivalent zum Einsatz der

Instruktion bipush unter Explizierung der abzulegenden Konstante.

Diese äquivalente Form der Belegung einer lokalen Variable zeigt der zweite Anweisungsblock, der die numerische Konstante 101, für die keine abkürzende Schreiweise angeboten wird, auf dem Operandenstack ablegt um sie der zweiten lokalen Variable (mit der Indexnummer 1) zuzuweisen.
In derselben Weise wird für die Initialisierung der dritten lokalen Variablen mit dem Wert 99 verfahren.

Anschließend wird durch getstatic der Dateideskriptor der Standardausgabe (d.h. desjenigen Streams mit dem Wert System/out) gelesen und die zurückgelieferte Adresse in der vierten lokalen Variable (Indexnummer 3) abgelegt.

Der darauffolgende Anweisungsblock zeigt die Umsetzung einer einfachen Ganzzahladdition, die zunächst die beiden zu verknüpfenden Operanden (die Inhalte der lokalen Variablen mit den Indexnummern 1 und 2) auf dem Stack ablegt und anschließend mittels der Ganzzahladdition (iadd) verknüpft.
Das auf dem Stack abgelegte Berechnungsergebnis wird durch istore_1 er zweiten lokalen Variablen zugewiesen.

Der nächste Anweisungsblock bereitet die Ausgabe des Berechnungsergebnisses auf der Standardausgabe vor. Hierzu plaziert er zunächst den Inhalt der lokalen Variable mit der Indexnummer 1 (d.h. das Berechnungsergebnis des direkt vorhergehenden Schrittes) auf dem Stack.
Anschließend wird eine Standard-API-Methode (die Methode valueOf) aufgerufen, welche den auf dem Stack übergebenen int-Parameter in eine Zeichenkette wandelt und die Referenz darauf als Rückgabewert auf dem Stack plaziert.
Dieser Rückgabewert wird in der fünften lokalen Variable (Indexnummer 4) abgelegt.

Anschließend werden die in zwischenzeitlich den vierten und fünften lokalen Variablen abgelegten Adressen des Ausgabe-Streams und der auszugebenden Zeichenkette geladen und auf dem Operanden-Stack abgelegt.
Durch Aufruf der Standard-Ausgabemethode println mittels invokevirtual wird die referenzierte Zeichenkette auf der Standardausgabe dargestellt.

Der folgende Anweisungsblock demonstriert eine Ganzzahldivision mittels idiv welche Divisior und Dividenden als Operadnen auf dem Stack erwartet und das Berechnungsergebnis ebenda plaziert.

Anschließend wird das (noch auf dem Stack liegende) Berechnungsergebnis direkt weiterverarbeitet und in eine Zeichenkette gewandelt. Hierbei kommt die bereits bekannte Funktion zum Einsatz.
Danach erfolgt wiederum die Ausgabe in der bekannten Form.

Abschließend wird eine fixe Zeichenkette ausgegeben, deren Zeichenkettendarstellung nicht berechnet zu werden braucht. Ihr Wert kann daher direkt aus dem Laufzeitkonstantenpool per ldc geladen werden.
Die übrigen Schritte zur Erzeugung der Ausgabe bleiben indes unverändert.

Nutzung von Methoden:
Bereits bei den einfachen Operationen aus Beispiel 1 zeigt sich, daß die wiederholte Angabe von sehr ähnlichen Instruktionsfolgen nicht zu vermeiden ist. Insbesondere die beiden Konversionen des int-Datentyps als Voraussetzung der zeichenbasierten Ausgabe ist vollständig identisch.
Zur Strukturierung stehen daher auf der Java-Assemblerebene die bereits aus der Java-Hochsprache bekannten *Methoden* zur Verfügung, wie Beispiel 2 zeigt.

Beispiel 2: Nutzun von Methoden

```
(1).class public examples/BC2
(2).super java/lang/Object
(3)
(4).method public <init>()V
(5)   aload_0
(6)   invokenonvirtual java/lang/Object/<init>()V
(7)   return
(8).end method
(9)
(10).method public static printInt(I)V
(11)    .limit locals 2
(12)    .limit stack 2
(13)
(14)    iload_0
(15)    invokestatic java/lang/String/valueOf(I)Ljava/lang/String;
(16)    astore_1
(17)
(18)    getstatic java/lang/System/out Ljava/io/PrintStream;
(19)    aload_1
(20)    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
(21)    return
(22).end method
(23)
(24).method public static main([Ljava/lang/String;)V
(25)    .limit locals 50
(26)   .limit stack 40
(27)
(28)    iconst_2
(29)    istore_0
(30)
(31)    bipush 101
(32)    istore_1
```

```
(33)
(34)    bipush 99
(35)    istore_2
(36)
(37)    ;we will need this twice
(38)    getstatic java/lang/System/out Ljava/io/PrintStream;
(39)    astore_3
(40)
(41)    iload_1
(42)    iload_2
(43)    iadd
(44)    istore_1
(45)
(46)    iload_1
(47)    invokestatic examples/BC2/printInt(I)V
(48)
(49)    iload_1
(50)    iload_0
(51)    idiv
(52)
(53)    invokestatic examples/BC2/printInt(I)V
(54)
(55)    ;print a fixed string
(56)    aload_3
(57)    ldc "The End"
(58)    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
(59)   return
(60).end method
```

Download des Beispiels

Die Funktionalität des Beispieles ist mit der der vorhergend vorgestellten Codesequenz identisch. Jedoch finden sich jetzt die Instruktionsfolgen zur Berechnung der Zeichenkettenrepräsentation einer Ganzzahl und ihrer anschließenden Ausgabe in die Methode `printInt` ausgelagert.

Diese Methode akzeptiert eine Ausprägung des Primitivtyps `int` als Übergabe und liefert keinen Rückgabewert. Die Signatur ist daher dahingehend vereinbart, daß genau eine `int`-konforme Zahl als Parameter auf dem Stack erwartet wird, d.h. der Aufrufer hat diese vor dem Aufruf dort abzulegen.

Zusätzlich benötigt die Methode selbst zu ihrer Ausführung einige lokale Variablen, die auf dem methodenspezifischen Stack abgelegt werden. Dieser stellt eine Erweiterung des bereits durch den Aufruf verwendeten Operandenstacks dar.

Mit Java steht jedoch keineswegs die einzige Hochsprache zur Erzeugung von Byte-Code-Dateien zur Verfügung. Diese Seite listet eine Vielzahl verschiedener Alternativen.
Beispielsweise erzeugt der Oberon-Compiler von Canterbury für alle Oberon-Module, einschließlich der Systemmodule, Java-Klassen.

Beispiel 3: Die Hello World Applikation als Oberon Programm

```
MODULE helloworld;

IMPORT Out;

BEGIN
  Out.String( "Hello World" );
  Out.Ln;
END helloworld.
```

Download des Beispiels

Die erzeugten `class`-Dateien -- SYSTEM.class, helloworld.class, Out.class, Sys.class -- können auf jeder JVM zur Ausführung gebracht werden. `java helloworld` liefert das erwartete Ergebnis.

Das **Class-File-Format**

Ausgangspunkt jeder Programmausführung innerhalb der JVM ist die `class`-Datei als Eingabe. Sie wird üblicherweise durch durch den Java-Compiler (im JDK: `javac` erzeugt).

Einige Eigenschaften jedes `class`-Files:

- Es enthält die Definition genau einer Klasse oder einer Schnittstelle, unabhängig von deren Zugriffs- und Sichtbarkeitseigenschaften.
- Es wird als Bytestrom aufgefaßt, daher werden alle größeren Informationseinheiten durch serielles Lesen und aneinanderfügen gebildet.
  Unabhängig von der physischen Realisierung der tatsächlich zugrundeliegenden Hardware werden alle Multibyte-Daten im *big endian*-Format, d.h. größere Einheiten werden durch Linksshift und Boole'sches ODER gebildet, abgelegt.

Die Schnittstellen `java.io.DataInput`, `java.io.DataOutput`, `java.io.DataInputStream` und `java.io.DataOutputStream` unterstützen dieses Format. (Beispiel)

- Die Strukturen innerhalb der `class`-Datei werden nicht zusätzlich optimiert abgelegt, daher erfolgt weder ein Auffüllen auf spezifische Wortgrenzen, noch ein Alignment an solchen.

Die JVM-Spezifikation legt zur Definition der Struktur des `class`-Files eigene Datentypen fest: `u1`, `u2` und `u4` zur Definition vorzeichenloser ein-, zwei- und drei-Bytetypen. Für diese (von der Java-üblichen vorzeichenbehafteten Mimik (abgesehen von char) abweichenden) Datentypen stehen mit `readUnsignedByte()`, `readUnsignedShort()` und `readInt()` entsprechende Lesemethoden zur Verfügung.

The `class` File Format @ Java Virtual Machine Specification

```
ClassFile {
      u4 magic;
      u2 minor_version;
      u2 major_version;
      u2 constant_pool_count;
      cp_info constant_pool[constant_pool_count-1];
      u2 access_flags;
      u2 this_class;
      u2 super_class;
      u2 interfaces_count;
      u2 interfaces[interfaces_count];
      u2 fields_count;
      field_info fields[fields_count];
      u2 methods_count;
      method_info methods[methods_count];
      u2 attributes_count;
      attribute_info attributes[attributes_count];
   }
```

Constant Pool @ Java Virtual Machine Specification

```
cp_info {
      u1 tag;
      u1 info[];
   }
```

`field_info` @ Java Virtual Machine Specification

```
field_info {
        u2 access_flags;
        u2 name_index;
        u2 descriptor_index;
        u2 attributes_count;
        attribute_info attributes[attributes_count];
           }
```

`method_info` @ Java Virtual Machine Specification

```
method_info {
        u2 access_flags;
        u2 name_index;
        u2 descriptor_index;
        u2 attributes_count;
        attribute_info attributes[attributes_count];
   }
```

`attribute_info` @ Java Virtual Machine Specification

```
  attribute_info {
        u2 attribute_name_index;
        u4 attribute_length;
        u1 info[attribute_length];
   }
```

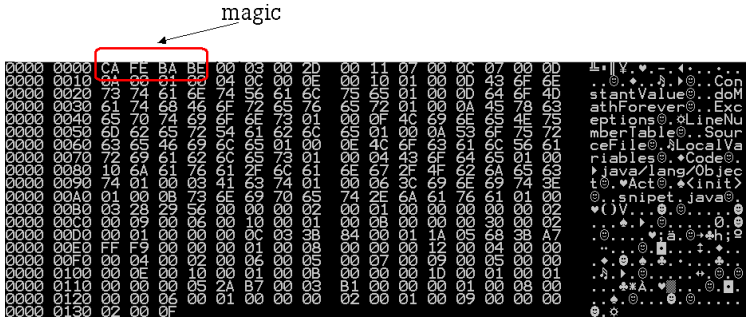
Aufbau einer `class`-Datei verdeutlicht an nachfolgendem Beispielquellcode.

Hinweis: Mit *Classeditor* existiert ein freies Werkzeug zur Inspektion und Modifikation übersetzter Class-Dateien.

Beispiel 4: Java-Quellcode der untersuchten Klassendatei

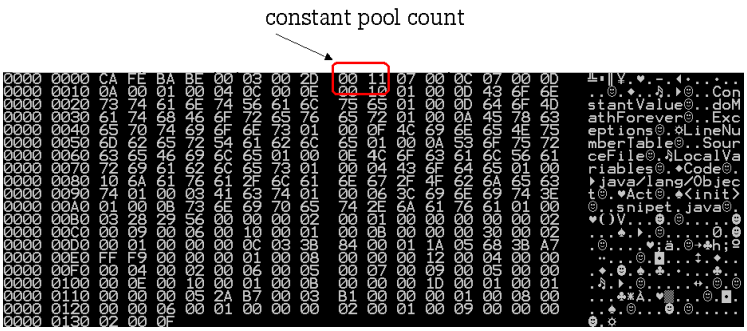
```
(1)class Act {
(2)     public static void doMathForever() {
(3)             int i=0;
(4)             while (true) {
(5)                     i += 1;
(6)                     i *= 2;
(7)             } //while
(8)     } //doMathForever()
(9)} //class Act
```

Download des Beispiels



Der `magic`-Identifier ist auf die Bytekombination (in hexadezimaler Darstellung) `CA FE BA BE` fixiert. Anhand dieser erkennt der Kassenlader der Laufzeitsystems die Datei als ausführbare Java-Bytecode-Datei an.
Ist diese gegenüber dem Vorgabewert modifiziert wird eine `java.lang.ClassFormatError` Ausnahme im Hauptthread generiert (`Bad magic number` wird als zusätzliche Nachricht der Ausnahme ausgegeben). siehe JVM-Spezifikation

Die beiden Versionskennungen `minor` und `major` bilden gemeinsam den Versionsschlüssel der `class`-Datei in der gängigen Punktnotation. Hierbei gilt: `major.minor`
Die Klassendatei des Beispiels trägt den Versionsschlüssel `45.3`.
Der im SUN JDK enthaltene Java-Compiler erlaubt per Kommandozeilenparameter (`target`) die JVM-spezifische Steuerung der Codegenerierung. Die Auswahl von `1.1` als Zielplattform generiert die Versionskennung `45.3`, `1.2` -- `46.0`, `1.3` -- `47.0` und `1.4` (mit den Abarten `1.4.1` und `1.4.2`) sowie der erste Prototyp des 1.5-Compilers -- `48.0`.
Die zweite Vorversion generierte bereits Bytecode mit der Versionsnummer `50.0`, während die erste regulär verfügbare Betaversion die Versionsnummer `49.0` erzeugt.
Vorgabegemäß wird durch die Compilerversion `1.4.2` das Format für `1.2` und ab Version `1.5` `48.0` erzeugt.
Trägt eine `class`-Datei eine durch die JVM nicht unterstützte Versionsnummer, so wird eine `java.lang.UnsupportedClassVersionError`-Ausnahme generiert.
Jede JVM kann verschiedene `class`-Datei-Versionen unterstützen, die letzendlich Festlegung welche Versionen durch einzelne Java-Plattform-Releases zu unterstützten sind obliegt jedoch SUN. So unterstützt SUNs JDK v1.0.2 `class`-Dateien der Versionen 45.0 bis einschließlich 45.3. Die JDK-Generation v1.1.x ab Version 45.0 bis einschließlich 45.65535 und Implementierungen der Java 2 Plattform, Version 1.2, bis einschließlich 46.0. JDK v1.3.0 verarbeitet Klassendateien bis hin zur Versionsnummer `47.0`. Zur Verarbeitung von Klassen, welche die in 1.5 eingeführten Generizitätsmechanismen verwenden nicht zwingend eine Ausführungsumgebung dieser Versionsnummer benötigt, da das erzeugte Klassenformat (bisher, da diese Aussagen auf dem Informationsstand der verfügbaren Betaversion basieren) nicht verändert wurde. Die Nutzung des dynamischen Boxing/Unboxings benötigt jedoch eine Ausführungsumgebung mindestens der Version 1.5.
siehe JVM-Spezifikation

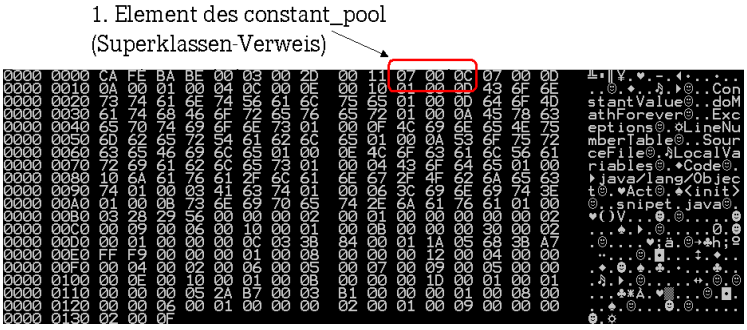


Die Bytefolge `constant_pool_count` enthält die um eins erhöhte Anzahl der Einträge der `constant_pool` Tabelle. Im Beispiel ist dies: 17.
siehe JVM-Spezifikation

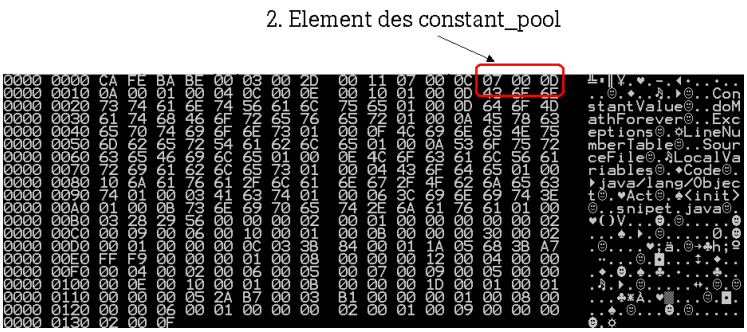
Der `constant pool` enthält die symbolische Information über Klassen, Schnittstellen, Objekte und Arrays. Die Elemente des dieser Datenstruktur sind vom Typ `cp_info` und variieren je nach `tag` in ihrer Länge. Als Konstantentypen (=Inhalt des Tag-Bytes) sind zugelassen:

Konstantentyp	Wert
`CONSTANT_Utf8`	1
`CONSTANT_Methodref`	10
`CONSTANT_InterfaceMethodref`	11
`CONSTANT_NameAndType`	12
`CONSTANT_Integer`	3
`CONSTANT_Float`	4
`CONSTANT_Long`	5
`CONSTANT_Double`	6
`CONSTANT_CLASS`	7
`CONSTANT_String`	8
`CONSTANT_Fieldref`	9

siehe JVM-Spezifikation

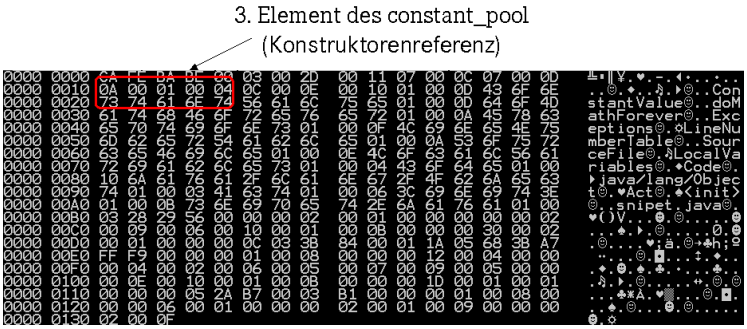


Konstante vom Typ `CONSTANT_class` die einen Verweis auf auf das zwölfte Element des Konstantenpools (0x0C) enthält. Zum Lesezeitpunkt der ersten Konstante kann diese Referenz noch nicht aufgelöst und auf Gültigkeit geprüft werden. (Später sehen wir, daß es sich um eine Referenz auf `java.lang.Object` handelt).
Hierbei handelt es sich immer um die Referenz auf die Superklasse. Auch für die API-Klasse `Object` selbst findet sich diese Refenz in der Klassendatei, auch wenn Diassemblierungswerkzeuge wie `javap` diese nicht ausgeben.

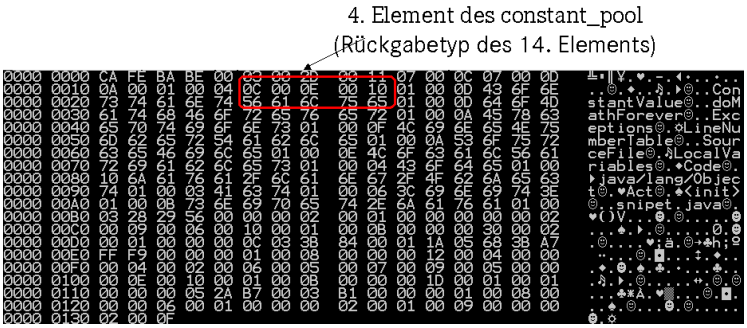


Konstante vom Typ `CONSTANT_class` die einen Verweis auf auf das 13. Element des Konstantenpools (0x0D) enthält. An dieser Stelle findet sich der String `Act`, also der Klassenname der zur Klassendatei gehörigen Klasse selbst. Auch

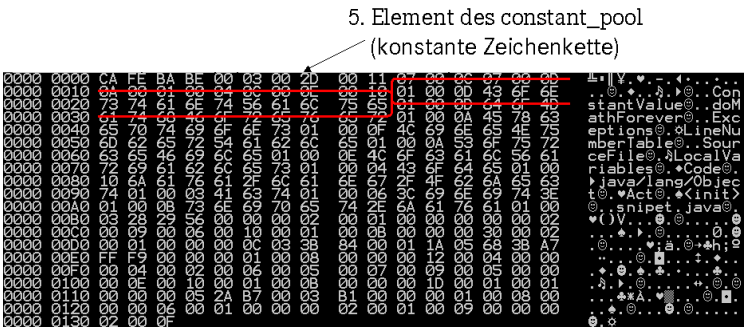
hierbei handelt es sich zunächst um eine nicht auflösbare Vorwärtsreferenz.
Technisch gesehen realisiert sie den `this`-Verweis.



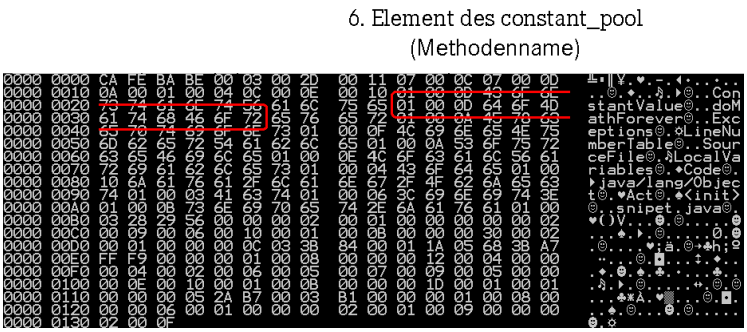
Konstante vom Typ `CONSTANT_Methodref`. Die folgenden zwei Bytes (im Beispiel: `0x00 01`) bezeichnen das referenzierte Objekt, gefolgt vom Methodenindex (`0x00 04`). Im Beispiel handelt es sich um das Objekt mit der Referenznummer 1 (=erstes Element des Konstantenpools, die Superklasse `Object`). Unter der Referenznummer `0x04` wird auf die Methode `<init>()` verweisen. Da die betrachtete Klasse `Act` keinen eigenen Konstruktor definiert, wird der der Superklasse aufgerufen.



Konstante vom Typ `CONSTANT_NameAndType`. Die folgenden beiden Bytes (`0x00 0E`) verweisen auf die zu beschreibende Position innerhalb des Konstantenpools (im Beispiel: `init`). Dieser Position wird der and Position `0x10` spezifizierte Typ als Rückgabetyp zugeordnet -- im Beispiel `()V`; also `void`.

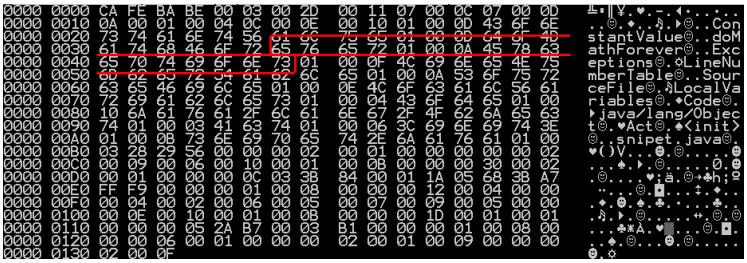


Konstante vom Typ `CONSTANT_Utf8` leitet einen konstenten Zeichenketten-Ausdruck ein.
Zeichenketten werden generell im Unicode UTF-8 Format abgelegt, wobei jedoch aus Speicherplatzeffizienzgründen für die Zeichen zwischen `\u0001` und `\u007F` nur jeweils ein Byte benötigt werden. Für alle anderen Unicode-Symbole wird der entsprechende 2- bzw. 3-Byte Speicherplatz zur Verfügung gestellt.
Die Konstante wird von der Länge der Zeichenkette (im Beispiel: 13) gefolgt, daher kann auf eine terminierende Null verzichtet werden.
Die Bedeutung dieser -- im Java-Quellcode nicht enthaltenen -- Zeichenkette wird im Kontext des Klassenladevorgangs deutlich.



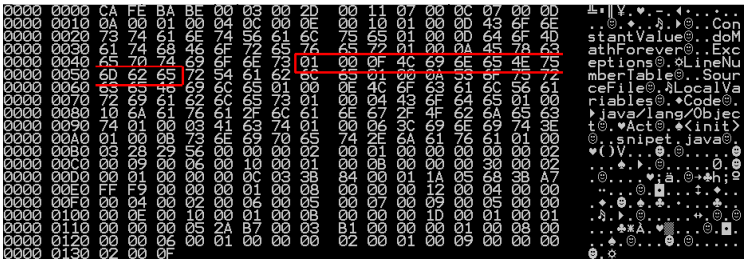
Konstante vom Typ `CONSTANT_Utf8`. Sie leitet den in der Klasse `Act` spezifizierten Methodennamen `doMathForever` ein.

7. Element des constant_pool
(Zeichenkette "Exceptions")



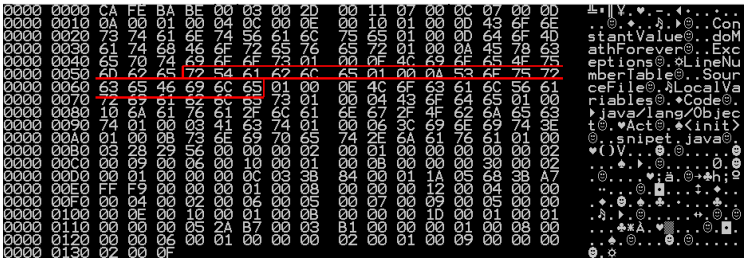
Konstante vom Typ CONSTANT_Utf8, die den fixen String *Exceptions* einleitet.

8. Element des constant_pool
(Zeichenkette "LineNumberTable")



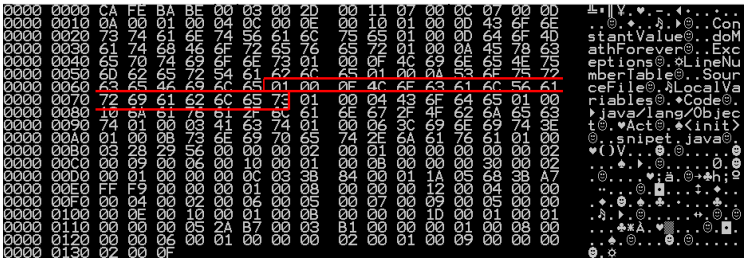
Konstante vom Typ CONSTANT_Utf8, die den fixen String *LineNumberTable* einleitet.

9. Element des constant_pool
(Zeichenkette "SourceFile")



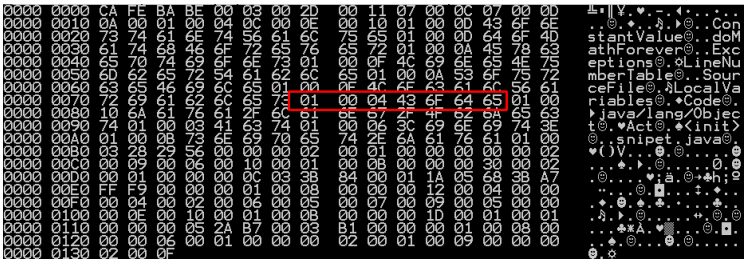
Konstante vom Typ CONSTANT_Utf8, die den fixen String *SourceFile* einleitet.

10. Element des constant_pool
(Zeichenkette "LocalVariable")



Konstante vom Typ CONSTANT_Utf8, die den fixen String *LocalVariables* einleitet.

11. Element des constant_pool
(Zeichenkette "Code")

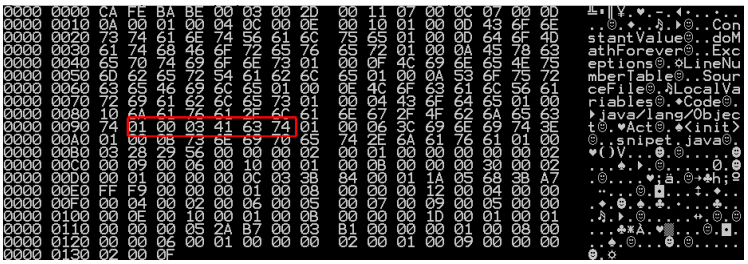


Konstante vom Typ CONSTANT_Utf8, die den fixen String *Code* einleitet.

12. Element des constant_pool
(Zeichenkette "java/lang/Object")

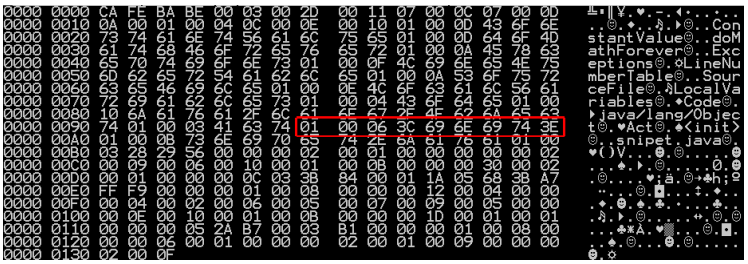
Konstante vom Typ `CONSTANT_Utf8`, die den String *java/lang/Object* einleitet. Vom ersten Element des Konstantenpools referenziertes Element. Die in der Java-Hochsprache üblichen Punkte zur Trennung der Pakte, Subpakete und Klassennamen werden in der JVM konsequent (aus historischen Gründen) durch Querstriche ersetzt.

13. Element des constant_pool
(Zeichenkette "Act")



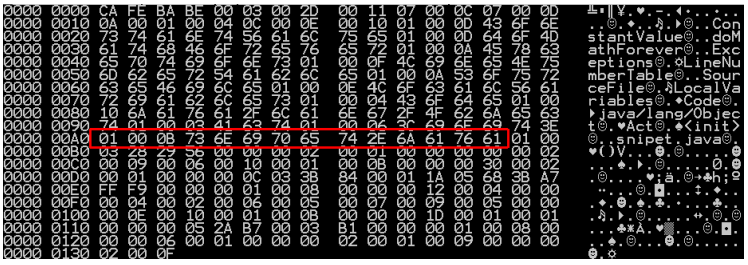
Konstante vom Typ `CONSTANT_Utf8`, die den String *Act* einleitet. Vom zweiten Element des Konstantenpools referenziertes Element. Name der Klasse.

14. Element des constant_pool
(Zeichenkette "<init>")



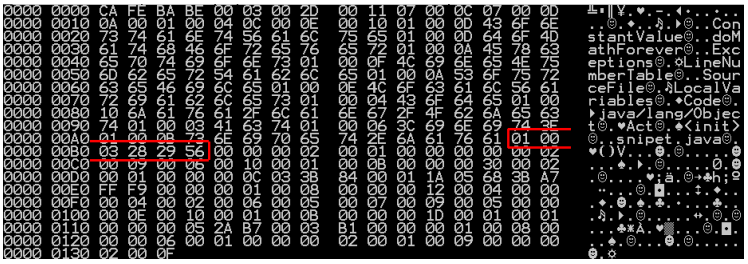
Konstante vom Typ `CONSTANT_Utf8`, die den String *<init>* einleitet. Methodenname der innerhalb der Superklasse `Object`. Der Rückgabewert dieser Methode ist im vierten Element des Konstantenpools abgelegt.

15. Element des constant_pool
(Zeichenkette "snipet.java")



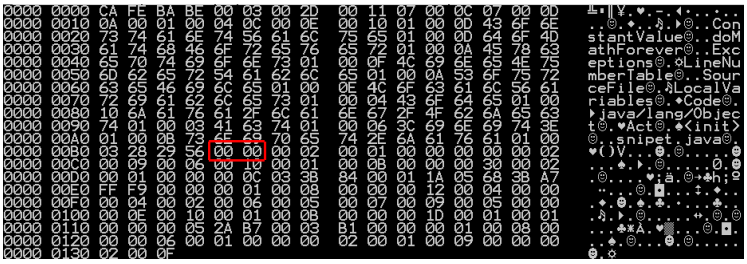
Konstante vom Typ `CONSTANT_Utf8`, die den String *snipet.java* -- den Namen der Quellcodedatei in der sich die Definition der Klasse `Act` befindet -- einleitet.

16. Element des constant_pool
(Zeichenkette "()V")



Konstante vom Typ `CONSTANT_Utf8`, die den String *()V* einleitet. Methodendeskriptor, der weder Übergabeargumente noch Rückgabetyp besitzt.

Zugriffsflags



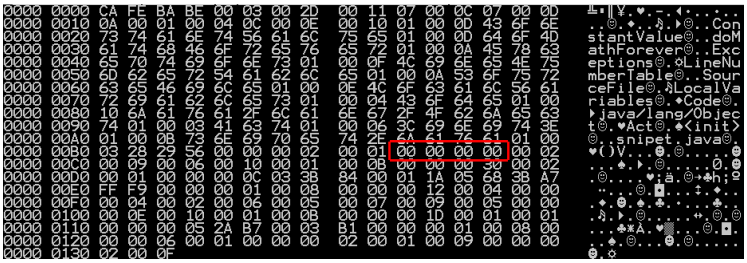
Zugriffsflags für die Klasse `Act`. Der konkrete Code ergibt sich aus der binären ODER-Verknüpfung verschiedener Zugriffsflaggen, die in untenstehender Tabelle wiedergegeben sind.

Flag	Wert	Beschreibung
ACC_PUBLIC	0x0001	`public`-Deklaration; Zugriffbar von allen anderen Klassen, auch außerhalb des eigenen Pakets
ACC_FINAL	0x0010	`final`-Deklaration; Verbot der Vererbung
ACC_SUPER	0x0020	Besondere Behandlung der Superklasseninstruktionen bei Aufruf über Opcode `invokespecial`
ACC_INTERFACE	0x0200	Struktur ist Schnittstelle, keine Klasse
ACC_ABSTRACT	0x0400	Abstrakte Struktur, von der keine Ausprägungen erzeugt werden können

Referenz auf this und super

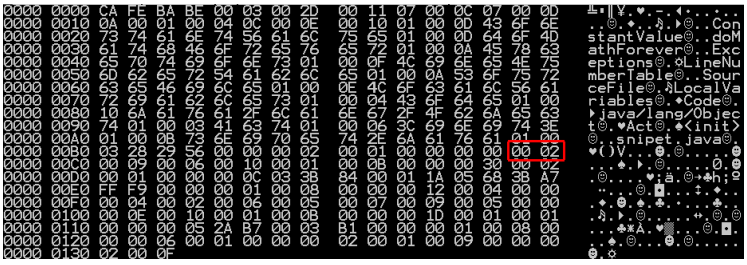
Referenz in den Konstantenpool, auf die Klasse selbst (`this`) und die Superklasse (`super`).

interface_count und fields_count



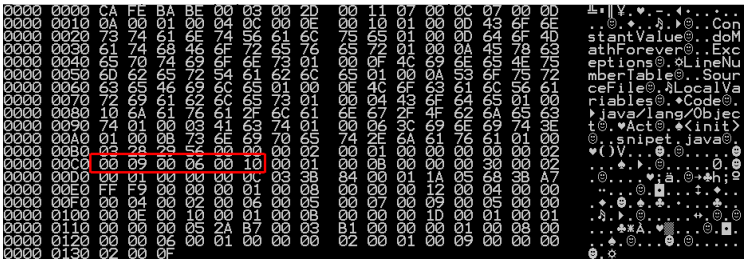
`interface_count`: Anzahl der durch die Klasse implementierten Schnittstellen.
`fields_count`: Anzahl der Klassen- oder Instanzvariablen der Klasse.

methods_count



Anzahl der durch die Klasse implementierten Methoden.
Die Zahl ergibt sich aus der tatsächlich durch den Anwender definierten Anzahl und den impliziten, d.h. durch den Compiler hinzugefügten (z.B. Konstruktor), Methoden.

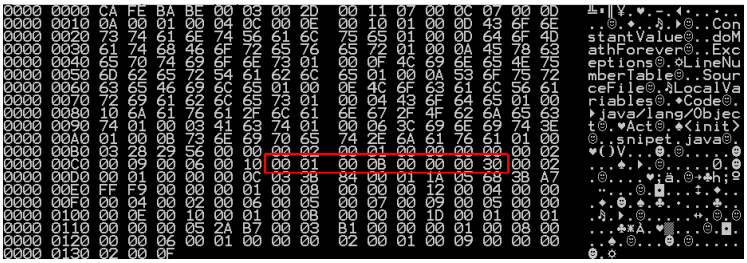
methods_info
(doMathForever())



Informationen über die in der Klasse `Act` definierten Methoden.
Die Zugriffsflaggen (0x00 09) weisen `doMathForever()` als `static` und `public` aus. (Die konkrete Wertebelegung kann untenstehender Aufstellung entnommen werden. Auch in diesem Falle wird der tatsächliche Wert durch Boole'sche ODER-Verknüpfung der Einzelwerte gebildet.)
Der Verweis (0x06) auf das sechste Element des Konstantenpools referenziert den Methodennamen im Klartext. Der zweite Verweis (0x10) auf den Konstantenpool kennzeichnet `doMathForever()` als parameterlose Methode ohne Rückgabetypen.

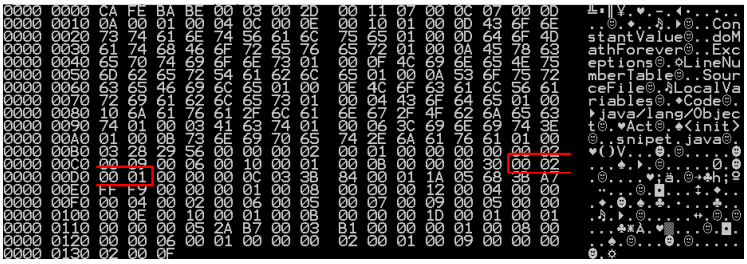
Flag	Wert	Beschreibung
ACC_PUBLIC	0x0001	`public`-Deklaration; Zugriffbar von allen anderen Klassen, auch außerhalb des eigenen Pakets
ACC_PRIVATE	0x0002	Als `private` deklariert, daher nur innerhalb der definierenden Klasse verwendbar
ACC_PROTECTED	0x0004	`protected`-Deklaration, Zugriff nur in Subklassen möglich
ACC_STATIC	0x0008	`static`-Deklaration, keine Auswirkungen auf Sichtbarkeit und Zugriffsrechte
ACC_FINAL	0x0010	`final`-Deklaration; nach initialer Zuweisung keine Wertänderung möglich
ACC_VOLATILE	0x0040	`volatile`-Deklaration; keine Berücksichtigung in Optimierungsmaßnahmen
ACC_TRANSIENT	0x0080	`transient`-Deklaration; keine Berücksichtigung durch Perisistenzmanager

Attribute der Methode doMathForever()



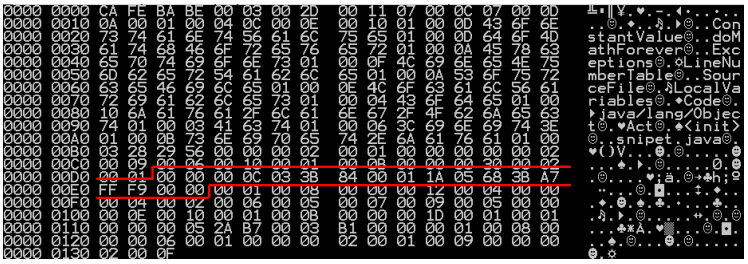
Die Methode `doMathForever()` verfügt nur über genau ein Attribut, daher ist der `attribute count` zu Beginn der Bytesequenz auf 0x00 01 gesetzt. Dieses eine Attribut wird durch Index 11 innerhalb des Konstantenpools referenziert. Dort ist die Zeichenkette `Code` lokalisiert. Dadurch wird angezeigt, daß die folgenden Bytes die Implementierung dieser Methode beinhalten.
Der abschließende vier-Byte Indikator enthält die Länge der Methodenimplementierung (im Beispiel: 0x30).

maxStack und maxLocals der
Methode doMathForever()



`maxStack`: Maximalhöhe des Operandenstacks die während der Methodenausführung erreicht werden kann. (Im Beispiel: 2; die Opcode-Implementierung der beiden verwendeten arithmetischen Operationen benötigen niemals mehr als zwei Stackpositionen.)
`maxLocals`: Anzahl der lokalen Variablen. (die verwendete Variable `i`)

Bytecode und Ausnahmentablle der Methode doMathForever()



Die code length legt die Anzahl der folgenden Bytecode-Instruktionen fest (im Beispiel: 12), darauf folgen die tatsächlichen Opcodes.

```
pc    instruction    mnemonic
0     03             iconst_0
1     3B             istore_0
2     840001         iinc 0 1
5     1A             iload_0
6     05             iconst_2
7     68             imul
8     3B             istore_0
9     A7FFF9         goto 2
```

Die Ausnahmentabelle (Exception Table) enthält die Anzahl der durch die Methode aufgefangenen Ausnahmeereignisse; im Beispiel: 0.

Eigenschaften des Code-Bereichs der Methode doMathForever()

In diesem Bereich werden zusätzliche Charakteristika des bereits definierten Codebereichs hinterlegt, z.B. Debugginginformation.
Im betrachteten Falle ist nur eine Eigenschaft angegeben (attribute_count = 0x01). Diese referenziert das achte Element des Konstantenpools -- die Zeichenkette LineNumberTable. Die beiden abschließenden Attribute bezeichnen die Länge dieser Tabelle (0x12) und die Anzahl der Einträge (0x4).
Die LineNumberTable des Beispiels:

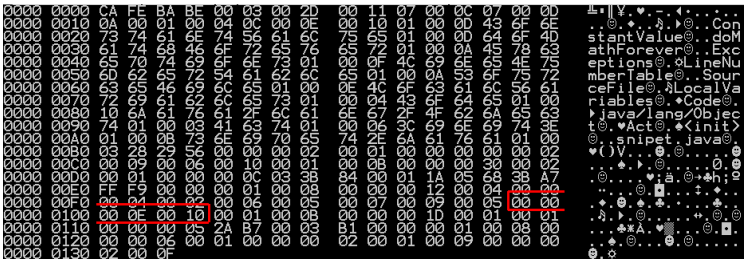
```
line 4: i = 0;
line 5: while(true) {
line 6: i += 1;
line 7: i *= 2;
```

Zuordnung zwischen LineNumberTable und der Methode doMathForever()

Diese Datenstruktur stellt die Zuordnung zwischen den Quellcodezeilen und den resultierenden Opcodes her.

```
LineNumberTable[0]:  iconst_0    istore_0
LineNumberTable[1]:  iinc 0 1
LineNumberTable[2]:  iload_0     iconst_2    imul  istore_0
LineNumberTable[3]:  goto 2
```

Zugriffsflaggen und Indizes der Klasse Act()



Diese zweite methodenbezogene Struktur gibt Auskunft über den Konstruktor der Klasse `Act`.
Im ersten Doppelbyte sind die Zugriffsrechte spezifiziert; in diesem Falle sind keine gesonderten Festlegungen getroffen -- es handelt sich um eine *einfache* Methode.
Die Referenz in den Konstantenpool verweist auf die implementierende Methode (im Beispiel: Position 0x0E, dort findet sich die Methode `<init>`).
Durch die letzten beiden Bytes wird der Typ des Konstruktors referenziert, im betrachteten Beispiel die Position 0x10 im Konstantenpool, mithin ein parameterloser Konstruktor.

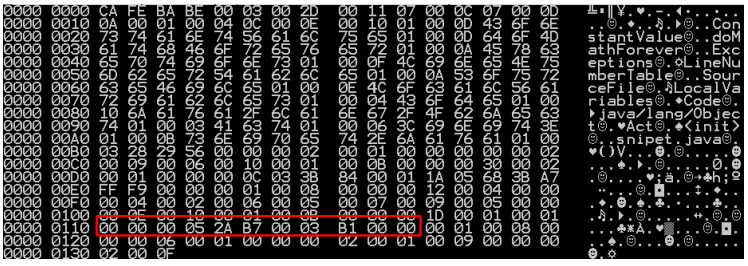
Attribute der Klasse Act()

Der Zähler (ersten beiden Bytes) zu beginn der Struktur zeigt an, daß nur ein Attribut der Klasse `Act()` folgt. Im Beispielfall handelt es sich dabei um das über den Index 11 (0x0B) angesprochene Element des Konstantenpools, die Zeichenkette `Code`.
Der abschließend angegebene Längenzähler fixiert die Anzahl der folgenden Bytes.

maxStack und maxLocals der Klasse Act()

Analog der Definition für Methoden, die maximale Höhe des Operandenstacks und die Anzahl der lokalen Variablen.

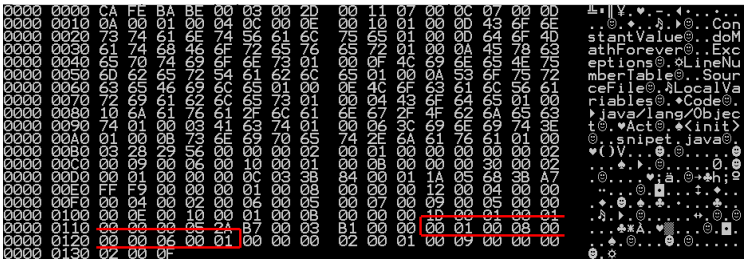
Bytecode und Ausnahmetabelle der Klasse Act()



Opcode-Implementierung des Konstruktors, sowie die Aufzählung der durch ihn potentiell ausgelösten Ausnahmeereignisse (im keine, daher Anzahl gleich Null).
Die Implementierung in Java-Bytecode:

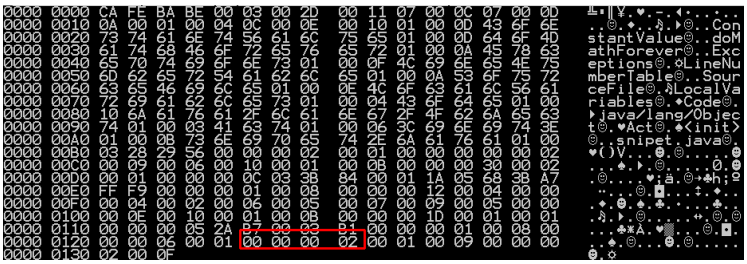
```
pc    instruction    mnemonic
0     2A             aload_0
1     B70003         invokeonvirtual #3 <Method java.lang.Object <init> ()V>
4     B1             return
```

Eigenschaften des Code-Bereichs der Klasse Act()



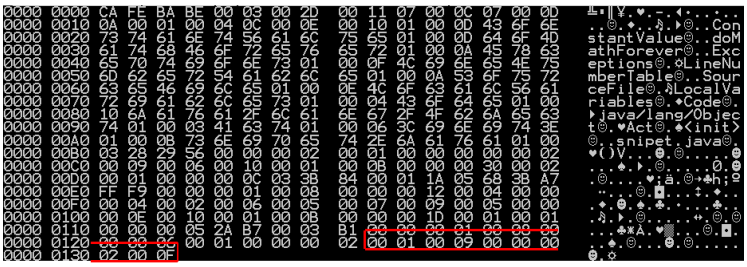
Anzahl der Eigenschaften im ersten Doppelbyte (im Beispiel: 1). Die spezifische Eigenschaft wird durch Index acht im Konstantenpool (=`LineNumberTable`) näher definiert.
Diese Tabelle hat die Länge 0x06, mit einem einzigen Eintrag.

Zuordnung zwischen LineNumberTable und der Klasse Act()



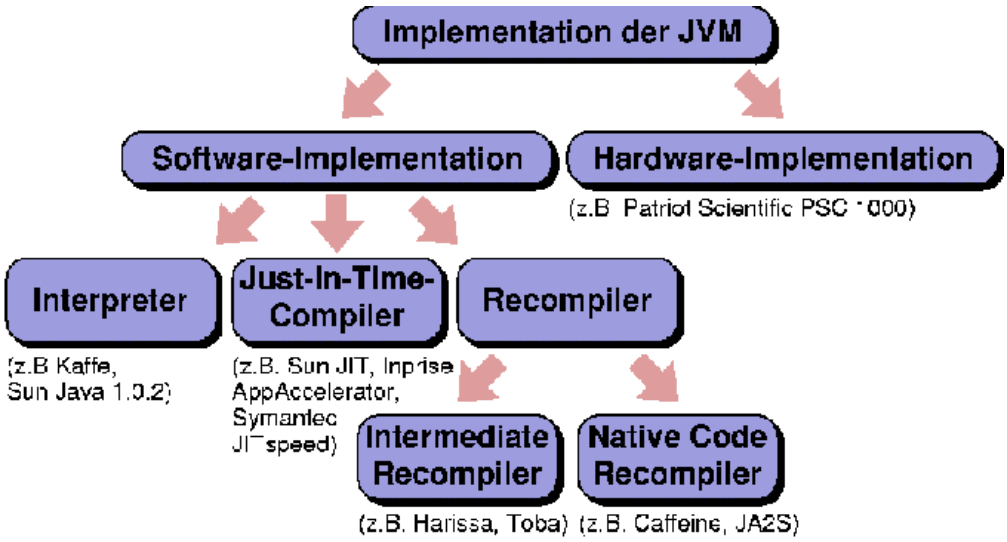
Zuordnung der Quellcodezeilennummern zu den resultierenden Opcodes.

Allgemeine Eigenschaften



Am Ende einer Klassendatei kann eine beliebige Menge allgemeiner Attribute angeben werden.
für das Beispiel wurde durch den Compiler genau ein Attribut erzeugt, ein Verweis auf die Quelldatei (Konstantenpool-Index 0x09). Auf diese Information folgt ein Verweis auf den Namen der Quellcodedatei (Konstantenpool-Index 0x15).

Schlußbemerkungen:



Graphik aus: Raner, M.: Blick unter die Motorehaube

Die interpretative Ausführung einer `class`-Datei mit der virtuellen Java-Maschine ist jedoch keineswegs zwingend, auch wenn sie das derzeit am häufigsten anzutreffende Vorgehen verkörpert. Bereits in der Standardedition des Java Development Toolkits von SUN wird seit Version 1.2 ein *just in time compiler* mitgeliefert, der transparent in die Ausführungsumgebung integriert ist. Er durchbricht die befehlweise interpretative Abarbeitung und greift den bei der Abarbeitung dynamisch durch den Interpretationsprozeß entstehenden plattformspezifischen Maschinencode ab und

puffert ihn zwischen. Bei jeder erneuten Ausführung derselben Bytecodesequenz wird nun dieser bereits übersetzte Code ausgeführt. Dieses Vorgehen ist in den verbreiteten JVM-Implementierungen der Internet-Browser von Netscape und Microsoft verwirklicht. Ebenso bieten fast alle verfügbaren Java-Entwicklungsumgebungen diese Laufzeit-optimierende Komponente an. Der dadurch erzielte Geschwindigkeitsvorteil bewegt sich, je nach Struktur der Anwendung, zwischen zehn und 50 Prozent.
Den größten Gewschwindigkeitszuwachs verspricht man sich jedoch von der vollständigen Realisierung der virtuellen Maschine in Hardware; damit wird sie de facto zur realen Maschine. Hierzu liegen jedoch noch keine Ergebnisse vor, welche die der derzeitigen Implementierung auf handelsüblichen Prozessoren signifikant überträfen.
Eine andere Sichweise nutzt das Bytecodeformat welches eine Zwischenrepräsentation darstellt nicht zur Interpretation mit dem Ziele der direkten Ausführung, sondern als Eingabeformat eines weiteren Übersetzungsschrittes, der üblicherweise plattformabhängigen nativen Code erzeugt. Für C++ existieren bereits Umsetzungen, die Bytecode in übersetzungsfähigen C++-Quellcode transformieren. Eine Spielart hiervon bildet das diskutierte Werkzeug `javap` dessen Ausgabeformat, nach einigen Umformumgen, direkt als Eingabe weiterer Übersetzer akzeptiert wird.

Web-Referenzen 1: Weiterführende Links

- Inside the Java Virtual Machine

3.2 Die Java API und weiterführende Themen

3.2.1 Ein-/Ausgabe -- Streams

Zur Erinnerung: **bisher behandelte Möglichkeiten zur Ein- und Ausgabe**:
Bisher wurden ausschließlich Bildschirm-Ausgaben, und diese ausschließlich mit der statischen Methode `System.out.println`, erzeugt werden. Der einzige uns bisher bekannte Mechanismus zur Eingabebehandlung war der der Kommandozeilenparameter in der `main`-Methode.

Wie aus C++ bekannt verfügt auch Java über die objektorientierte Kapselung der Ein- und Ausgabebehandlung in Form von *Streams*. Hierbei stehen beliebigste Ein- und Ausgabequellen über denselben programmiersprachlichen Mechanismus zur Verfügung, unabhängig davon wie das physische Gerät realisiert ist.

Zentrales Paket für die Ein-/Ausgabebehalung ist `java.io`. Es enthält neben den wichtigsten Klassen zur Implementierung des E/A-Verhaltens auch verschiedene Schnittstellen, sowie die möglichen Ausnahmeereignisse während der verschiedenen E/A-Operationen.
Gegenüber den aus C++ bekannten Datenströmen tritt bei Java hinzu, daß auch Informationen über Netze mit denselben Mechanismen übertragen werden.

In Java werden generelle zwei Streamtypen unterschieden: **Character Streams** und **Byte Streams**. Wie der Name schon andeutet, sind Byte Streams auf die Verarbeitung von beliebigen Byte-artigen Informationseinheiten -- und damit acht Bit große Einheiten -- beschränkt. Diese Mimik stellt insbesondere bei der Verarbeitung von Unicode-Zeichenketten eine große Einschränkung dar, da hierbei nicht gesamte Zeichen-Information (ein Symbol mißt 16 Bit) in einem Zugriff verarbeitet werden kann. Daher wurde mit dem JDK v1.1 zusätzlich das Konzept der Character Streams eingeführt, die generell 16 Bit lange Zeichen bereitstellen.

Als **Byte Streams** stehen zur Verfügung:
(Einrückungen kennzeichnen Subklassenbeziehungen, Kursivsetzungen abstrakte Klassen)

- *`InputStream`*
  Abstrakte Basisklasse der meisten Eingabeströme. Sie definiert die Signatur die grundlegenden Eingabemethoden.
  - `FileInputStream`
    Byteweises lesen einer Datei.
  - `PipedInputStream`
    Hochsprachliches Äquivallent der Betriebssystem Pipes. Einsatzfeld: Gemeinsam mit PipedOutputStream zum Datenaustausch zwischen verschiedenen Threads derselben Applikation.
  - `FilterInputStream`
    Gemeinsam mit anderen Eingabeströmen eingesetzt. Einsatzgebiet: Nachbearbeitung roher Eingaben. Die entsprechenden Subklassen stellen konrete Implementierungen häufiger Einsatzfälle dar.
    - `LineNumberInputStream`*deprecated*
      Liefert zur Eingabe die zugehörige Zeilennummer des Eingabestroms.
      Achtung: Diese Klasse geht von der (i. A. falschen) Entsprechung zwischen Bytes und Charactern aus. `LineNumberReader` liefert für denselben Anwendungsfalle eine korrekte Implementierung.
    - `DataInputStream`
      Liest primitive Java-Datentypen plattformunabhängig aus einem Datenstrom.
    - `BufferedInputStream`
      Verleiht beliebigen `InputStream`s die Möglichkeit des gepufferten Lesens aus dem Eingabestrom. Zusätzlich wird das Setzen von, und die Rücksetzung des Strompositionszeigers zu, Markierungspunkten unterstützt.
    - `PushbackInputStream`
      Verleiht beliebigen `InputStream`s die Möglichkeit der Wiedereinstellung von bereits gelesenen Byte-Inhalten in den Eingabestrom.
  - `ByteArrayInputStream`
    Lesen aus einem Byte Array.
  - `StringBufferInputStream`*deprecated* Lesen aus String.*Achtung*: Diese Klasse geht von der (i. A. falschen) Entsprechung zwischen Bytes und Charactern aus. StringReader liefert für denselben Anwendungsfalle eine

korrekte Implementierung.
  - SequenceInputStream
    Konkatenation von Eingebströmen; nach vollständigem Lesen des *i*-ten Eingebstroms wird der *i+1*-te angesprochen.
  - ObjectInputStream
    Lesen vollständig serialisierter Objekte, die die Schnittstellen java.io.Serializable oder java.io.Externalizable unterstützten.
- *OutputStream*
  Abstrakte Basisklasse der meisten Ausgabeströme. Sie definiert die Signatur die grundlegenden Ausgabemethoden.
  - FileOutputStream
    Schreibt Byte-artige Daten in Dateien oder durch Dateideskriptoren bezeichnete Ziele.
  - PipedOutputStream
    Umkehrung (Schreibende) eines PipedInputStreams.
  - FilterOutputStream
    Verleiht Ausgabestömen die Möglichkeit Ausgabedaten zur Verändern. Einige konkrete Implementierungen sind durch die Subklassen gegeben.
    - DataOutputStream
      Erlaubt plattformunabhängiges Schreiben der primitiven Java-Datentypen.
    - BufferedOutputStream
      Verleiht verschiedenen OutputStreams die Möglichkeit des gepufferten (d.h. nicht mehr Byte-weisen) Schreibens.
    - PrintStream
      Verleiht anderen OutputStreams diverse Möglichkeiten zur Ausgabe der verschiedenen Java Datentypen. Anders als die übrigen Stromtypen erzeugt PrintStream keine IOException.
      Anmerkung: Es werden Bytes, keine Characters geschrieben. Mit PrintWriter existiert eine Standardimplementierung, die auf die Besonderheiten im Zusammenhang mit Unicode-Ausgaben Rücksicht nimmt.
  - ByteArrayOutputStream
    Ausgabestrom, der in einen Byte-Array schreibt.
  - ObjectOutputStream
    Schreibt Primitivtypen und Objekte. Ausschließlich Objekte, welche die java.io.Serializable-Schnittstelle implementieren können über diesen Mechanismus persistent geschrieben werden.

Als **Character Streams** stehen zur Verfügung:
(Einrückungen kennzeichnen Subklassenbeziehungen, Kursivsetzungen abstrakte Klassen)

- *Reader*
  Abstrakte Basisklasse der verschiedenen Reader-Implementierung. Sie definiert die Signatur der grundlegenden Character-bezogenen Ausgabemethoden.
  - BufferedReader
    Liest Textinhalte gepuffert aus einem Character-Eingabestrom.
    - LineNumberReader
      Liest einzelne Unicode-Zeichen, Zeichensequenzen und ganze Zeilen gepuffert, und stellt zusätzlich die Zeilennummer des aktuell verarbeiteten Inhaltes.
  - CharArrayReader
    Einlesen von Unicode-Zeichen aus Character-Puffer.
  - InputStreamReader
    - FileReader
      Eingabestrom für character-Dateien.
  - FilterReader
    Abstrakte Basisklasse verschiedener filternder Eingabeströme. Konkrete Implementierungen werden durch die verschiedenen Subklassen zur Verfügung gestellt.
    - PushbackReader
      Unicode-Character Eingabestrom, der die Rückstellung von bereits gelesenen Inhalten in den Strom erlaubt.
  - PipedReader
    Unicode-Character Eingabestrom als Lese-Ende einer Pipe.
  - StringReader
    Unicode-Character Eingabestrom, der aus einem String liest.
- *Writer*
  Abstrakte Basisklasse verschiedener Implementierungen zur Ausgabe von Unicode-Zeichenströmen.
  - BufferedWriter
    Gepuffertes Schreiben. Zeilenvorschub wird transparent plattformabhängig umgesetzt. (siehe system properties).
  - CharArrayWriter
  - OutputStreamWriter
    Brücke zwischen Character und Byte-Strömen; Unicode-Zeichen werden automatisch während des Schreibens in die entsprechende Byte-Repräsentation umgesetzt.
    - FileWriter
      Schreibt Character-Dateien.
  - *FilterWriter*
    Abstrakte Basisklasse der verschiedenen Implementierungen zum gefilterten Schreiben.
  - PipedWriter
    Ausgabestrom über Character-Pipe.
  - StringWriter
    Ausgabestrom, der in einen String schreibt.
  - PrintWriter
    Ausgabestrom der formatierte Objektrepräsentationen schreibt. Methoden dieser Klasse verursachen keine E/A-Ausnahmeereignisse.

Üblicherweise existieren die Ströme immer symmetrisch, d.h. in gleicher Weise sowohl für Ein- als auch Ausgabe. Dieses Prinzip wird nur für einzelne Klassen durchbrochen, die zwar Eingebeseitig existieren (beispielsweise Lesen mit Zeilennummer), für die jedoch kein expliziter Ausgabemechanismus benötigt wird.

Zusätzlich zu den nach Zugriffsarten klassifizierten Strömen existiert mit der Klasse `RandomAccessFile` eine Strom über den sowohl lesende als auch schreibende Zugriffe abgewickelt werden können.

Die abstrakten Basisklassen `InputStream` bzw. `Reader` und definieren ähnliche Lese- und Zugriffsmethoden, die in allen abgeleiteten Klassen auf den entsprechenden Stromtypen zur Verfügung stehen.
Die aktuell gewünschte Zugriffsart (nur-lesend, nur-schreibend oder beides) wird über einen Parameter des Konstrukturs gesteuert.
Wie durch den Klassennamen bereits angedeutet, existiert dieser Strom ausschließlich für Dateien; eine Netzwerkanwendung ist nicht möglich.

grundlegende Lesemethoden:

- `java.io.InputStream`:
  - `int available()` Liefert die Anzahl Bytes die an der Eingabeschnittstelle zur Verfügung stehen. Diese Anzahl kann ohne Blockierung des Aufrufers gelesen werden.
  - `void close()` Schließt Eingabestrom unter Freigabe der belegten Systemressourcen
  - `void mark(int)` Markiert die gegenwärtige Position des Eingabezeigers im Eingabestrom; ein folgender Aufruf von `reset` setzt den Eingabezeiger wieder an die markierte Position zurück.
  - `boolean markSupported()` Gibt Auskunft darüber, ob der Eingabestrom die Positonsmarkierung, und das Rücksetzen darauf, unterstützt.
  - `int read()` Ließt den nächsten Bytewert (`>0` und `<256`) aus dem Eingabestrom. Ist das Ende des Eingabestromes erreicht, wird `-1` retourniert (kein Ausnahmeereignis! Die Verwendung des `StreamTokenizer`s ermöglicht hier ein handlicheres Vorgehen).
    Subklassen von `InputStream` sind gezwungen diese Methode zu überschreiben.
  - `int read (byte[])` Ließt eine Bytesequenz aus dem Eingabestrom, und legt sie im übergebenen Array ab. Die Anzahl der gelesenen Zeichen, bzw. `-1` beim Erreichen den Eingabeendes, wird zurückgegeben.
  - `int read(byte[] b, int off, int len` Ließt Bytesequenz der Länge `len` und speichert sie ab Position `off` im übergebenen Array.
  - `void reset()` Setzt den Stromzeiger auf die Position der letzten vorhergehenden Markierung zurück, falls eine solche existiert.
  - `long skip(long n)` Versucht den Stromzeiger um `n` Bytepositionen vorzurücken. Die Anzahl der tatsächlich übersprungenen Bytes wird zurückgegeben.
- `java.io.Reader`:
  - `void close()` Schließt den Ausgabestrom.
  - `void mark(int readAheadLimit)` Markiert gegenwärtige Position des Eingabezeigers. Ein nachfolgender Aufruf von `reset` versucht den Positionszeiger auf die Markierte Stelle zurückzusetzen, falls die aktuelle Position nicht weiter als `readAheadLimit` Zeichen entfernt liegt.
  - `void markSupported()` Gibt Auskunft darüber, ob der Eingabestrom die Positonsmarkierung, und das Rücksetzen darauf, unterstützt.
  - `int read()` zur Extraktion genau eines Zeichens (16 Bit Character).
  - `int read(char[] cbuf)` zur Extraktion einer Sequenz, beginnend ab der aktuellen Stromzeigerpositon.
  - `int read(char[] cbuf, int off, int len)` zur Extraktion einer Sequenz der Länge `len` oder weniger von Zeichen, und Ablage in Array beginnend ab der Position `off`.
  - `boolean ready()` Liefert dieser Aufruf `true` zurück, so liegen weitere Eingaben vor, die durch ein folgendes `read` gelesen werden können.
  - `reset()` Versucht den Positionszeiger auf die durch die letzte gesetzte Markierung bezeichnete Stelle zurückzurücken.
  - `long skip(long n)` Versucht `n` Zeichenpositionen zu überspringen. Die Anzahl der tatsächlich übersprungenen Positionen wird zurückgeliefert.

grundlegene Schreibmethoden:

- `java.io.OutputStream`:
  - `void close()` Schließt Ausgabestrom und gibt durch ihn belegte Systemressourcen frei.
  - `void flush()` Leert Ausgabestrom und schreibt alle Pufferbereiche.
  - `void write(byte[])` Schreibt Byte-Array in Ausgabestrom.
  - `void write(byte[] b, int off, int len)` Schreibt `len` Bytes eines Byte-Arrays ab Position `off`.
  - `void write(int)` Schreibt Bytewert in Ausgabestrom.
    Subklassen von `OutputStream` müssen diese dieser Methode überschreiben.
- `java.io.Writer`:
  - `void close()` Schließt Ausgabestrom nach erfolgter Pufferleerung (flushing).
    Subklassen von `Writer` müssen diese Methode überschreiben.
  - `void flush()` Leert Pufferbereiche.
  - `write(int)` Schreibt ein Unicode-Zeichen.
  - `write(char[])` Schreibt Character-Array.
  - `write(char[] cbuf, int off, int len)` Schreibt Teil der Länge `len` eines Character-Arrays beginnend ab Position `off`.
  - `write(String)` Schreibt `String`.
  - `write(String str, int off, int len)` Schreibt Teil der Länge `len` eines `String`s beginnend ab Position `off`.

Mit den Dateideskriptorenin, `out` und `err` stehen die aus C/C++ bekannten drei **Standardstörme** zur Verfügung. Diese standardmäßig geöffneten Ströme stehen während der Ausführungzeit jeder Applikation zur Verfügung.

```
(1)import java.io.FileDescriptor;
(2)import java.io.FileWriter;
(3)import java.io.IOException;
(4)
(5)public class PrintLn {
(6)     public static void main(String[] args) {
(7)             FileWriter fw = null;
(8)             try {
(9)                     fw = new FileWriter(FileDescriptor.out);
(10)                    for (int i=0; i < args.length; i++)
(11)                            fw.write (args[i]+" ");
(12)            } catch (IOException ioe) {
(13)                    System.out.println("cannot open stdout!");
(14)            } finally {
(15)                    try {
(16)                            fw.close();
(17)                    } catch (Exception e) {
(18)                            //ignore it
(19)                    } //catch
(20)            } //finally
(21)    } //main()
(22)} //class PrintLn
```

Beispiel 57: Ausgabe auf standard out PrintLn.java

Das Programm öffnet erzeugt ein `FileWriter`-Objekt mit dem vorgegebenen Dateideskriptor `out`.
Anschließend werden die Kommandozeilenparameter (allesamt Typ `String`) mit `write` ausgegeben.
Alle Methoden der Klasse `FileWriter` können ein Ausnahmeereigniss vom Typ `IOException` erzeugen. Daher müssen Operationen auf dem erzeugten Ausgabestrom durch `try`-Blöcke abgesichert werden.

Wird als Ausgabekanal des `FileWriter`-Stroms auf eine pysikalische Datei ausgerichtet, so muß lediglich die Konstruktoranweisung zu `fw = new FileWriter("myFile.asc")` modifiziert werden.
Alle E/A-Klassen interpretieren die Pfadangabe relativ zum aktuellen Verzeichnis. Die Verzeichnisseparatoren variieren plattformabhängig. Verzeichnistrenner und aktueller Katlog können über die *system properties* ermittelt werden.

Streamerzeugung und mögliche Datenquellen aller (nicht als deprecated gekennzeichneten) Streamtypen:

Strom	**Erzeugbar aus**
BufferedInputStream	`InputStream`
BufferedOutputStream	`OutputStream`
BufferedReader	`Reader`
BufferedWriter	`Writer`
ByteArrayInputStream	Byte Array
CharArrayReader	Character Array
DataInputStream	`InputStream` Beispiel
DataOutputStream	`OutputStream`
FileInputStream	`File`-Objekt `FileDescriptor` (nur auf die Standardströme stdin, stdout, stderr andwendbar) `String`, der einen gültigen Pfad enthält
FileOutputStream	`File`-Objekt, `FileDescriptor` (nur auf die Standardströme stdin, stdout, stderr andwendbar) `String`, der einen gültigen Pfad enthält
FileReader	`File`-Objekt, `FileDescriptor` (nur auf die Standardströme stdin, stdout, stderr andwendbar) `String`, der einen gültigen Pfad enthält
FileWriter	`File`-Objekt, `FileDescriptor` (nur auf die Standardströme stdin, stdout, stderr andwendbar) `String`, der einen gültigen Pfad enthält
FilterInputStream	`InputStream`
FilterOutputStream	`OutputStream`
FilterReader	`Reader`
InputStreamReader	`InputStream`

LineNumberReader	Reader
ObjectInputStream	InputStream
ObjectOutputStream	OutputStream
OutputStreamWriter	OutputStream
PipedInputStream	PipedOutputStream Der parameterlose Vorgabekonstruktor erzeugt einen unverbundenen Strom.
PipedOutputStream	PipedInputStream Der parameterlose Vorgabekonstruktor erzeugt einen unverbundenen Strom.
PipedReader	PipedWriter
PipedWriter	PipedReader
PrinterWriter	OutputStream Writer
PrintStream	OutputStream
PushbackInputStream	InputStream
PushbackReader	Reader
SequenceInputStream	InputStream Oder Inhalt eines Objekts dessen Klasse die Enumeration-Schnittstelle implementiert.
StringReader	String

Der LineNumberReader liefert ein Beispiel eines Stroms, der auf Basis eines anderen Stroms definiert wird. Im Falle des folgenden Beispiels wird ein LineNumberReader ausgehend von einem bestehenden FileReader erzeugt.

Wie bereits im zweiten Kapitel angesprochen unterstützt Java den Unicode-Zeichensatz. Durch ihn wird die plattformübergreifende Darstellung verschiedenster Zeichensätze ermöglicht. Als Erweiterung des klassischen ISO 8859-Teil 1 Zeichensatzes benötigt er jedoch generell 16 Bit zur Darstellung eines Symbols. Zusätzlich ist zu einem Unicode codierten Datenstrom die Codierungsschema, identifiziert durch ein eindeutiges Kürzel, anzugeben um die korrekte Darstellung zu ermöglichen.
Hierzu erlauben die Stream-Klassen InputStreamReader und OutputStreamWriter die explizite Spezifikation der Eingabe- bzw. Ausgabe Encodierung.
Jede Java-Implementierung muß mindestens folgende Code-Formate unterstützen: US-ASCII, ISO-8859-1, UTF-16BE (16-Bit Unicode im big-endian Format), UTF-16LE (dergleichen als little-endian) und UTF-16 (allgemeines 16-Bit Unicodeformat, *byte order mark* am Beginn des Stroms definiert verwendetes Anordnungsschema).

```
(1)import java.io.FileDescriptor;
(2)import java.io.FileOutputStream;
(3)import java.io.FileReader;
(4)import java.io.IOException;
(5)import java.io.LineNumberReader;
(6)import java.io.OutputStreamWriter;
(7)
(8)public class UnicodeWriter {
(9)     public static void main(String[] args) {
(10)            LineNumberReader lnr = null;
(11)            OutputStreamWriter osw = null;
(12)
(13)            String encoding, text;
(14)
(15)            try {
(16)                    System.out.print("specify encoding:");
(17)                    lnr = new LineNumberReader(new FileReader(FileDescriptor.in));
(18)                    encoding = lnr.readLine();
(19)
(20)                    System.out.print("Specify text to encode:");
(21)                    text = lnr.readLine();
(22)
(23)                    System.out.print("encoded text:");
(24)
(25)                    osw = new OutputStreamWriter((new FileOutputStream(FileDescriptor.out)),
encoding);
(26)                    osw.write(text);
(27)            } catch (IOException ioe) {
(28)                    System.out.println("an IOException occurred\n"+ioe.getMessage() );
(29)            } finally {
(30)                    try {
(31)                            lnr.close();
(32)                            osw.close();
(33)                    } catch (Exception e) {
(34)                            //ignore it
(35)                    } //catch
(36)            } //finally
(37)    } //main()
```

```
(38)} //class UnicodeWriter
```

Beispiel 58: Schreiben in frei wählbarem Ausgabeencoding UnicodeWriter.java

Das Programm ließt zunächst von der Standardeingabe die gewünschte Encodingdefinition und den auszugebenen Text als Zeichenkette.
Dann wird ein Ausgabestrom auf die Standardausgabe erzeugt, der das zuvor spezifizierte Encoding verwendet. Unterstützt die Java-Implementierung das angegebene Codierungsschema nicht, schlägt die Erzeugung des Ausgabestroms fehl, und es wird ein Ausnahmeereignis erzeugt.
Zum Abschluß wird der Text unter Anwendung des definierten Encodingschemas über den Ausgabestrom ausgegeben.

Beispielinteraktionen:

```
specify encoding:US-ASCII
Specify text to encode:abcäöüß
encoded text:abc????

specify encoding:UTF8
Specify text to encode:aä
encoded text:aÔÇ×

specify encoding:UTF-16LE
Specify text to encode:test
encoded text:t e s t
```

```
(1)import java.io.IOException;
(2)import java.io.FileDescriptor;
(3)import java.io.LineNumberReader;
(4)import java.io.FileReader;
(5)import java.io.FileWriter;
(6)
(7)
(8)public class Type {
(9)     public static void main(String[] args) {
(10)            FileReader fr = null;
(11)            FileWriter fw = null;
(12)            LineNumberReader lnr = null;
(13)            String line;
(14)
(15)            try {
(16)                    fr = new FileReader(args[0]);
(17)                    lnr = new LineNumberReader (fr);
(18)                    fw = new FileWriter(FileDescriptor.out);
(19)
(20)                    while ( (line = lnr.readLine()) != null) {
(21)                            fw.write( lnr.getLineNumber()+": "+line+"\n" );
(22)                    }//while
(23)            } catch (IOException ioe) {
(24)                    System.out.println("an IOException occurred");
(25)                    System.out.println( ioe.getMessage() );
(26)            } finally {
(27)                    try {
(28)                            fr.close();
(29)                            fw.close();
(30)                    } catch (IOException ioe) {
(31)                            //ignore it
(32)                    } //catch
(33)            } //finally
(34)    } //main()
(35)} //class Type
```

Beispiel 59: Zeilenweise Ausgabe einer Datei incl. Zeilennummern Type.java

---

Serialisierung von Objekten
Durch den Stromtyp `ObjectOuputStream` können vollständige Objekte geschrieben, und durch `ObjectInputStream` wieder in den Speicher eingeladen werden.

```
(1)import java.io.FileInputStream;
(2)import java.io.FileOutputStream;
(3)import java.io.IOException;
(4)import java.io.ObjectInputStream;
(5)import java.io.ObjectOutputStream;
(6)import java.io.Serializable;
(7)import java.util.Calendar;
(8)import java.util.GregorianCalendar;
(9)
(10)public class SerializeData {
(11)    public static void main(String[] args)  {
(12)            //just for determining the current year
(13)            GregorianCalendar gregCal = new GregorianCalendar();
(14)
(15)            Person hans = new Person();
(16)            hans.name = new String("hans");
(17)            hans.yearOfBirth = 1950;
(18)            hans.age = gregCal.get(Calendar.YEAR) - hans.yearOfBirth;
(19)
(20)            System.out.println("object before serialization:\n" + hans.toString() );
(21)
(22)            ObjectOutputStream oos = null;
(23)
(24)            try {
(25)                    oos = new ObjectOutputStream( new FileOutputStream("hans") );
(26)                    oos.writeObject ( hans );
(27)            } catch (IOException ioe) {
(28)                    System.out.println("an IOException occurred");
(29)                    System.out.println( ioe.getMessage() );
(30)            } finally {
(31)                    try {
(32)                            oos.close();
(33)                    } catch (IOException ioe) {
(34)                            //ignore it
(35)                    } //catch
(36)            } //finally
(37)
(38)            gregCal = null;
(39)            hans = null;
(40)
(41)            ObjectInputStream ois = null;
(42)            try {
(43)                    Person anotherOne;
(44)                    ois = new ObjectInputStream( new FileInputStream("hans") );
(45)                    anotherOne = (Person) ois.readObject();
(46)
(47)                    System.out.println( "object retrieved from file:\n"+ anotherOne.toString
() );
(48)            } catch (IOException ioe) {
(49)                    System.out.println("an IOException occurred while reading back object");
(50)            } catch (ClassNotFoundException cnfe) {
(51)                    System.out.println("could not find class Person");
(52)            } finally {
(53)                    try {
(54)                            ois.close();
(55)                    } catch (IOException ioe) {
(56)                            //ignore it
(57)                    } //catch
(58)            } //finally
(59)    } //main()
(60)} //class serializeData
(61)
(62)class Person implements Serializable {
(63)    public int yearOfBirth;
(64)    public String name;
(65)    transient int age;
(66)
(67)    public void finalize() {
(68)            System.out.println("object destroyed");
(69)    } //finalize()
(70)
(71)    public String toString() {
(72)            return("name="+name+"\n"+"yearOfBirth="+yearOfBirth+"\n"+"age="+age);
(73)    }//toString()
(74)} //class Person
```

Beispiel 60: Serialisieren und Laden eines Objekts SerializeData.java

Bildschirmausgabe:

```
object before serialization:
name=hans
yearOfBirth=1950
age=50
object retrieved from file:
name=hans
yearOfBirth=1950
age=0
```

Das transiente Attributage wird nicht in die Datei übernommen. Die Inhalte aller anderen Attribute werden gesichert, und können rückgelesen werden.
Beim (Wieder-)Einlesen eines Objektes wird versucht dessen Klassendefinition aus der entsprechenden Klassendatei zu laden. Ist dies nicht möglich, so wird ein Ausnahmeereignis vom Typ `ClassNotFoundException` generiert.

Anmerkung: Der Java-Serialisierungsmechanismus verhindert die mehrfache Ablage desselben Objektes im Bytestrom.

Für die häufig umzusetzende Aufgabe der Eingabeformatprüfung, und anschließenden Klassifizierung in bestimmte Kategorien steht die Klasse `StreamTokenizer` zur Verfügung.

```
(1)import java.io.StreamTokenizer;
(2)import java.io.FileReader;
(3)import java.io.FileDescriptor;
(4)import java.io.IOException;
(5)
(6)public class TokenTest {
(7)     public static void main(String[] args) {
(8)             StreamTokenizer st = null;
(9)
(10)            int     op1=0,
(11)                            op2=0;
(12)            try {
(13)                    st = new StreamTokenizer(new FileReader(FileDescriptor.in));
(14)
(15)                    while(st.nextToken() == StreamTokenizer.TT_NUMBER)
(16)                            op1 = (op1*10) +(int) st.nval;
(17)
(18)                    while(st.nextToken() == StreamTokenizer.TT_NUMBER)
(19)                            op2 = (op2*10) + (int) st.nval;
(20)
(21)                    System.out.println(op1+"+"+op2+"="+(op1+op2));
(22)            } catch (IOException ioe) {
(23)                    System.out.println("an IOException occurred\n"+ioe.getMessage() );
(24)            } //catch
(25)    } //main()
(26)} //class TokenTest
```

Beispiel 61: Einfache Addition zweier Zahlen unter Verwendung des StringTokenizers TokenTest.java

Die beiden Operanden können durch ein beliebiges nicht numerisches Zeichen voneinander abgetrennt werden, abenso kann das Bereichnungsende erklärt werden.

Zum Zugriff auf **(G-)ZIP komprimierte Dateien** bieten die Klassen `ZipInputStream` und `GZIPInputStream` bzw. `ZipOutputStream` und `GZIPOutputStream` Lese- bzw. Schreibströme an.

3.2.2 Threads und Nebenläufigkeit

Java bietet mit den sog. Programmfäden engl. *Threads* die Möglichkeit an, paralle leichtgewichtige Prozesse direkt in der Hochsprache zu definieren und zu kontrollieren. Hierbei werden keine Anforderungen an eine spätere Unterstützung dieses Konzepts durch die tatsächliche physische Hardware gestellt; der gesamte Mechanismus ist rein Hochsprachen-basiert.
Als Bestandteil des automatisch importierten Paketes `java.lang` stehen Threads in jeder Applikation und jedem Applet ohne zusätzliche Aufwende zur Verfügung.

Weiterführende Informationen: Java Language Specification, chap. 17 und Java JVM Spezifikation, chap. 8.

Inhaltlich unterscheidet sich ein Thread nur in marginalen Modifikationen von einer gewöhlichen Klasse. Hauptunterschied ist die eigenständige aktive und nebenläufige Ausführung von Objekte einer solchen Klasse.

Voraussetzungen zur Erzeugung eines Threads:

- Die Klasse spezialisiert die Standard-API-Klasse `Thread`.
  Wie alle Klassen, die nebenläufig ausgeführt werden sollen implementiert auch `Thread` die Schnittstelle `Runnable`.

Sie definiert die Operation `run()`.

- Die Klasse überschreibt die Methode `run()`.
  Sie wird explizit durch die Methode `start()` aufgerufen. `start()` ist asynchron und kehrt sofort nach Erzeugung des Threads zum Aufrufer zurück.

Hinweis: Die Methode `run()` sollte nicht direkt aufgerufen werden! Bei der direkten Ausführung unterbliebe die notwendige Initialisierung; insbesondere wäre der dann „gewöhnliche“ Methodenaufruf nicht ansynchron, und das neue Objekt würde nicht nebenläufig ausgeführt.

```
(1)public class Threads1 {
(2)     public static void main(String[] args) {
(3)             HelloThread northGerman = new HelloThread( "Moin Moin" );
(4)             HelloThread southGerman = new HelloThread( "Gruess Gott" );
(5)
(6)             northGerman.start();
(7)             southGerman.start();
(8)     } //main()
(9)} //class Threads
(10)
(11)class HelloThread extends Thread {
(12)    protected String greetingText;
(13)
(14)    public HelloThread (String greetingText) {
(15)            this.greetingText = greetingText;
(16)    } //constructor
(17)
(18)    public void run() {
(19)            while (true) {
(20)                    try {
(21)                            Thread.sleep(500);
(22)                    } catch (InterruptedException ie) {
(23)                            System.out.println("an InterruptedException occurred\n"+ie.toString
()+"\n"+ie.getMessage() );
(24)                    } //catch
(25)                    System.out.println( greetingText);
(26)            } //while
(27)    } //run()
(28)}//class HelloThread
```

Beispiel 62: Zwei einfache Threads Threads1.java

Das Programm gibt die beiden Texte *Moin Moin* und *Gruess Gott* jeweils im Wechsel durch separate Threads aus. Besonders fällt auf, daß die Applikation nach Abarbeiten der letzten Anweisung der `main`-Methode nicht terminiert.

Java-Programme die (noch) *Vordergrund*-Threads ausführen terminieren nicht am Ende der `main`-Methode, sondern führen weiterhin die noch laufenden Threads -- bis zu deren eigenständigem Terminieren oder Abbruch -- aus.
Die zweite Threadklasse bilden die *Daemon*-Threads. Verfügt ein laufendes Programm ausschließlich über solche Hintergrund-Threads terminiert es am Ende der `main`-Methode wie gewohnt.
Demnach läßt sich der bisher bekannte Programmtyp als nebenläufige Applikation mit dem Vordergrundthread `main` betrachten. Terminiert dieser Thread, so terminiert auch die gesamte Applikation.

Hinweis: Programme mit laufenden Vordergrundthreads lassen sich durch Beenden der virtuellen Maschine mittels `System.exit(int)` terminieren.

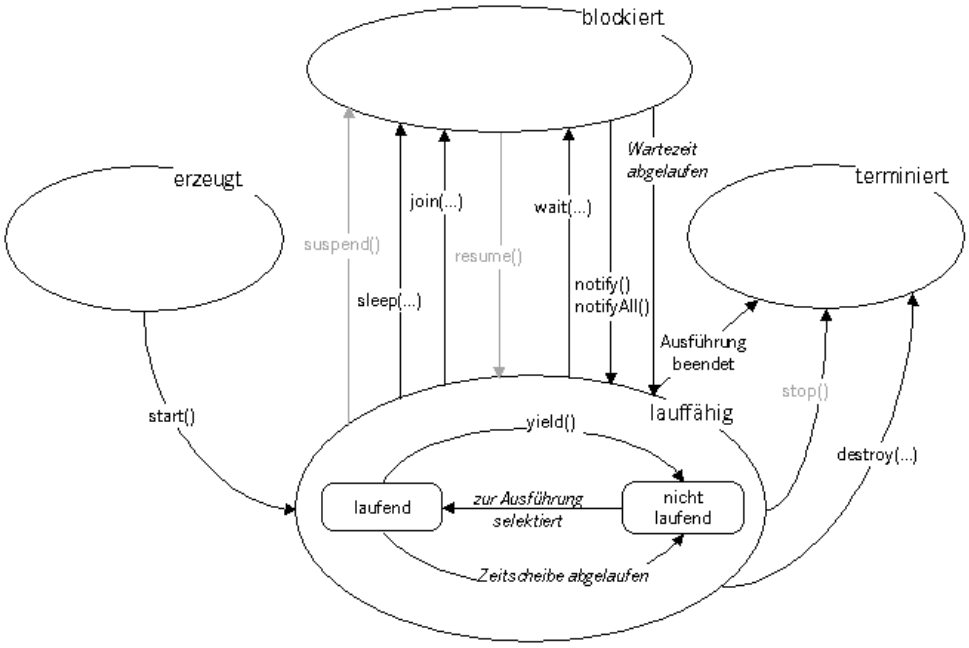
Das Beispiel enthält auch bereits zwei der möglichen Status innerhalb des Lebenszyklus eines Threads, nämlich *erzeugt* (aber noch nicht in Ausführung befindlich, nach Aufruf des Konstruktors) und *in Ausführung* nach dem Starten durch den Aufruf der Methode `run()`.
Die möglichen Threadzustände können jedoch noch durch weitere Methoden beinflußt werden:

- `start()`
  Erzeugung eines Threads (starten) aus einem bereits erstellten Thread-Objekts.
- `sleep(long)` und `sleep(long, int)`
  Anhalten eines Threads für eine durch die Übergabeparameter festgelegte Zeitspanne.
  Während der Ausführungspause bleiben gesetzte Sperren (Monitore) erhalten.
- `join()`, `join(long)` und `join(long, int)`
  Wartet auf das Terminieren des erzeugten Threads. (Quasi synchrone Variante von `run()`). Optional kann eine maximale abzuwartende Zeitspanne definiert werden.
- `yield`
  Freiwillige Abgabe der Zeitscheibe.
- `destroy()`
  Zerstört Thread ohne vorherige Aufräumarbeiten; insbesondere werden gesetzte Sperren nicht freigegeben.

Hinweis:
Die beiden `stop`-Methoden, sowie die Methoden `suspend` und und `resume` sind seit der Version 1.3 als *deprecated* gekennzeichnet und sollten nicht mehr verwendet werden!
Der Grund liegt in der, mit der durch die API zugesicherten, sofortigen Unterbrechung. Dabei kann die Unterbrechung auch innerhalb eines kritischen Abschnittes erfolgen, wodurch es zu verschiedensten Inkonsistenzen und weiteren Folgeproblemen kommen kann.

Hinzu kommen von `java.lang.Object` ererbte Methoden:

- `wait()`
  `wait(long)`
  `wait(long, int)`
  Hält Thread beim Zugriff auf ein bestimmtes Objekt an, bis ein anderer Thread die Methode `notify()` oder `notifyAll()` ausführt.
  Zur Synchronisation werden Monitore verwendet.
- `notify()`
  `notifyAll()`
  Fährt mit der Ausführung eines im Threads fort, der beim Zugriff auf ein gesperrtes Objekt in den Wartezustand versetzt wurde.



Die Graphik zeigt die verschiedenen möglichen Zustände eines Threads. Als *deprecated* gekennzeichnete Methoden sind grau unterlegt.

Das interne Scheduling aller laufenden Threads erfolgt prioritätsgesteuert. Die **Priorität**en zugreifbarer (d.h. eigen erzeugter) Threads kann erfragt und gezielt verändert werden.
Die Threadpriorität wird als ganzzahliger Wert angegeben. Plattformspezifisch kann dieser Wert variieren; jedoch kann die konkrete Unter- und Obergrenze über die Konstanten `MIN_PRIORITY` bzw. `MAX_PRIORITY` zur Laufzeit erfragt werden. Ohne manuellen Eingriff werden neue Threads mit der durch `NORM_PRIORITY` definierten Priorität erzeugt.

Methoden zur Beeinflussung der Thread-Priorisierung:

- `int getPriority()`
  liefert die Priorität des Threads.
- `setPriority(int)`
  versucht die Priorität auf den übergebenen Wert zu setzen. Dies kann wegen mangelnder Berechtigung, oder ungültiger Übergabeparameter, fehlschlagen. In diesem Falle wird ein Ausnahmeereignis der Klasse `SecurityException`, bzw. -- bei ungültigen Übergabeparametern -- `IllegalArgumentException` erzeugt.

Aus Gründen der vereinfachten Verwaltung können Threads in Gruppen (API-Klasse `ThreadGroup`) zusammengefaßt und damit gebündelt beeinflußt werden.
Operationen zur Zustandsänderung die auf einzelnen Threads wirken, können auch auf **Thread-Gruppen** angewendet werden.
Die Verwaltung von solchen Thread-Bündeln ist in der API-Klasse `ThreadGroup` zusammengefaßt; die Gruppenzugehörigkeit eines spezifischen Threads kann durch `getThreadGroup()` ermittelt werden.

Daemon-Threads:
wie bereits angerissen gibt es neben den „normalen“ Anwenderthreads auch die Klasse der Daemon-Threads.
Ihr Hauptunterscheidungsmerkmal zu den Anwenderthreads ist das Charakteristikum, daß die JVM auch terminiert, wenn sich Threads dieser Kategorie in Ausführung befinden. Dies prädestiniert sie für Hintergrundaufgaben wie Verwaltungs- oder Überwachungstätigkeiten.
Der Typ eines Threads kann vor dessen Ausführungsbeginn durch den `start()`-Aufruf mittels der Methode `setDaemon(boolean)` festgelegt, und zur Laufzeit mit `isDaemon()` jederzeit während der Ausführung erfragt, werden.

```
(1)public class IsPrime {
(2)     public static void main(String[] args) {
(3)             PrimitivePrimeTest ppt;
(4)             StatusInformation myInfo = new StatusInformation();
(5)             myInfo.setDaemon(true); //declare as daemon thread
(6)             myInfo.setPriority( (Thread.NORM_PRIORITY + 2) <= Thread.MAX_PRIORITY ? Thread.
NORM_PRIORITY + 2 : Thread.MAX_PRIORITY );
(7)             myInfo.start();
(8)
(9)             ThreadGroup workerThreads       = new ThreadGroup("worker threads");
(10)
(11)            for (int i = Integer.parseInt(args[0]); i <= Integer.parseInt(args[1]); i++) {
(12)                    ppt = new PrimitivePrimeTest(workerThreads, i, "calculating " +i );
(13)                    ppt.start();
(14)            } //for
(15)
(16)    } //main
(17)} //class IsPrime2
(18)
(19)class PrimitivePrimeTest extends Thread {
(20)    protected int numberToTest;
(21)    boolean earlyExit=false;
(22)
(23)    public PrimitivePrimeTest(ThreadGroup tg, int numberToTest, String threadName) {
(24)            super (tg, threadName); //call to super's class constructor
(25)            this.numberToTest = numberToTest;
(26)    } //constructor
(27)
(28)    public void run() {
(29)            if (numberToTest == 1)
(30)                    earlyExit = true;
(31)            for (int i=2; i < ((int) Math.sqrt(numberToTest))+1; i++) {
(32)                    try {
(33)                            Thread.sleep(1000);
(34)                    } catch (InterruptedException ie) {
(35)                            //ignore it
(36)                    } //catch
(37)                    if (numberToTest % i == 0) {
(38)                            earlyExit=true;
(39)                            break;
(40)                    } //endif
(41)            } //for
(42)            if  (earlyExit != true)
(43)                    System.out.println(numberToTest+" is prime");
(44)    } //run()
(45)} //class PrimitivePrimeTest
(46)
(47)class StatusInformation extends Thread {
(48)    public void run() {
(49)            while (true) {
(50)                    System.out.println("no of currently running threads: "+Thread.activeCount
() );
(51)                    try {
(52)                            Thread.sleep(500);
(53)                    } catch (InterruptedException ie) {
(54)                            //ignore it
(55)                    } //catch
(56)            } //while
(57)    } //run()
(58)} //class StatusInformation
```

Beispiel 63: Ein einfacher Primzahlenprüfer IsPrime.java

Beispielablauf:

```
$java IsPrime 1 25
no of currently running threads: 2
2 is prime
3 is prime
no of currently running threads: 24
no of currently running threads: 24
5 is prime
7 is prime
no of currently running threads: 11
no of currently running threads: 11
11 is prime
13 is prime
no of currently running threads: 6
no of currently running threads: 6
17 is prime
```

```
19 is prime
23 is prime
no of currently running threads: 3
no of currently running threads: 3
```

Das Programm prüft in jeweils einem eigenen Thread, ob die Ganzzahlen im Intervall zwischen den gegebenen Grenzen prim sind. Entdeckte Primzahlen werden mit einer entsprechenden Meldung ausgegeben.
Um die unterschiedliche Ausführungsdauer der jeweiligen Threads hervorzuheben führt jeder Programmfaden nur eine Berechnung pro Sekunde aus.
Die Berechnungsthreads sind alle in einer eigenen Threadgruppe (`workerThreads`) zusammengefaßt. Jeder Thread innerhalb dieser Gruppe wird durch die Zeichenkette *calculating* gefolgt von der zu prüfenden Zahl benannt.
Zusätzlich gibt ein Daemon-Thread alle halbe Sekunde die Anzahl der (noch) aktiven Threads aus. Dieser Thread wird automatisch durch die JVM nach dem Abarbeiten des letzten aktiven Anwenderthreads terminiert. Die Priorität des Daemon-Threads ist um zwei gegenüber dem Vorgabewert erhöhrt, sofern dadurch nicht die maximal erlaubte Priorisierung überschritten wird.

Synchronisation von Methodenzugriffen:

```
(1)import java.io.DataOutputStream;
(2)import java.io.FileOutputStream;
(3)import java.io.FileInputStream;
(4)import java.io.DataInputStream;
(5)import java.io.IOException;
(6)import java.io.FileNotFoundException;
(7)
(8)public class Threads2 {
(9)     public static void main(String[] args) {
(10)            for (int threadCount=0; threadCount < Integer.parseInt(args[0]); threadCount++)
(11)                    (new IncrementCounter()).start();
(12)    } //main()
(13)} //class Threads2
(14)
(15)class IncrementCounter extends Thread {
(16)    public void run() {
(17)            int counterValue;
(18)            DataInputStream dis = null;
(19)            DataOutputStream dos = null;
(20)
(21)            try {
(22)                    for (int i=0; i<10; i++) {
(23)                            dis = new DataInputStream( new FileInputStream( "counter" ) );
(24)                            counterValue = dis.readInt();
(25)                            dis.close();
(26)
(27)                            System.out.println(counterValue+" read by thread "+ (Thread.
currentThread()).getName() );
(28)
(29)                            dos = new DataOutputStream( new FileOutputStream( "counter",
false ) );
(30)                            dos.writeInt( ++counterValue );
(31)                            dos.close();
(32)                    } //for
(33)            } catch (FileNotFoundException fnfe) {
(34)                    System.out.println("file counter could not be opened!\n"+fnfe.toString()
+"\n"+fnfe.getMessage() );
(35)                    System.exit(1);
(36)            } catch (IOException ioe) {
(37)                    System.out.println("an IOException occurred in thread "+(Thread.
currentThread()).getName()+"!\n"+ioe.toString()+"\n"+ioe.getMessage() );
(38)                    System.exit(1);
(39)            } //catch
(40)    } //run()
(41)} //class IncrementCounter
```

Beispiel 64: Gemeinsames Inkrementieren eines Zählers in einer Datei Threads2.java

Hilfsprogramme: manuelles Rücksetzen des Zählerstandes auf 0, manuelles Auslesen des Zählerstandes und Ausgabe auf Standardausgabe

Das Programm liest den Stand einer ganzzahligen Variable aus einer Datei aus, erhöht um Eins und schreibt ihn zurück in dieselbe Datei, unter Verlust des alten Standes (Überschreiben). Die beschriebene Vorgangsfolge wird im Multithreading-Betrieb durch mehrere Ausführungsfäden nebenläufig durchgeführt.
Beim Auffangen eines Ausnahmeereignisses wird die virtuelle Maschine terminiert, da das einzelne Terminieren nur eines Threads das Ergebnis verfälschen würde.
Tritt zwischen dem Auslesevorgang aus der Datei, und dem Rückschreiben des modifizierten Wertes ein Threadwechsel ein, so kommt es zum Phänomen des *lost updates*.

mögliche Bildschirmausgaben:

Nach dem Einlesen der Zahl 5 durch Thread-33 tritt, vor ihrem inkrementierten Rückschreiben, der Threadwechsel ein, wodurch der nachfolgend ausgeführte Thread-34 dieselbe Zahl nochmals ausliest.

```
$java Threads2
...
3 read by thread Thread-28
4 read by thread Thread-30
5 read by thread Thread-33
5 read by thread Thread-34
6 read by thread Thread-42
7 read by thread Thread-45
...
```

Sicherlich ein Extremum dieses Verhaltens stellt das nachfolgende Beispiel dar:

mögliche Bildschirmausgaben:
Threadwechsel tritt jeweils direkt nach dem Einlesen auf. Als Konsequenz wird derselbe Zahlenwert mehrfach durch verschiedene Threads gelesen.

```
$java threads2
...
10 read by thread Thread-35
10 read by thread Thread-10
10 read by thread Thread-36
10 read by thread Thread-37
10 read by thread Thread-14
10 read by thread Thread-38
10 read by thread Thread-12
10 read by thread Thread-39
10 read by thread Thread-16
10 read by thread Thread-41
10 read by thread Thread-17
10 read by thread Thread-19
10 read by thread Thread-42
10 read by thread Thread-18
10 read by thread Thread-40
10 read by thread Thread-0
10 read by thread Thread-11
...
```

Um Daten-Inkonsistenzen zur Laufzeit zu verhindern, wie sie potentiell entstehen könnten, wenn zwei Programmfäden gleichzeitig dieselbe Methode ausführen oder zeitlich verschränkt dieselbe kritische Ressource benutzen, ist mit dem (aus Kapitel zwei bekannten) Schlüsselwort synchronized ein Hochsprachenmechanismus gegeben um mehrere Aufrufer gezielt zu serialisieren.
Technisch handelt es sich dabei um das aus den Betriebsystemen bekannte Konzept der Monitore (siehe C. A. R. Hoare: *Monitors: An Operating System Structuring Concept*) zur Synchronisation des Zugriffs auf kritische Abschnitte.

Jedoch besteht bei dieser Vorgehensweise die Gefahr eines **Deadlocks** durch wechselseitiges Blockieren!

Durch synchronisierte Blöcke können nicht nur einzelne Methoden, sondern Bündel von Zugriffen gemeinsam geschützt werden.
Hierzu wird dem Schlüsselwort synchronized ein auswertbarer Ausdruck nachgestellt, der die Resource bezüglich der zu synchronisieren ist bezeichnet.
Im Falle des betrachteten Beispiels ist daher hinsichtlich der gemeinsam beanspruchten (und daher kritischen) Ressource der Zählerstandsdatei zu synchronisieren.

```
(1)import java.io.DataInputStream;
(2)import java.io.DataOutputStream;
(3)import java.io.File;
(4)import java.io.FileInputStream;
(5)import java.io.FileNotFoundException;
(6)import java.io.FileOutputStream;
(7)import java.io.IOException;
(8)
(9)public class Threads21 {
(10)    public static synchronized void main(String[] args) {
(11)            File counterFile = new File ("counter");
(12)
(13)            for (int threadCount=0; threadCount < Integer.parseInt(args[0]); threadCount++)
(14)                    (new IncrementCounter(counterFile)).start();
(15)    } //main()
(16)} //class Threads21
(17)
(18)class IncrementCounter extends Thread {
(19)    File    counterFile;
(20)    public IncrementCounter(File rwFile) {
(21)            counterFile = rwFile;
(22)    } //constructor
(23)
(24)    public void run() {
```

```
(25)            int counterValue;
(26)            DataInputStream dis = null;
(27)            DataOutputStream dos = null;
(28)
(29)            try {
(30)                    for (int i=0; i<10; i++) {
(31)                            synchronized (counterFile) {
(32)                                    dis = new DataInputStream( new FileInputStream
( counterFile ));
(33)                                    counterValue = dis.readInt();
(34)                                    dis.close();
(35)
(36)                                    System.out.println(counterValue+" read by thread "+ (Thread.
currentThread()).getName() );
(37)
(38)                                    dos = new DataOutputStream( new FileOutputStream
( counterFile ));
(39)                                    dos.writeInt( ++counterValue );
(40)                                    dos.close();
(41)                            }//synchronized
(42)                    } //for
(43)            } catch (FileNotFoundException fnfe) {
(44)                    System.out.println("file counter could not be opened!\n"+fnfe.toString()
+"\n"+fnfe.getMessage() );
(45)                    System.exit(1);
(46)            } catch (IOException ioe) {
(47)                    System.out.println("an IOException occurred in thread "+(Thread.
currentThread()).getName()+"!\n"+ioe.toString()+"\n"+ioe.getMessage() );
(48)                    System.exit(1);
(49)            } //catch
(50)    } //run()
(51)} //class IncrementCounter
```

Beispiel 65: Lost-Update-freie Variante des vorhergehenden Beispiels Threads21.java

Auffallendstes Kennzeichen der Synchronisation mittels Schlüsselwort synchronized ist es, daß gemeinsam genutzte Objekte existieren müssen, um die alle Threads konkurrieren.
Auf den Einsatzfall einer eher losen Kopplung der einzelnen Ausführungseinheiten sind die Methoden wait und notify ausgelegt. Der wohl bekannteste Vertreter diese Problemklasse ist das klassische *Erzeuger-Verbracher-Schema*, in dem zwei grundlegend verschiedene Rollen, die des Erzeugers und die des zeitlich nachgelagerten -- und daher zu synchronisierenden -- Verbrauchers unterschieden werden.
Für diese Problemklasse bietet Java das Schlüsselwort wait an. Es erlaubt das Abwarten eines Ereignisses, welches durch notify angezeigt wird.

Das Standardschema zur Benutzung lautet:

```
synchronized doWhenCondition() {
   while (!condition)
      wait();
   ...Bedingung true...
} //synchronized
```

Als Randbedingung gilt die zwingende Einbettung in eine als synchronized deklarierte Methode, um die Modifikation der While-Bediungung durch nebenläufig ausgeführte Threads zu verhindern.
Der wait-Aufruf suspendiert die Ausführug des Prozesses, und gibt gleichzeitig (atomar) seine gesetzten Sperren frei.

Das Benutzungsschema für notify lautet:

```
synchronized changeCondition() {
   ...Änderung von Werten, die in der Bedingung auftreten...
   notify();
} //synchronized
```

Die wait-Methode steht in verschiedenen Ausgestaltungen zur Verfügung:
wait() wartet bis zur Wiedererweckung durch notify; gleiche Wirkung wie wait(0).
wait(long) wartet bis zum Ablauf des durch den Übergabeparameter fixierten Zeitraumes in Millisekunden auf den Aufruf von notify().
wait(long, int) wartet bis zum Ablauf des durch die Übergabeparameter spezifizierten Zeitraumes -- der long-Wert wird als Millisekunden interpretiert, zu dem die als int-Wert gegebenen Nannosekunden addiert werden -- auf den Aufruf von notify().

Zur Wiedererweckung wartender Threads kann die parameterlose Methode notify() benutzt werden; sie erweckt maximal einen wartenden Thread.
Den Wiederanlauf beliebig vieler wartender Threads ermöglicht notifyAll().

Anmerkungen:

- Konstruktoren sind implizit *synchronized*, da ein Objekt nur genau von einer aktiven Einheit gleichzeitig erzeugt werden kann.

- Synchronisierte Klassenmethoden ziehen eine Sperre auf Klassenebene nach sich.
- `synchronized`-Angaben können in erbenden Klassen überschrieben werden. Das Verhalten der Superklasse bleibt hiervon jedoch unberührt.
- Synchronisierte Abschnitte erfordern zusätzlichen Byte-Code zur Zugriffskontrolle und notwendigen Serialisierung, daher sinkt durch Nutzung dieses Mechanismus im Allgemeinen die Ausführungsgeschwindigkeit.

Ermitteln von Threadinformation:
Die Anzahl der laufenden aktiven Threads der aktuellen Gruppe kann durch `activeCount()` ermittelt werden.
Informationen über jeden Thread der virtuellen Maschine liefern die Methoden `getName()` und `setName(String)`. Diese beiden Methoden erlauben die Abfrage, bzw. Definition eines Namens für jeden Thread. Bei der Erzeugung eines Threads wird automatisch ein Name durch die virtuelle Maschine vergeben. (SUNs JVM setzt hier die Zeichenkette `Thread-`*`i`*, wobei *i* die laufende Nummer des Threads ist.)
`enumerate(Thread[])` befüllt den als Parameter übergebenen Array mit Verweisen auf die derzeit aktiven Thread Objekte.
Der aktuelle Thread kann durch die statische Methode `currentThread()` ermittelt werden.

Außer durch Spezialisierung der Klasse `Thread` kann durch Impelementierung des **`Interfaces Runnable`** Nebenläufigkeit erzeugt werden.
Die Schnittstelle definiert ausschließlich die Operation `run`. Auch die Umsetzung der Klasse `Thread` nutzt in den vordefinerten Konstruktoren dieses Interface. Durch den Methodenaufruf `start()` wird im zugehörigen nebenläufigen Objekt per *`0`*`.run()` (*`0`* ist hierbei vom Typ `Runnable`) die Ausführung begonnen.
Hinweis: Dieser Anwendungsfall ist insbesondere für Applets von Bedeutung, da hier keine Spezialisierung von `Thread` erfolgen kann, da zwingend von Klasse `Applet` abgeleitet werden muß.
(Beispiel dazu)

Mehr zu Java Threads erfahren ... (eigene Vorlesung zum Thema)

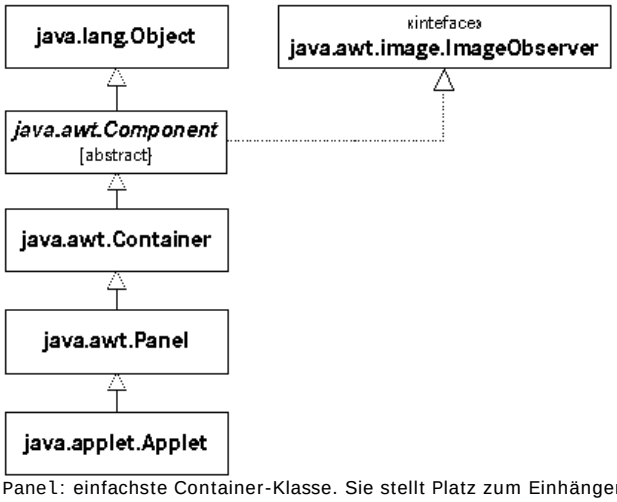
3.2.3 Applets

Applets sind Java-Programme die innerhalb einer Browserumgebung ausgeführt werden. Daher gelten für sie einige besondere Charakteristika, die sie von den bisher betrachteten Java-Applikationen unterscheiden:

- Ein Applet muß innerhalb einer (X)HTML-Seite referenziert werden, es kann nicht eigenständig ausgeführt werden.
- Die Applet-Hauptklasse muß zwingend von `Applet` abgeleitet sein.
- Ein Applet verfügt über keine `main`-Methode.
  Stattdessen erzeugt der Browser ein Objekt der Hauptklasse und führt die Methoden `init` und `start` automatisch aus.
- Durch die Browsereinbettung, konkret: dessen Sicherheitskonzept, ist es einem Applet nicht gestattet auf die lokalen Ressourcen (Dateisystem, etc.) des Rechners zuzugreifen.
  Als Ausnahme hierzu können spezielle „vertrauenswürdige“ Applets erstellt werden, die durch den Programmierer signiert sein müssen.
- Applets bieten keine Textschnittstelle wie (Kommandozeilen-)Applikationen. Daher muß zwingend eine graphische Benutzeroberfläche erstellt werden.

Hinweis: Zu Testzwecken kann ein Applet auch durch einen Applet-Viewer, wie der u.a. in SUNs JDK enthalten ist, ausgeführt werden. Hierfür wird neben dem Applet selbst eine minimale HTML-Seite benötigt.

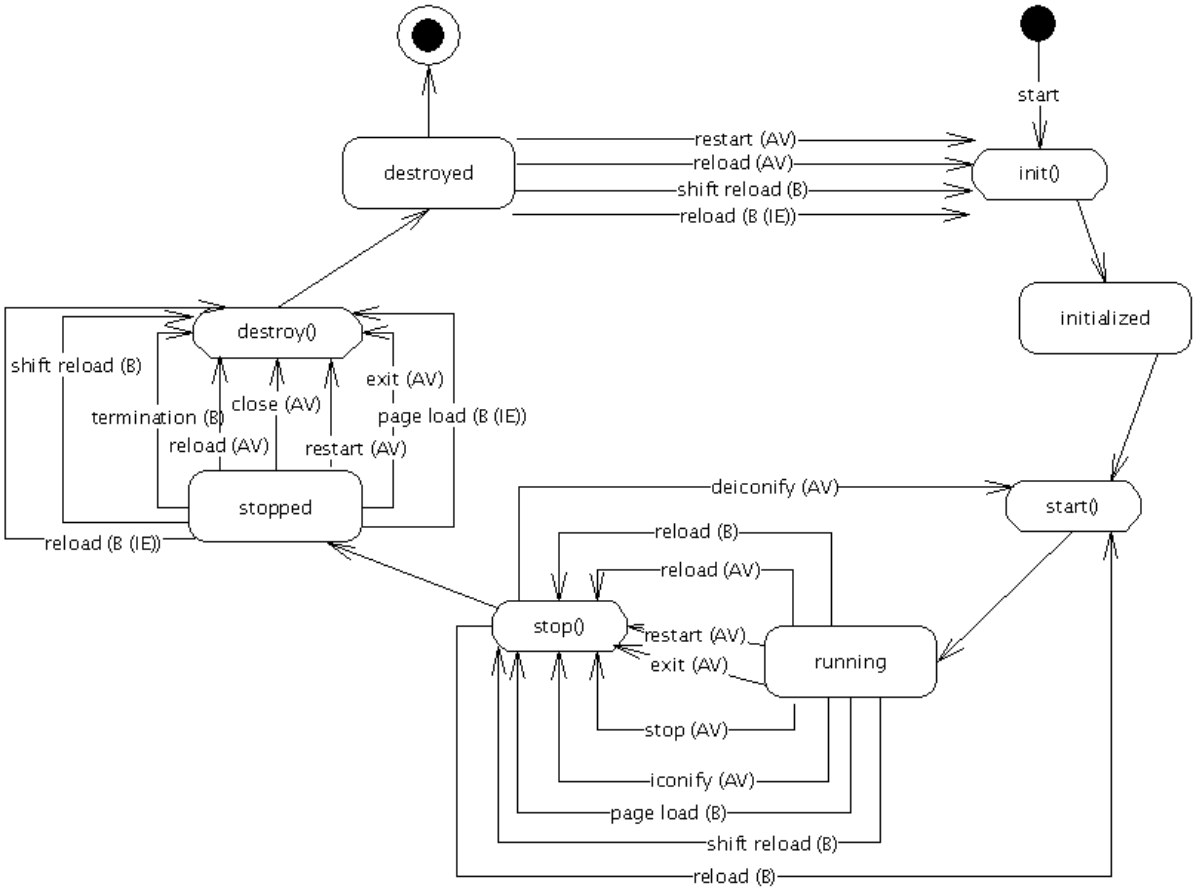
Die Klassenhierarchie von `Applet` und ihre Klassen:



- `Panel`: einfachste Container-Klasse. Sie stellt Platz zum Einhängen beliebiger weiterer Komponenten zur Verfügung.
- `Container`: Generischer Container des *Abstract Window Toolkit* (AWT), der andere AWT-Komponenten enthalten kann.
- `Component`: Objekt das über eine graphische Repräsentation verfügt, die am Bildschirm angezeigt werden kann, und die die Möglichkeit zur Benutzerinteraktion bietet.
  Beispiele: Schaltflächen (Button), Check Boxen und Scrollbars.

Lebenszyklus eines Applets -- die **zentralen Methoden der Klasse `Applet`**:

- `init()`
  Durch den Browser nach dem Laden des Applets, jedoch noch vor Beginn seiner Ausführung, aufgerufen.
  Verwendung: Parameter einlesen, Laden von Bildern und Audiodateien, Objekterzeugung.
- `start()`
  Direkt nach Abschluß der `init()`-Methode durch den Browser aktiviert. Darüberhinaus wird diese Methode immer dann aufgerufen, wenn die beherbergende Web-Seite neu geladen wird.
  Verwendung: Starten von Animationen, Audiowiedergabe.
- `stop()`
  Durch den Browser aufgerufen, wenn die beherbergende Seite nicht mehr angezeigt wird.
  Verwendung: Beenden von Multimediawiedergaben.
- `destroy()`
  Im Anschluß an `stop()` durch den Browser aufgerufen, um die durch das Applet belegten Systemressourcen zurückzufordern.
  Verwendung: Explizite Speicher-Freigabe (z.B. `flush`-Methode für Images).



Das Zustandsübergangsdiagramm zeigt die verschiedenen Status im Leben eines Applets.
Die Übergänge sind mit den auslösenden Ereignissen beschriftet. Ist kein Ereignis angetragen, so erfolgt der Übergang automatisch ohne weitere Bedingungen. In Klammern ist die jeweils ereignisauslösende Umgebung (AV für Appletviewer und B für Web-Browser) angegeben. Im Allgemeinen verhalten sich die beiden großen Browser gleich. Weicht das Verhalten eines Browers ab, so ist dies explizit angegeben (IE kennzeichnet hierbei den MS Internet Explorer).

Weitere wichtige Methoden, die von den Superklassen von `Applet` geerbt werden sind:

- `paint(Graphics)`
  Wird zum vollständigen Neuzeichnen des Applets aufgerufen.
- `update(Graphics)`
  Aktualisiert Fensterinhalt.

Der Aufruf der **Methode `repaint()`** erzeugt einen Aufruf an die `update`-Methode der Komponente.
`repaint()` stellt die asynchrone Variante des direkten Aufrufes von `paint` mit dem aktuellen Graphics-Objekt (`paint( getGraphics() )`) dar. Die Methode garantiert das Neuzeichnen nicht, im Falle hoher Auslastung der virtuellen Maschine kann es auch unterbleiben.

Die **benötigten (X)HTML-Strukturen**:
Leider existieren derzeit verschiedene Varianten ein Java-Applet in eine Hypertextseite einzubinden.
Zunächst die „klassische“ Applet-Referenz: `<applet code="`*`className`*`" width="`*`Breite in Pixel`*`" height="`*`Höhe in Pixel`*`">`
Zusätzlich können (gemäß HTML v4.01 Standard) noch weitere Attribute (Namen-Wert-Paare wie "code" oder "width") definiert werden:
`codebase` -- ermöglicht die relative Interpretation von Pfadnamen ausgehend vom in codebase definierten Katalog
`archive` -- ZIP komprimiertes Archiv das Applet gesucht wird.
`alt` -- zusätzlicher erläuternder Text.

name -- eindeutige Benennung des Applets innerhalb der beherbergenden Seite.
align -- horizontale Ausrichtung auf der Seite.
hspace -- horizontaler Abstand des Applets zur Umgebung.
vspace -- vertikaler Abstand zur Umgebung.

Ferner existieren noch einige proprietäre Attribute, die jedoch nicht durch alle Browser unterstützt werden.

Parameterübergabe an Applets:
erfolgt durch Name-Wert-Paare der Form `<param name="..." value="...">`, die in den Grenzen der Applet-Tags eingeschlossen sind.

```
<applet code="HelloWorldApplet.class" width="100" height="100">
<param name="color" value="green">
</applet>
```

Mit XHTML v1.0, dem Nachfolger des HTML v4.01 Standards, wurde die Applet-Syntax für veraltet (*deprecated*) erklärt, und nunmer ausschließlich die `Object`-Variante zugelassen. Über die Änderung der Tag-Namen hinausgehend dürfte die Plazierung der Referenz auf die Java-Klassendatei, und die explizite Typangabe, als Parameter die augenfälligste Änderung sein.
Diese Syntaxvariante wird jedoch, obgleich Standard, noch nicht von allen derzeit gängigen Browservarianten unterstützt.

```
<object>
<param name="type" value="application/x-java-applet">
<param name="JAVA_CODE" value="HelloWorldApplet.class">
```

Hinweis: In älteren Referenzen findet sich teilweise noch die ursprüngliche HotJava-Syntax, welche ein app-Tag definiert. Ferner enthält dieses die Referenz auf die Klasse als Attribut, jedoch ohne die Dateiextension `.class` explizit anzuführen.

Das **Hello World-Applet**:
Das absolute Minimalapplet, es schreibt lediglich den bekannten Schriftzug, kann folgendermaßen umgesetzt werden:

```
(1)import java.applet.Applet;
(2)import java.awt.Graphics;
(3)
(4)public class HelloWorldApplet extends Applet {
(5)
(6)     public void paint(Graphics g) {
(7)             g.drawString(   getParameter("displayText"),
(8)                                                     Integer.parseInt(getParameter("xPos")),
(9)                                                     Integer.parseInt(getParameter("yPos")) );
(10)    } //paint()
(11)    public String getAppletInfo() {
(12)            return( "written by Mario Jeckle\nversion: 1.1\nCopyright (c) by Mario Jeckle");
(13)    } //getAppletInfo()
(14)    public String[][] getParameterInfo() {
(15)            String appInfo[][] = {
(16)                    {"displayText", "anyString", "string to display"},
(17)                    {"xPos", "numeric value" , "horizontal position"},
(18)                    {"yPos", "numeric value", "vertical position"}
(19)            };
(20)            return appInfo;
(21)    } //getParameterInfo()
(22)} //end class HelloWorldApplet
```

Beispiel 66: Hello World als Java-Applet HelloWorldApplet.java

Applet ausführen

Interaktion mit der Ausführungsumgebung
Durch verschiedene vordefinierte Methoden und Schnittstellen kann jedes Applet mit seiner Ausführungsumgebung, dem Browser oder Applet Viewer, in Interaktion treten. Hierdurch können von Seiten des Applets auch Dienste wie Caching, Anzeigen einer Nachricht, Abspielen von Audiodateien der Ausführungsumgebung benutzt werden.
Einfachster Fall ist die Methode `getAppletInfo`. Sie liefert eine Zeichenkette zurück, welche üblicherweise Informationen über den Autor, die Version und die Rechte am Applet beinhaltet.
Eine ähnliche Funktion erfüllt die Methode `getParameterInfo()`. Sie liefert einen Array von dreielementigen String-Arrays zurück, welche die erlaubten Parameter näher beschreiben.
Mittels `getParameter(String)` können die Werte jedes einzelnen Parameters ausgelesen werden.

```
(1)import java.applet.Applet;
(2)import java.awt.Graphics;
(3)
(4)public class HelloWorldApplet extends Applet {
(5)
(6)     public void paint(Graphics g) {
(7)             g.drawString(   getParameter("displayText"),
(8)                                                     Integer.parseInt(getParameter("xPos")),
(9)                                                     Integer.parseInt(getParameter("yPos")) );
(10)    } //paint()
(11)    public String getAppletInfo() {
(12)            return( "written by Mario Jeckle\nversion: 1.1\nCopyright (c) by Mario Jeckle");
(13)    } //getAppletInfo()
(14)    public String[][] getParameterInfo() {
(15)            String appInfo[][] = {
(16)                    {"displayText", "anyString", "string to display"},
(17)                    {"xPos", "numeric value" , "horizontal position"},
(18)                    {"yPos", "numeric value", "vertical position"}
(19)            };
(20)            return appInfo;
(21)    } //getParameterInfo()
(22)} //end class HelloWorldApplet
```

Beispiel 67: Parametrisierung des bekannten HelloWorld-Applets HelloWorldApplet.java

Applet ausführen

Die Erweiterung des bekannten Beispielapplets zeigt den Einsatz der `getAppletInfo`-Methode zur Rückgabe Applet-spezifischer Inhalte.
Zusätzlich wird der anzuzeigende Text und seine Position (=Parameter der Methode `drawString`) über Parameter gesteuert, die durch die Methode `getParameterInfo` in ihrer Funktion näher beschrieben sind.
Die Parameter werden aus der (X)HTML-Quelle durch die Methode `getParameter` ausgelesen.

Die Schnittstelle `AppletContext` erlaubt die aktive Beeinflussung der Ausführungsumgebung des Applets. Die Methode `getAppletContext` liefert die aktuelle Context-Ausprägung zurück.
Jedes `AppletContext`-Objekt stellt folgende Methoden zur Verfügung:

- `getApplet(String)`
  liefert ein `Applet`-Objekt zurück, das entsprechend der übergebenen Zeichenkette benannt ist.
- `getApplets()`
  Liefert alle Applets im aktuellen Kontext (d.h. dem aktuell angezeigten (X)HTML-Dokument) zurück.
- `getAudioClip(URL)`
  Liefert ein durch URL identifiziertes `AudioClip`-Objekt zurück.
- `getImage(URL)`
  Liefert eine über URL identifzierte Graphik als `Image`-Objekt zurück.
- `showDocument(URL)` und `showDocument(URL, String)`
  Laden das durch die URL bezeichnete Dokument, bzw. den bezeichneten Frame, in die aktuelle Ausführungsumgebung, sofern es sich dabei um einen Browser handelt. Appletviewer ignorieren diesen Methodenaufruf.
- `showStatus(String)`
  Zeigt die übergebene Zeichenkette im Statusfenster des Browsers oder Appletviewers an.

```
(1)import java.applet.Applet;
(2)import java.awt.Graphics;
(3)import java.net.URL;
(4)import java.net.MalformedURLException;
(5)
(6)public class HWAURLForward extends Applet {
(7)     public void paint(Graphics g) {
(8)
(9)             g.drawString(   getParameter("displayText"),
(10)                                                    Integer.parseInt(getParameter("xPos")),
(11)                                                    Integer.parseInt(getParameter("yPos")) );
(12)            try {
(13)                    Thread.sleep(5000); //five seconds delay
(14)            } catch (InterruptedException iee) {
(15)                    System.out.println("an InterrupedException occurred");
(16)            } //catch
(17)
(18)            try {
(19)                    (this.getAppletContext()).showDocument( new URL(getParameter("URL")) );
(20)            } catch (MalformedURLException murle) {
(21)                    System.out.println("illegal URL");
(22)            } //catch
(23)    } //paint()
(24)    public String getAppletInfo() {
(25)            return( "written by Mario Jeckle\nversion: 1.1\nCopyright (c) by Mario Jeckle");
(26)    } //getAppletInfo()
(27)    public String[][] getParameterInfo() {
(28)            String appInfo[][] = {
```

```
(29)                    {"displayText", "anyString", "string to display"},
(30)                    {"xPos", "numeric value" , "horizontal position"},
(31)                    {"yPos", "numeric value", "vertical position"},
(32)                    {"URL", "any valid URL according to IETF's RFC 1738", "URL to jump to"}
(33)            };
(34)
(35)            return appInfo;
(36)    } //getParameterInfo()
(37)} //class HWAURLForward
```

Beispiel 68: Hello World Applet mit URL-Forward nach fünf Sekunden HWAURLForward.java

Das Beispiel erweitert das parametrisierte HelloWorld-Applet um eine URL-Weiterleitung nach fünf Sekunden. Die notwendige Zieladresse wird aus dem Parameter `URL` geladen.
zugehörige HTML-Seite

Sicherheitskonzept von Applets:
Nachfolgende Tabelle faßt die Rechte und Beschränkungen von Applikationen und Applets, sowohl für die Ausführungsumgebung Web-Browser als auch im Appletviewer, zusammen.

Aktion	Browser	Applet Viewer	Java Applikation, ohne weitere Einschränkungen
Lesen lokaler Dateien	⊘	✓	✓
Schreiben lokaler Dateien	⊘	✓	✓
Zugriff auf Dateinformation	⊘	✓	✓
Löschen von Dateien	⊘	⊘	✓
Ausführen anderer Programme	⊘	✓	✓
Auslesen der Property `user.name`	⊘	✓	✓
Zugriff auf Netzwerkport des Herkunfts-Servers des Applets	✓	✓	✓
Zugriff auf Netzwerkport eines beliebigen Servers	⊘	✓	✓
Dynamisches Laden von Java-Bibliotheken	⊘	✓	✓
Beenden der virtuellen Maschine mit `exit`	⊘	✓	✓
Erzeugen von Fenstern	✓ mit Warnhinweis	✓	✓
Ausführen nativen-Codes	⊘	⊘	✓

Das Applet `openProperties` zeigt die Inhalte der abfragbaren Systemeigenschaften an.

CannotDisplayApplet

Beim Versuch innerhalb eines Applets eine aus Sicherheitsgründen nicht zugleassene Funktionalität auszuführen wird ein Ausnahmeereignis-Objekt der Klasse `AccessControlException` erzeugt.

Hinweis: Die dargestellten Einschränkungen stellen für manche Anwendungsfälle zu deutliche Restriktionen dar. Daher existiert mit den sog. *signed Applets* eine Möglichkeit einzelne Einschränkungen für bestimmte Applets gezielt zu öffnen.

Drei Komponenten wirken bei der Realisierung des Java-Sicherheitsmodells zusammen:

1. Der binäre Bytecode wird nicht direkt durch die physische Hardware ausgeführt, sondern durch die virtuelle Maschine interpretiert.
2. Der Security Manager überwacht alle sicherheitsrelevanten Operationen.
3. Applets mit weitergehenden Berechtigungen müssen sich explizit identifzieren.

Abspielen von Audiodateien:
War diese Funktionalität bis zum Erscheinen der Java 2 Plattform ausschließlich Applets vorbehalten steht sie nun

auch in Java-Applikationen zur Verfügung.
Generelles Vorgehen:
Laden einer Audiodatei mittels der Methodenfamilie `getAudioClip` über eine gültige URL.
Im Anschluß kann das rückgegebene Objekt, welches die `AudioClip`-Schnittstelle implementiert abgespielt werden.
Java ab Version 2 unterstützt MIDI-Dateien sowie Audiodateien der Formate `RMF`, `WAVE`, `AIFF` und `AU`.
Zur Soundausgabe in Applikationen wurde die Klassenmethode `newAudioClip(URL)` eingeführt, die eine Audiowiedergabe auch ohne Existenz eines Applet-Objekts ermöglicht.
Üblicherweise werden im Zusammenhang mit der Audiowiedergabe `Threads` verwendet um die Sounds mit mit anderen Ereignissen zu synchronisieren.

Anzeigen von Graphiken:

Bilder in den Formaten `GIF`, `PNG` oder `JPEG` können durch die Methoden `getImage` geladen werden.
Die Anzeige erfolgt durch `drawImage(...)`.
Das GIF98a-Animationsformat wird ebenfalls unterstützt.

Anmerkung: Alternativ kann auch signaturgleiche Methode `getImage` der Klasse `java.awt.Toolkit` benutzt werden.

```
(1)import java.applet.Applet;
(2)import java.awt.Graphics;
(3)import java.awt.GridBagLayout;
(4)import java.awt.Button;
(5)import java.awt.TextField;
(6)import java.awt.Image;
(7)import java.awt.image.ImageObserver;
(8)import java.awt.event.ActionEvent;
(9)import java.awt.event.ActionListener;
(10)
(11)public class GfxViewer extends Applet implements ActionListener {
(12)    protected TextField graphicsName;
(13)    Image imgToDisplay = null;
(14)
(15)    public void init() {
(16)            graphicsName = new TextField ("enter URL of image to view", 50);
(17)            Button loadBtn = new Button ("view");
(18)            GridBagLayout gbl = new GridBagLayout();
(19)            setLayout(gbl);
(20)
(21)            loadBtn.addActionListener(this);
(22)            add(loadBtn);
(23)            add(graphicsName);
(24)    } //init
(25)
(26)    public void actionPerformed(ActionEvent event) {
(27)            imgToDisplay = getImage( getDocumentBase(), graphicsName.getText() );
(28)            repaint();
(29)    } //actionPerformed()
(30)
(31)    public boolean imageUpdate(Image img, int infoflags, int x, int y, int width, int height) {
(32)            repaint();
(33)            if ((infoflags &  ImageObserver.SOMEBITS) != 0) {
(34)            System.out.println("part just loaded:"+x+" "+y+" "+ width+" "+height);
(35)                    return true;
(36)            } //if
(37)            if ((infoflags &  ImageObserver.ALLBITS) != 0) {
(38)            System.out.println("image loaded completely");
(39)                    return false;
(40)            } //if
(41)            //gets here when height and width of image is available
(42)            return true;
(43)   } //imageUpdate()
(44)
(45)    public void paint(Graphics g) {
(46)            if (imgToDisplay != null) {
(47)                    g.drawImage(imgToDisplay, 1, 1, this);
(48)            } //if
(49)    } //paint
(50)} //class GfxViewer
```

Beispiel 69: Ein einfacher Graphik-Viewer GfxViewer.java

Applet ausführen

Das Beispiel implementiert einen einfachen Graphik-Viewer für die beiden standardmäßig unterstützten Formate.
Es verwendet bereits interaktive Komponenten aus dem *Abstract Windows Toolkit* (AWT) (siehe `init`- und `actionPerformed`Methode. Die rudimentäre Funktionalität erschöpft sich darin, eine Graphikdatei aus dem aktuellen Verzeichnis (ermittelt per `getDocumentBase()` zu laden, und im aktuellen Fenster anzuzeigen.
Zunächst wird das Bild per `getImage`-Methode referenziert (`graphicsName.getText()` gibt den im Textfeld

eingegebenen Namen als String zurück).
Der Aufruf von repaint() wirkt auf die gesamte Komponente. Erfolgt das Neuzeichnen, so wird die paint-Methode des Applets aufgerufen. Der dort plazierte Methodenaufruf drawImage auf dem Graphics-Objekt zeichnet das Bild.

Durch Überschreiben der Methode imageUpdate (ererbt von Component, welches die Schnittstelle ImageObserver implementiert) kann der Ladevorgang des Bildes verfolgt werden. Diese Methode wird wärend des Ladevorganges immer wieder automatisch aufgerufen.

Anmerkungen:

- Die getImage-Methoden sind asynchron ausgelegt, und kehren daher sofort nach ihrem Aufruf zurück, unabhängig davon ob das referenzierte Bild existiert.
  Die erforderlichen Graphikdaten werden erst beim Zeichenversuch in den Speicher geladen.
- Die drawImage-Methoden sind (ebenfalls) asynchron ausgelegt, und kehren sofort nach ihrem Aufruf zurück, das Laden des Bildes erfolgt ggf. in einem eigenen Thread.
  Hinweis: Die Klasse MediaTracker stellt eine Implementierung zur Verfügung, welche es ermöglicht das Laden eines Bildes oder einer Audiodatei zu verfolgen.

Animationen:
...sind prinzipiell nichts anderes als die schnelle Wiedergabe verschiedener Einzelbilder. Jedoch gilt es auf einige Randbedingungen Rücksicht zu nehmen:

- Die Anzeigefrequenz der Einzelbilder sollte nicht zu niedrig wählt sein, sonst erscheint die Animation ruckelig, allerdings auch nicht zu hoch, sonst gehen dem Auge des Betrachters Einzelbilder verloren.
- Beim Neuzeichnen der visuellen Komponente durch die repaint()-Methode kann es zu Flimmer- oder Flackereffekten kommen.
- Durch den Darstellungsaufwand sollte der übrige interaktive Programmablauf nicht gestört werden.

Das Beispiel zeigt die Realisierung einer simplen Uhr (Java-Date-Objekt) als animiertes Applet.

```
(1)import java.applet.Applet;
(2)import java.awt.Graphics;
(3)import java.util.Date;
(4)
(5)public class Clock extends Applet implements Runnable {
(6)     protected int counter;
(7)
(8)     public void start() {
(9)             Thread myClock = new Thread(this);
(10)            myClock.setPriority(Thread.MIN_PRIORITY);
(11)
(12)            myClock.start();
(13)    } //start()
(14)
(15)    public void run() {
(16)            while (true) {
(17)                    repaint();
(18)                    try {
(19)                            Thread.sleep(1000);
(20)                    } catch (InterruptedException ie) {
(21)                            //ignore it
(22)                    } //catch
(23)            } //while
(24)    } //run()
(25)
(26)    public void paint(Graphics g) {
(27)            g.drawString( new Date().toString(),50,50);
(28)    } //paint()
(29)} //class Clock
```

Beispiel 70: Eine einfache Uhr Clock.java

Applet ausführen

Der notwendige Bildschirmneuaufbau wird in einen separaten niedrig priorisierten Thread verlagert.
Hinweis: Der neue Thread wird nicht, wie bisher per Ableitung von Thread definiert und als spezialisiertes Objekt instanziiert. Das Applet implementiert die Runnable-Schnittstelle; daher kann ein Objekt vom Typ des Applets einer Variable des Typs Thread zugewiesen werden. Erst hierdurch werden die Thread-spezifischen Methoden, wie Prioritätssänderungen, verfügbar.

Die häufigst anzutreffende Realisierung graphischer Animationen ist die Einblendung sich leicht unterscheidender Bilder, wodurch ab einer gewissen Bildfrequenz der Eindruck kontinuierlicher Bewegung entsteht (Trickfilmeffekt). Diese Animationsvariante kann leicht mit den bisher vorgestellten Möglichkeiten realisiert werden. Hierzu werden zunächst die benötigten Bilder vollständig geladen (z.B. getImage(String) oder getImage(URL) aus der Toolkit-Klasse) und anschließend in schneller Folge am Bildschirm angezeigt (z.B. mit Methoden der drawImage(...)-Familie).
Im Kern ist das Verfahren identisch zur Anzeige statischer Graphiken.

```
(1)import java.awt.Graphics;
(2)import java.awt.Image;
(3)import java.applet.Applet;
(4)
(5)public class AnalogousClock extends Applet implements Runnable {
(6)     static Image images[];
(7)     static int counter=0;
(8)   Thread animator;
(9)
(10)   public void init() {
(11)            images = new Image[12];
(12)      for (int i=0;i<images.length;i++)
(13)            images[i] = getImage( getDocumentBase(), "clock-"+i+".gif");
(14)    } //init()
(15)
(16)   public void start() {
(17)    while ( images[0].getWidth(this) == -1 || images[0].getHeight(this) == -1)
(18)            ;
(19)    this.resize( images[0].getWidth(this) , images[0].getHeight(this) );
(20)    animator = new Thread (this);
(21)            animator.setPriority(Thread.MIN_PRIORITY);
(22)      animator.start ();
(23)   } //start()
(24)
(25)    public void stop() {
(26)            animator = null;
(27)    } //stop()
(28)
(29)   public void run () {
(30)            while (true) {
(31)                    try {
(32)            Thread.sleep(200);
(33)          } //try
(34)          catch (InterruptedException ie) {
(35)                          //ignore it
(36)          } //catch
(37)
(38)                    repaint();
(39)          if (++counter==images.length)
(40)             counter=0;
(41)            } //while
(42)   } //run()
(43)
(44)   public void paint (Graphics g) {
(45)    System.out.println("drawing image: clock-"+counter+".gif");
(46)    g.drawImage (images[counter], 0, 0, this);
(47)   } //paint()
(48)} //class AnalogousClock
```

Beispiel 71: Analoge Uhr als Bildfolge AnalogousClock.java

Applet ausführen

Das Beispielprogramm zeichnet in einem eigenen Thread alle 200 Millisekunden den Bildschirm mit einem neuen Bild neu.
Zunächst werden durch die `init`-Methode die benötigten zwölf Graphiken in einen `Image`-Array geladen. Zum Ausführungsbeginn (`start`-Methode) wird zunächst die Fenstergröße des Applets (bei Ausführung im Applet-Viewer) auf die Dimensionen der ersten geladenen Graphik -- im Beispiel besitzen alle Graphiken dieselben Ausmaße, daher ist die Auswahl der Referenzgraphik unbedeutend -- gesetzt. Aufgrund der asynchronen Realisierung von `getImage()` muß auf den Abschluß des Ladevorganges gewartet werden; andenfalls liefern die Methoden `getHeight` und `getWidth` mit `-1` einen ungültigen Wert zurück.
Der niederpriore Thread `animator` sorgt alle 200 Millisekunden für den notwendigen Bildschirmneuaufbau -- durch Aufruf der `repaint`-Methode --, und schaltet die Graphiken fort, wodurch der Animationseffekt entsteht.

Bei dieser Anwendung fällt ein starker **Flimmereffekt** ins Auge. Dieses (unschöne) Verhalten ist oftmals bei graphischen Java-Applets und -Applikationen anzutreffen. Zur Behebung existieren einige Standardlösungsvarianten:

Die einfachste Lösung liegt im Überschreiben der von `Component` ererbten Methode `update()`.
Im Standardfalle löscht `update()` den gesamten Bildschirmbereich und ruft im Anschluß `paint()` zum vollständigen Neuzeichnen auf.

Im vorliegenden Beispiel ist jedoch das dem Neuzeichnen vorausgehende Löschen nicht notwendig, da die nachfolgende Graphik ohnehin die vorhergende vollständig überdeckt.
Daher kann die `update`-Methode auf den reinen Aufruf von `paint` beschränkt werden.
Das zusätzliche Codefragment ergibt sich als:

```
public void update(Graphics g) {
   paint(g);
} //update()
```

ergänzter Quellcode
ergänztes Applet ausführen

Diese einfache Maßnahme hilft oftmals bei auftretenden Problemen, jedoch ist sie nur für den eng umrissenen Problemkreis anwendbar, in dem neue graphische Elemente die vorhergehenden vollständig überdecken. Andernfalls kann es zu Darstellungsfehlern kommen.
Daher kommt bei realen Problemstellungen zumeist **double Buffering** zum Einsatz.

Der Mechanismus des double bufferings macht es sich zu Nutze, daß auch im nicht sichtbaren Bereich des Bildschirms graphische Operationen vorgenommen werden können. Die Anwendung verläuft üblicherweise in zwei Schritten: zunächst wird der darzustellende Bildschirminhalt verdeckt vorbereitet, und im zweiten Schritt in einer einzigen -- im Allgemeinen sehr performant ausgeführten -- Kopieroperation in den sichtbaren Bereich übertragen.

Ablaufprinzip:

1. Erzeugung eines `Image`-Objekts beliebiger Größe (gesteuert durch die beiden Übergabeparameter) mittels `createImage(int, int)`.
   `createImage(int, int)` ist in der Klasse `Component` definiert, und wird an `Applet` vererbt.
2. Erzeugung des Zeichenbereichs (*graphics context*) durch die Methode `getGraphics()` auf dem `Image`-Objekt.
3. Zeichenoperationen auf dem Graphikkontext.
4. Anzeigen des Graphikkontextes inerhalb der `paint`-Methode mit einer Methode aus der `drawImage(...)`-Familie.

Codeschablone:

```
Image im = createImage(100,100);
Graphics graph = im.getGraphics();
//Operationen auf graph

public void paint(Graphics g) {
   g.drawImage(im, ...);
} //paint()
```

In `paint()` werden die vorgezeichneten `Image`-Objekte ohne weitere Veränderungen angezeigt.
Je nach Komplexität der Anwendung können so viele `Image`-Objekte wie benötigt parallel gehalten und bei Bedarf angezeigt werden.

3.2.4 Collection API

Die Collection API liefert als Bestandteil der Java Standard API Hilfsklassen zum Umgang mit generischen Objektsammlungen.
Der Begriff *Collection* bezeichnet Datenstrukturen wie Listen, Stacks, Schlangen, Bäume und Mengen im mathematischen Sinne.

Das **Collection Framework** der Java-API wird aus den vorgegebenen Schnittstellen, den Operationen und ihrer definierten Semantik, den konkreten Implementierunge der wiederverwendbaren Datenstrukturen und der zugrundeliegenden Algorithmen gebildet.
Das wohl bekannteste Beispiel eines solchen Rahmens dürfte die *C++ Standard Template Libraray* (Abk. STL) und die Collection-Klassen der Programmiersprache SmallTalk sein.
Neben dem Collection Framework der Java API existiert mit der *Generic Collection Library for Java* (Abk. JGL) auch noch eine frei verfügbare Adaption der STL an Java.

Da Java bis zur Version 1.5 keine parametrische Polymorphie unterstützte sind diese Klassen auch (noch) in ihrer generischen Fassung umgesetzt. Ihr Einsatz unter Nutzung der *Java Generics* wird Gegenstand des nächsten Kapitels sein.
Um ihre Einsetzbarbeit möglichst universell zu gestalten erlauben alle Collection-Klassen die Verwaltung von Objekten des Typs `Object`, dem gemeinsamen Basistyp aller Java-Objekte.

Durch diese Mimik geht die **Typsicherheit verloren**!
Das bedeutet, daß in jeder Collection ausschließlich Objekte des Typs `Object` verwaltet werden können, die vor dem Einstellen in die Sammlung (implizit und automatisch) in diesen Typ konvertiert werden. Diese Typumwandlung ist als *up-cast* typsicher möglich, da alle Java-Klassen immer direkte oder transitive Subklasse von `Object` sind. Bei der Entnahme von Collection-Elementen ist jedoch die Rekonstruktion des Ursprungstypen notwendig. Diese Typumwandlung ist als *down-cast* explizit anzugeben. Beim Konvertierungsversuch in den falschen Typ kann es hier zu einer `ClassCastException` kommen.
Unter praktischen Gesichtspunkten bietet sich die Vorschaltung einer Typprüfung durch `instanceof` an.

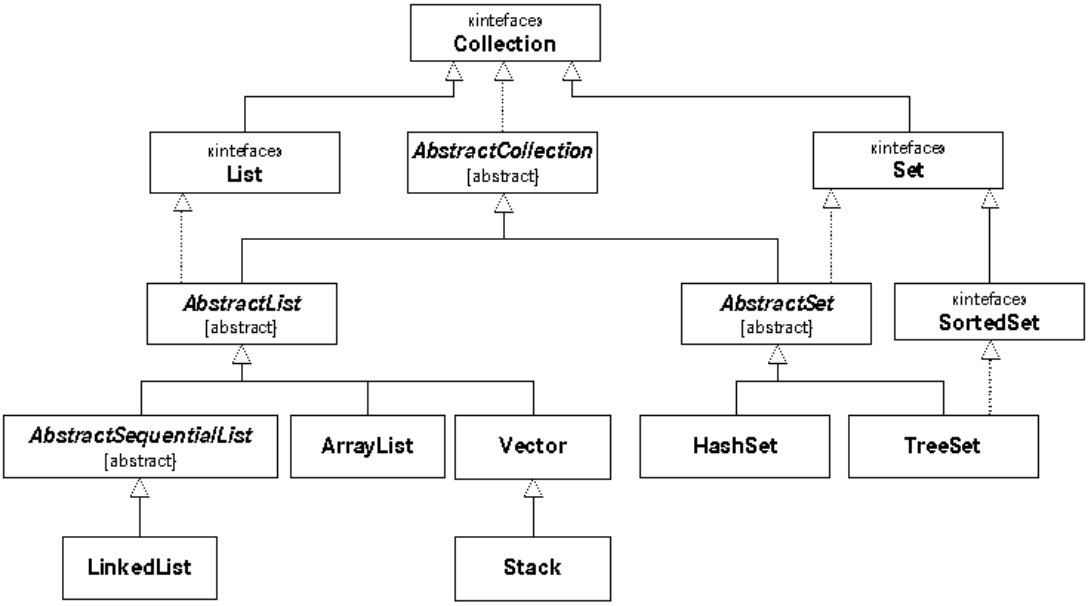
Andererseits schafft die Abkunft aller Java-Objekte von `Object` auch erst die Voraussetzungen für ein generisches Collection Framework zur Verwaltung beliebiger Java-Objekte.
Hiermit sind im Speziellen drei Methoden der Klasse `Object` gemeint:

- `boolean equals(Object)`
  Erlaubt den Vergleich beliebiger Java-Objekte auf Identität. Daher liefert sie auch bei der Gleichheit aller Attributwerte `false` zurück, da sich die Objekte in ihrer (für den Benutzer nicht sicht- und zugreifbare) Objektidentität unterscheiden.

`equals()` bildet eine Äquivalenzrelation auf `Object` und ist daher reflexiv, symmetrisch und transitiv.
Insondere duplikatfreie Sammlungen greifen auf diese Methode zurück, um zu ermitteln, ob sich ein aufzunehmendes Objekt bereits in der Sammlung befindet.

- `hashCode()`
  Die native Methode liefert für einen über die gesamte Laufzeit des Java-Programms unveränderten `int`-Wert als eineindeutige Kennung jedes Objekts. Identische Objekte (d.h. Objekte für die `equals` den Wert `true` liefert) haben identische Kennungen.
  Die über Hashfunktion jedem Objekt eineindeutig zuordenbare `int`-Repräsentation kann als Hash-Code verwendet werden.

Bereits seit Vesion 1.0 des JDKs stehen die Klassen `Bitset`, `Dictionary`, `Hashtable`, `Stack` und `Vector` zur Verfügung.
Diese wurde mit dem JDK v1.2 um einige weitere Klassen und Schnittstellen ergänzt.



Mit Ausnahme von `Vector` und `Stack` wurden alle dargestellten Klassen und Schnittstellen mit der Java-2-Plattform eingeführt.

Grundlegende Konzepte:
Gemeinsam sind allen Collection-Varianten die basalen Operationen

- Erzeugen der Sammlung
- Einstellen von Elementen
- Entnehmen von Elementen
- Zugriff auf einzelne Elemente
- Löschen der Sammlung

Darüberhinaus bieten die verschiedenen Collection-Klassen auch Möglichkeiten zur Suche, Sortierung usw. an.

Interessante (Realisierungs-)Details einer jeden Collection-Klasse sind weiter:

- interne Umsetzung
  Die tatsächliche technische Realisierung einer wie auch immer präsentierten abstrakten Menge spielt eine entscheidende Rolle hinsichtlich Skalierbarkeit und damit verbundenem Zugriffsverhalten.
- unterstützte Zugriffsmodi
  Sie bestimmen wesentlich die praktische Verwendbarkeit einer Sammlung in exisitierenden Algorithmen. Neben den gebotenen Zugriffsmodi spielt auch ihre Realisierung -- konkret die Anzahl der notwendigen Zugriffe bis das gesuchte Datum gefunden ist -- eine entscheidende Rolle.
- Duplikatfreiheit und Anordnungseigenschaft
  Der Sammlungsgedanke an sich unterstellt zunächst weder eine (ungeordnete duplikatfreie) Menge im mathematischen Sinne, noch eine (nach welchen Kriterien auch immer) angeordnete Aufzählung der Elemente.

Das Standard Collection Framework umfaßt folgende ausprogrammierte Klassen:

Klasse	Charakteristika / Einsatzgebiete
`ArrayList`	Dynamischer Array, der intern durch statischen (Java nativen) Array realisiert ist. Lesender und schreibender indizierter Zugriff auf beliebige Elementposition. Ähnelt `Vector`, ist jedoch nicht threadsicher.
`Vector`	Bereits im JDK v1.0 enthalten, mittlerweile um Schnittstelle `List` angereichert. Threadsichere Variante der `ArrayList`.
`LinkedList`	Implementierung einer doppelt verketteten Liste. Dadurch konstanter Aufwand beim Einfügen und Löschen an jeder beliebigen Listenposition, jedoch höherer Aufwand beim Anfügen und Entnehmen von letzter Listenposition als bei `ArrayList`.
`Stack`	Stack nach dem *last-in-first-out*-Prinzip. Wie `Vector` bereits seit dem JDK v1.0 angeboten, und daher nicht originärer Bestandteil des Collection Frameworks. Ebenfalls wie `Vector` threadsicher realisiert.

Hashtable	Hash-Tabelle die Einstellung von Werten über einen eineindeutigen Schlüssel. Bereits mit JDK v1.0 eingeführt; nicht Bestandteil des Collection Frameworks.
HashSet	Realisierung einer Menge im mathematischen Sinne, d.h. anordnungslose (assoziative) Speicherung paarweise verschiedener (Duplikatfreiheit!) Elemente.
TreeSet	Gleiches Prinzip wie HashSet, jedoch interne Speicherorganisation als Baum. Die Elemente werden, sofern nicht anders angegeben, in ihrer *natürlichen* Ordnung abgelegt. Hierzu muß durch die Klasse jedes abzulegenden Objektes die Comparable-Schnittstelle umgesetzt sein, d.h. die compareTo-Methode implementiert werden.
HashMap	Erweiterung des HashSets um Schlüssel. Ablage von beliebigen Werten (auch Duplikaten) gemeinsam mit einem eineindeutigen Schlüssel. (Dieser Strukturtyp wird auch als *Bag* bezeichnet.) Die Einträge werden intern durch eine Hash-Tabelle verwaltet. Äquivallent zur Hashtable, abgesehen von der mangelnden Threadsicherheit und den zugelassenen Null-Elementen.
TreeMap	Erweiterung der HashMap auf eine Baum-basierte (rot-schwarz-Baum) Ablage eines Wertes (Object) mit einem eineindeutigen Schlüssel.
WeakHashMap	Prinzipiell identisch zur HashMap, jedoch werden Schlüssel-Wert-Paare entfernt, sobald die letzte externe Referenz auf den Schlüssel ungültig wird.

Ein **einführendes Beispiel**:
Die Klasse Vector ist dem „klassischen“, d.h. bereits mit JDK v1.0 eingeführten, Anteil der Collection-API zuzurechnen. Sie stellt im Grunde eine dynamische Arrayimplementierung zur Verfügung, die intern als statischer Array realisiert ist. Wie bei Arrays üblich erfolgt der Element-Zugriff durch einen ganzzahligen int-Index.
Durch den Ausbau der Collection-API in der Java-2-Plattform wurde die Vector-Implementierung nochmals überarbeitet und an die heute Struktur (namentlich die neu eingeführte Schnittstelle List) adaptiert.

```
(1)import java.io.FileDescriptor;
(2)import java.io.FileNotFoundException;
(3)import java.io.FileReader;
(4)import java.io.IOException;
(5)import java.io.LineNumberReader;
(6)import java.util.Vector;
(7)
(8)public class VectorEx1 {
(9)     public static void main(String[] args) {
(10)            LineNumberReader lnr;
(11)            String line;
(12)            Vector vec = new Vector();
(13)
(14)            try {
(15)                    lnr = new LineNumberReader(new FileReader( FileDescriptor.in ) );
(16)
(17)                    while ( (line = lnr.readLine()).compareTo("exit")!=0 ) {
(18)                            vec.add(line);
(19)                    } //while
(20)                    lnr.close();
(21)
(22)                    for (int i=0; i<vec.size(); i++)
(23)                            System.out.println("element no. "+i+": "+vec.get(i) );
(24)            } catch (FileNotFoundException fnfe) {
(25)                    System.out.println("cannot find stdin!");
(26)            } catch (IOException ioe) {
(27)                    System.out.println("an IOException occurred\n+"+ioe.toString()+"\n"+ioe.
getMessage() );
(28)            } //catch
(29)    } //main()
(30)} //class VectorEx1
```

Beispiel 72: Vektor-Operationen VectorEx1.java

Das Programm liest solange von der Standardeingabe zeilenweise Zeichenketten ein, bis der String *exit* eingegeben wird. Jede eingelesene Zeichenkette wird -- mit der Methode add() -- als Vector-Element angelegt.
Nach Beendigung des Einlesevorganges werden die gespeicherten Vector-Elemente in der Reihenfolge der Anlage ausgegeben. Dies geschieht per indiziertem Zugriff auf jedes Element mit der Methode get(int). Der Zugriff erfolgt nicht mehr, wie bei Array üblich per direkter Indizierung in eckigen Klammern, sondern jetzt über eine eigene Methode, die jedoch den gewohnten Sicherheitsmechanismus der ArrayOutOfBoundException beim Zugriffsversuch außerhalb der Vector-Grenzen beibehält.

Beispielablauf:

```
$java vectorEx1
this
is
a
simple
test
containing an
```

```
empty
line
exit
element no. 0: this
element no. 1: is
element no. 2: a
element no. 3: simple
element no. 4: test
element no. 5: containing an
element no. 6:
element no. 7: empty
element no. 8: line
```

Die **Skalierbarkeit eines Vektor-Objekts** kann durch Konstruktor beeinflußt werden. Der erlaubt die Angabe zweier Freiheitsgrade, der initialen Kapazität und eines Wachstumsfaktors. Werden, wie im Beispiel, keine Parameter übergeben, so wird gegenwärtig (d.h. im JDK v1.3) ein Vector der Größe zehn mit einem Wachstumsfaktor von Null erzeugt.
Wird durch die Einfügemethode add die Aufnahmekapazität des Vectors überschritten, so wächst er um den als *capacityIncrement* angegebenen Faktor, oder verdoppelt seine gegenwärtige Größe falls kein Wachstumsfaktor angegeben wurde. In beiden Fällen sind jedoch alle Vector-Elemente in einen neu zu erzeugenden Array zu transferieren, wodurch es bei umfangreichen Vectoren zu Zeitverlusten kommen kann. Die in der gezeigten Implementierung benutzte Umsetzung durch System.arraycopy läuft als native Methode jedoch sehr performant ab.
Vor grösseren Einfügevorgängen bekannten Umfanges bietet sich jedoch die explizite Allokation des notwendigen Speicherplatzes durch Aufruf der Methode ensureCapacity(int) an, die sicherstellt, daß mindestens die im Parameter übergebene Anzahl von Objekten ohne zusätzliche Allokations- und Kopiervorgänge in den Vector eingestellt werden kann.
Der nachfolgende Codeausschnitt aus den Java-Quellen läßt das interne Verhalten von Vector anschaulich werden.

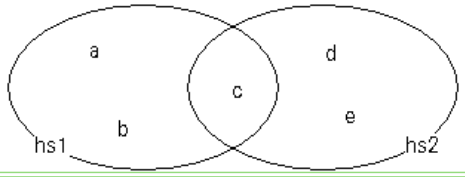
```
package java.util;

public class Vector extends AbstractList implements List, Cloneable, java.io.Serializable {
   protected Object elementData[];
   protected int capacityIncrement;
   protected int elementCount;

   public Vector() {
      this(10);
   } //constructor

   public Vector(int initialCapacity) {
      this(initialCapacity, 0);
   } //constructor

   public Vector(int initialCapacity, int capacityIncrement) {
      ...
      this.elementData = new Object[initialCapacity];
      ...
   } //constructor

   public synchronized Object get(int index) {
      if (index >= elementCount)
         throw new ArrayIndexOutOfBoundsException(index);

      return elementData[index];
   } //get()

   public synchronized boolean add(Object o) {
      ...
      ensureCapacityHelper(elementCount + 1);
      elementData[elementCount++] = o;
      return true;
   } //add()

   private void ensureCapacityHelper(int minCapacity) {
      int oldCapacity = elementData.length;
      if (minCapacity > oldCapacity) {
         Object oldData[] = elementData;
         int newCapacity = (capacityIncrement > 0) ? (oldCapacity + capacityIncrement) :
(oldCapacity * 2);
         if (newCapacity < minCapacity) {
            newCapacity = minCapacity;
         } //if
         elementData = new Object[newCapacity];
         System.arraycopy(oldData, 0, elementData, 0, elementCount);
      } //if
    } //ensureCapacityHelper()

   public Enumeration elements() {
      return new Enumeration() {
         int count = 0;
```

```
            public boolean hasMoreElements() {
               return count < elementCount;
            } //hasMoreElements()

            public Object nextElement() {
               synchronized (Vector.this) {
                  if (count < elementCount) {
                     return elementData[count++];
                  } //if
               } //synchronized
               throw new NoSuchElementException("Vector Enumeration");
            } //nextElement()
         }; //elements()
      } //class Vector
```

Einen einfacheren Weg zur **Traversierung eines `Vectors`** bietet die **Schnittstelle `Enumeration`** an.
Ein solches Objekt wird durch die Methode `elements()` zurückgegeben. Sie abstrahiert nochmals vom indizierten Zugriff, und stellt mit den Methoden `hasMoreElements()` und `nextElement()` generische Zugriffsprimitiven bereit, die so auch in anderen Collection-Klassen Anwendung finden.
Nebenbemerkung: Die Implementierung von `elements()` stellt ein (schönes) Beispiel für die Verwendung anonymer innerer Klassen dar.

Bei Modifikationen durch nebenläufig ausgeführte Threads am zugrundeliegenden `Vector` nachdem ein `Enumeration`-Objekt erzeugt wurde, kann es zu Konsistenzproblemen kommen. Daher wird seit dem JDK v1.2 die Verwendung der Schnittstelle `ListIterator` gegenüber der weiter angebotenen `Enumeration` empfohlen. Ihre Implementierung ist *fail-fast*, d.h. sie erzeugen ein Ausnahmeereignis-Objekt der Klasse `ConcurrentModificationException` im Falle nebenläufiger Modifikation. Zusätzlich erlaubt der `Iterator` selbst die Entnahme von Elementen aus der zugrundeliegenden Sammlung.
Der zu einer Collection gehörige Iterator wird über die (von `List` geerbte) Methode `listIterator()` geliefert.
Vorwärtsreferenz: Beispiel zur Nutzung eines `Iterators`

Mengen im mathematischen Sinne (d.h. duplikatfrei und ungeordnet) lassen sich durch die Klasse **`HashSet`** realisieren.
Mit den (ererbten) Methoden `addAll` (Vereinigung), `retainAll` (Durchschnitt), `removeAll` (Differenz) können die grundlegenden Mengenoperationen realisiert werden.




```
(1)import java.util.HashSet;
(2)import java.util.Iterator;
(3)
(4)public class HashSetEx1 {
(5)     public static void main(String[] args) {
(6)             HashSet hs1 = new HashSet();
(7)             HashSet hs2 = new HashSet();
(8)             HashSet hs3;
(9)
(10)            hs1.add( new String("a") );
(11)            hs1.add( new String("b") );
(12)            hs1.add( new String("c") );
(13)
(14)            hs2.add( new String("c") );
(15)            hs2.add( new String("d") );
(16)            hs2.add( new String("e") );
(17)
(18)            System.out.println("hs1="+hs1.toString() );
(19)            System.out.println("hs2="+hs2.toString() );
(20)
(21)            //union
(22)            hs3 = new HashSet( hs1 );
(23)            hs3.addAll(hs2);
(24)            System.out.println("UNION(hs1,hs2)="+hs3.toString() );
(25)
(26)            hs3 = null;
(27)            //difference
(28)            hs3 = new HashSet( hs1 );
(29)            hs3.removeAll( hs2 );
(30)            System.out.println("hs1 WITHOUT hs2 ="+hs3.toString() );
(31)
(32)            hs3 = null;
(33)            //intersection
(34)            hs3 = new HashSet( hs1 );
(35)            hs3.retainAll( hs2 );
(36)            System.out.println("INTERSECTION(hs1,hs2)="+hs3.toString() );
(37)
```

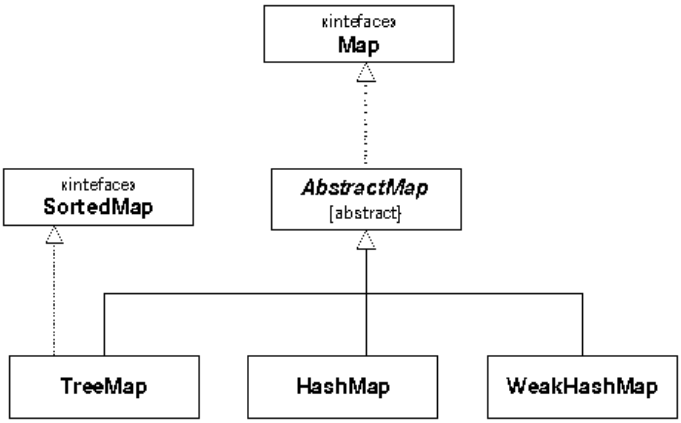
```
(38)            hs3 = null;
(39)            //symmetric difference
(40)            hs3 = new HashSet( hs1 );
(41)            hs3.addAll( hs2 );
(42)            HashSet tmp1 = new HashSet( hs1 );
(43)            tmp1.retainAll( hs2 );
(44)            hs3.removeAll( tmp1 );
(45)            System.out.println("Symmtetric Difference(hs1,hs2)="+hs3.toString() );
(46)
(47)            hs3 = null;
(48)            //cartesian product
(49)            hs3 = new HashSet();
(50)            String c1;
(51)
(52)            for( Iterator hs1It = hs1.iterator(); hs1It.hasNext(); ) {
(53)                    c1 = hs1It.next().toString();
(54)                    for( Iterator hs2It = hs2.iterator(); hs2It.hasNext(); )
(55)                    {
(56)                            hs3.add( new String( c1 + (hs2It.next()).toString() ) );
(57)                    } //for
(58)            } //while
(59)            System.out.println("Cartesian Product(hs1,hs2)="+hs3.toString() );
(60)    } //main()
(61)} //class HashSetEx1
```

Beispiel 73: Die grundlegenden Mengenoperationen HashSetEx1.java

Bildschirmausgabe:

```
hs1=[b, a, c]
hs2=[e, d, c]
UNION(hs1,hs2)=[b, a, e, d, c]
hs1 WITHOUT hs2 =[b, a]
INTERSECTION(hs1,hs2)=[c]
Symmtetric Difference(hs1,hs2)=[b, a, e, d]
Cartesian Product(hs1,hs2)=[ce, cd, cc, be, bd, bc, ae, ad, ac]
```

Das Beispiel zeigt neben Realisierungen der grundlegenden Mengenoperationen auch die Nutzung eines `Iterators` anstelle der veralteten `Enumeration`.



Die von der abstrakten Basisklasse `AbstractMap` abgeleiteten konkreten Klassen `HashMap`, `TreeMap` und `WeakHashMap` stellen eine von der `Collection`-Hierarchie losgelöste besondere Variante allgemeiner `Object`-Sammlungen zur Verfügung.
Ihnen allen gemein ist die Speicherung eines eineindeutigen Zugriffsschlüssels, zusätzlich zu den Nutzinformationen der verwalteten Objekte.

Die Klasse `HashMap` organisiert die abgelegten Objekte intern über eine Hashfunktion, `TreeMap` über einen rot-schwarz-Baum.
`WeakHashMap` ist prinzipiell identisch zur `HashMap`, jedoch werden Schlüssel-Wert-Paare entfernt, sobald die letzte externe Referenz auf den Schlüssel ungültig wird.

Das Beispielprogramm ermittelt die Auftrittshäufigkeit von einzelnen Worten oder Zahlen im Eingabestrom. Hierfür wird der bekannte `StreamTokenizer` zur entsprechenden Eingabefilterung benutzt.
Grundgedanke des realisierten Algorithmus ist es, das Wort oder die Zahl als Schlüssel in der `HashMap` zu nutzen, und die bisher ermittelten Vorkommen als Nutzinformation in einem `Integer`-Objekt abzulegen.

```
(1)import java.io.StreamTokenizer;
(2)import java.io.FileReader;
(3)import java.io.FileDescriptor;
(4)import java.io.IOException;
(5)import java.util.HashMap;
(6)import java.util.Map;
(7)
(8)public class WordCount {
(9)     private static final Integer ONE = new Integer(1);
(10)
(11)    public static void main(String[] args) {
(12)            Map m = new HashMap();
(13)
(14)            try {
(15)                    StreamTokenizer st = new StreamTokenizer( new FileReader( FileDescriptor.
in ) );
(16)                    int res;
(17)                    Integer freq;
(18)
(19)                    while( (res = st.nextToken()) == StreamTokenizer.TT_WORD || res ==
StreamTokenizer.TT_NUMBER) {
(20)                            if (res == StreamTokenizer.TT_WORD)
(21)                                    freq = (Integer) m.get( st.sval );
(22)            else
(23)                    freq = (Integer) m.get( new String(""+st.nval) );
(24)
(25)            if (res == StreamTokenizer.TT_WORD)
(26)                    m.put( st.sval, (freq==null ? ONE : new Integer(freq.intValue() + 1)));
(27)                            else
(28)                                    m.put( new String(""+st.nval), (freq==null ? ONE : new
Integer(freq.intValue() +1)));
(29)                    } //while
(30)        System.out.println(m.size()+" distinct words detected:");
(31)        System.out.println(m);
(32)            } catch (IOException ioe) {
(33)                    System.out.println("an IOException occurred\n"+ioe.getMessage() );
(34)            } //catch
(35)    } //main()
(36)} //class WordCount
```

Beispiel 74: Ermittelt die Auftrittshäufigkeit eines Wortes WordCount.java

Beispielablauf:
Die Datei *testfile.asc*:

```
the quick brown fox jumps over the lazy dog
1 2 3 123 1 2 X

$java wordCount < testfile
13 distinct words detected:
{X=1, lazy=1, fox=1, quick=1, 1.0=2, over=1, the=2, dog=1, 123.0=1, brown=1, 2.0=2, 3.0=1, jumps=1}
```

Die Positionierung eines Elements innerhalb einer der Hash-Sammlungen (`HashMap`, `HashSet` und `Hashtable`) wird während der Ausführung der `put`-Methode durch eine Hashfunktion, die auf die Methode `hashCode` der Klasse `Object` zurückgreift, bestimmt.
Beim Erzeugungsvorgang jedes Hash-...-Objekt kann der Speicherplatzbedarf und das Laufzeitverhalten über zwei Parameter beeinflußt werden. Die *initiale Kapzität* (engl. *initial capacity*) definiert die Anzahl der Einträge in der Hashtabelle, und der *Lastfaktor* (engl. *load factor*) den maximalen Füllungsgrad einer Hashtabelle bevor automatisch eine Kapazitätserweiterung durchgeführt wird.
Werden die beiden Parameter bei der Erzeugung nicht angegeben, so wird vorgabegemäß eine Datenstruktur mit Freiraum für elf Objekte, und einem Lastfaktor von 0.75 erzeugt. Mithin wird die erste Kapazitätserweiterung beim Einfügeversuch des neuten Elements durchgeführt.

Algorithmen auf Objektsammlungen vom Typ `Array`
... werden durch die Klasse `Arrays` in Form statischer Methoden angeboten.

- `asList(Object[])`
  Erzeugt eine `Liste` fester Größe aus einem Array beliebiger Objekte.
- `binarySearch(...)`
  Stellt für die verschiedenen Primitivtypen-Arrays und allgemeine `Object`-Arrays eine Suche nach beliebigen Inhaltswerten zur Verfügung.
- `equals(...)`
  Vergleicht zwei Arrays von identischem Typ auf inhaltliche Gleichheit.
- `fill(...)`
  Ersetzt alle Elemente eines Arrays durch das Übergebene.
- `sort(...)`
  Sortiert übergebenen Array mit einer modifizierten Variante des Quicksort-Algorithmus.

Algorithmen auf Objektsammlungen vom Typ `List`
... werden durch die Klasse `Collections` in Form statischer Methoden angeboten.

- `binarySearch(...)`
  Führt eine binäre Suche nach einem gegebenen Wert in einer Listen-artigen (d.h. Collection-Klasse welche die Schnittstelle `List` implementiert) durch.
  Die gegenwärtige Implementierung nimmt jedoch `AbstractSequentialLists` und deren Subklassen (in der Standard-API realisiert: `LinkedList`) explizit von dieser Mimik aus, und realisiert für diese Sammlungstypen eine sequentielle Suche.
- `copy(List, List)`
  Kopiert Listenelemente in eine andere Liste.
- `Enumeration(Collection)`
  Liefert für jeden Collection-Typ eine `Enumeration` zur Traversierung zurück.
  Anmerkung: Leider existiert derzeit keine Äquivallent das die modernere und sicherere Variante `Iterator` zurückliefern würde.
- `fill(List, Object)`
  Ersetzt alle Elemente einer `Liste` durch das gegebene `Object`.
- `min(...)` und `max(...)`
  Liefern das Minimum, bzw. Maximum, einer `Liste`, ggf. unter Anwendung einer speziellen Vergleichsoperation. Die Suche wird nach der Methode des sequentiellen Schwellenwertes durchgeführt.
- `nCopies(Object)`
  Liefert eine `Liste` mit *n* Kopien eines bestimmten Objekts.
- `reverse(List)`
  Kehrt die Elementordnung für eine `Liste` um.
- `reverseOrder()`
  Liefert ein `Comparator`-Objekt zurück welches die Anordnung der `Liste` umkehrt.
- `shuffle(List)`
  Permutiert Objekte der `Liste`.
- `sort(...)`
  Sortiert `Liste` nach einem modifizierten Mergesort-Verfahren.
  Die Verwendung des Mergesort-Algorithmus ist nicht zwingend, API-Implementierungen können ihn gegen durch Algorithmen ersetzen. Einzige Bedingung hierbei ist die Stabilität. D.h. die Positionserhaltung gleicher Elemente; sie sollen durch den Sortiervorgang nicht verschoben werden.
  In der Implementierung wird zunächst aus der `Liste` in ein `Object`-Array erzeugt, auf dem dann die `sort`-Methode der Klasse `Arrays` aufgerufen wird, die den Mergesort dann für `Objects` implementiert.
- `synchronizedCollection(Collection)`, `synchronizedList(List)`, `synchronizedMap(Map)`, `synchronizedSet(Set)`, `synchronizedSortedMap(SortedMap)` und `synchronizedSortedSet(SortedSet)`
  Liefern eine threadsichere Variante der jeweiligen Eingabedatenstruktur zurück.
- `unmodifiableCollection(Collection)`, `unmodifiableList(List)`, `unmodifiableMap(Map)`, `unmodifiableSet(Set)`, `unmodifiableSortedMap(SortedMap)` und `unmodifiableSortedSet(SortedSet)`
  Liefern eine unveränderbare Sicht auf die jeweilige Eingabedatenstruktur zurück.

Anmerkung zur Verwendung der beiden Sortierverfahren *Mergesort* und *Quicksort* in den Klassen `Arrays` und `Collections`:
Beim ersten Hinsehen verwundert die Implementierung zweier verschiedener Sortierverfahren ein wenig, nicht zuletzt wegen des Wechsels des Sortierverfahrens in der Implementierung der `sort`-Methoden der Klasse `Arrays`. Wird dort doch für alle Sortieroperationen auf Primitivtypen-Arrays eine modifizierte Quicksort-Variante eingesetzt, jedoch für `Object`-Arrays zu Mergesort übergegangen.
Dies liegt in der Natur der beiden Sortierverfahren begründet. Quicksort ist eines der bekanntesten und vermutlich auch am besten erforschtesten Sortierverfahren; mit guten Leistungsdaten im allemeinen Falle. Jedoch kann der Sortiervorgang im schlechtesten Falle bis zu n^2 Schritte benötigen. Insbesondere für große n ist daher Mergesort -- mit seinen garantierten $n \log(n)$ Schritten effizienter.
Allerdings benötigt Quicksort deutlich weniger Speicherplatz (eigentlich nur den Stack durch die rekursiven Aufrufe), während Mergesort für seinen Ablauf stets den doppelten Speicherbedarf der zu sortierenden Datenstruktur für sich reklamiert.
Aus praktischen Gründen, nicht zuletzt wegen des uniformen Laufzeitverhaltens, wurde dem speicherplatzintensiveren Mergesort der Vorzug gegeben. Da in realen Anwendungen in den seltensten Fällen reine Primitivtypen-Arrays sortiert werden, nutzt der Anwender überwiegend die vorhandenen Mergesort-Implementierungen, die jedoch in (begründeten) Einzelfällen durchaus durch eigene Routinen ersetzt werden können.

3.2.5 Parametrische Polymorphie / Generics

Die wirkungsmächtigste Neuerung der Java Version 1.5 bildet die Einführung der sog. *Generics*, welche das Feld der Template-basierten Metaprogrammierung und mithin die parametrische Polymorphie eröffnen.

Das sinnvollste Einsatzfeld dieses Mechanismus bildet die Anwendung auf die Vorhandenen Klassen der Collection-API zur Realisierung typsicherer Objektsammlungen.

Die allgemeine Syntax der Generics erfordert die Nachstellung, des durch spitze Winkelklammern eingeschlossenen Typnamen, nach dem Namen der so typisierten Sammlung. So wird eine Liste, die ausschließlich Objekte des Typs `String` enthält als `List<String>` definiert.

Als Resultat einer so definierten Sammlungsklasse wird bereits zum Übersetzungszeitpunkt die Konformität abgeprüft. Daher wird bei jedem Versuch Daten in eine solch konkretisierte Sammlung einzufügen geprüft, ob diese

unter Beachtung der Typrestriktion kompatibel zum in der Deklaration angegebenen Inhaltstyp sind.

Konventionsgemäß erlaubt Java ausschließlich die Festlegung von Klassen als Inhaltstypen von Objektsammlungen; Primitivtypen sind von der Verwendung ausgeschlossen.

Das nachfolgende Beispiel zeigt die Definition einer Sammlung des Typs `LinkedList`, deren Inhaltselemente auf Ausprägungen des Typs `Integer` beschränkt sind.

```
(1)import java.util.LinkedList;
(2)
(3)public class GenTest1 {
(4)     public static void main(String[] args) {
(5)             LinkedList<Integer>  integerList  = new LinkedList<Integer>();
(6)
(7)             integerList.add(new Integer(42));
(8)             integerList.add( 84 );
(9)
(10)            System.out.println( integerList );
(11)    } //main()
(12)} //class GenTest1
```

Beispiel 75: Typisierung einer generischen Liste GenTest1.java

Das Beispiel zeigt die Definition einer typisierten `LinkedList`, für die bereits zum Übersetzungszeitpunkt geprüft wird, daß ausschließlich Ausprägungen der Klasse `Integer` der Liste hinzugefügt werden.
Diese Restriktion verhindert jedoch, vermöge der Nutzung des dynamischen Boxings nicht, daß Ausprägungen des Primitivtyps `int` direkt übergeben werden, da diese für den Programmierer transparent in Objekte des Type `Integer` umgewandelt werden.
Versuche Objekte oder Werte anderen Typs in die Liste aufzunehmen werden bereits zum Übersetzungszeitpunkt erkannt und als Fehler gemeldet.

Die Einführung typisierter Sammlungen bedingt auch die entsprechende Adaption der Hilfsklassen zur Traversierung. vor diesem Hintergrund zeigt das nachfolgende Beispiel die Typisierung eines Iterators. Als Konsequenz dieser Konkretisierung des Iterators liefert der Aufruf der Methode `next` statt der generischen `Object`-Instanz eine Ausprägung des Parametertyps `String`.

```
(1)import java.util.LinkedList;
(2)import java.util.Iterator;
(3)
(4)public class GenTest2 {
(5)     public static void main(String[] args) {
(6)             LinkedList<String>  nameList  = new LinkedList<String>();
(7)
(8)             nameList.add( "Berta" );
(9)             nameList.add( "Anton" );
(10)            nameList.add( "Cesar" );
(11)
(12)            Iterator<String> listIterator = nameList.iterator();
(13)            String value;
(14)            while(listIterator.hasNext()) {
(15)                    value = listIterator.next();
(16)                    System.out.println( value );
(17)            } //while
(18)    } //main()
(19)} //class GenTest2
```

Beispiel 76: Typisierung eines Iterators GenTest2.java

Neben der Anwendung auf bestehende Klassen des Collection-Frameworks können Generics auch für eigenerstelle Klassen eingesetzt werden.
Das Beispiel zeigt die Definition der Klasse `TopTen`, welche eine Liste von Einträgen beliebigen Typs zusammen mit einer Plazierung gestattet. Der Inhaltstyp wird hierbei als Polymorphieparameter zum Konstruktionszeitpunkt der `TopTen`-Objekte festgelegt. Innerhalb der Klasse wird zur Verwaltung der Daten eine `TreeMap` eingesetzt, die entsprechend der Initialisierung des `TopTen`-Objekts erzeugt wird um auch auf dieser Ebene die Typkonformität zu gewährleisten.

```
(1)import java.util.TreeMap;
(2)import java.util.Collection;
(3)import java.util.Iterator;
(4)
(5)public class GenTest3 {
(6)     public static void main(String[] args) {
(7)             TopTen<String> music = new TopTen<String>();
(8)             music.addAtPosition(1, "St. Anger/Metallica");
(9)             music.addAtPosition(3, "20 Jahre - Nena feat. Nena/Nena");
(10)            music.addAtPosition(2, "9/Eros Ramazotti");
(11)
(12)            music.print();
(13)    } //main()
```

```
(14)} //class GenTest3
(15)
(16)class TopTen<Type> {
(17)    private TreeMap<Integer,Type> content;
(18)    public TopTen() {
(19)            content = new TreeMap<Integer,Type>();
(20)    } //constructor
(21)    public boolean addAtPosition(int pos, Type value) {
(22)            if (content.size() == 10) {
(23)                    return false;
(24)            } else {
(25)                    content.put(new Integer(pos), value);
(26)                    return true;
(27)            } //else
(28)    } //addAtPosition()
(29)    public Type getFromPosition(int pos) {
(30)            return content.get(new Integer(pos));
(31)    } //getFromPosition()
(32)    public void print() {
(33)            Iterator<Type> i = content.values().iterator();
(34)            while (i.hasNext()) {
(35)                    System.out.println( i.next() );
(36)            } //while()
(37)    } //print()
(38)} //class TopTen
```

Beispiel 77: Anwendung von Generics für eigene Klassen GenTest3.java

Neben der Angabe von Klassen zur Typisierung einer generischen Aufzählung können, mit derselben Mächtigkeit, auch Aufzählungstypen angegeben werden. Das nachfolgende Beispiel zeigt eine Anwendung:

```
(1)import java.util.LinkedList;
(2)import java.util.HashMap;
(3)
(4)public class EnumTest4 {
(5)public static void main(String argv[]) {
(6)     enum DayOfWeek {monday, tuesday, wednesday, thursday, friday,
(7)     saturday, sunday};
(8)
(9)     LinkedList<DayOfWeek> allDays = new LinkedList<DayOfWeek>();
(10)    allDays.add(DayOfWeek.monday);
(11)    allDays.add(DayOfWeek.tuesday);
(12)    allDays.add(DayOfWeek.wednesday);
(13)    allDays.add(DayOfWeek.thursday);
(14)    allDays.add(DayOfWeek.friday);
(15)    allDays.add(DayOfWeek.saturday);
(16)    allDays.add(DayOfWeek.sunday);
(17)
(18)    HashMap<DayOfWeek,Double> sales = new HashMap<DayOfWeek,Double>();
(19)
(20)    sales.put(DayOfWeek.monday, 3000.0 );
(21)    sales.put(DayOfWeek.tuesday, 5000d );
(22)    sales.put(DayOfWeek.thursday,  7500D );
(23)    sales.put(DayOfWeek.wednesday,  (double) 8000 );
(24)    sales.put(DayOfWeek.friday,  55000.0 );
(25)    sales.put(DayOfWeek.saturday, new Double("10000"));
(26)    sales.put(DayOfWeek.sunday, (new Double("0")).doubleValue() );
(27)
(28)    for (DayOfWeek w : allDays)
(29)            System.out.println(w+" "+sales.get(w));
(30)    } //main()
(31)} //class EnumTest4
```

Beispiel 78: Aufzählungstypen als Polymorphieparameter EnumTest4.java

Zwar vermeidet der Einsatz der Generics die Anwendung unsicherer expliziter Typkonversionen fast völlig, jedoch ist ihre Notwendigkeit in Einzelfällen nach wie vor gegeben.
Das nachfolgende Beispiel zeigt einen solchen Fall. Es definiert eine als `Vector` angelegte Sammlung von Objekten des Typs `Person`. Gemäß der Substitutionsregeln können auch Spezialisierungen der `Person`, mithin Ausprägungen der davon abgeleiteten Klasse `Mitarbeiter`, typkonform der Sammlung hinzugefügt werden.
Da die Methode get im allgemeinen Falle Ausprägungen der Klasse `Object` und unter Einsatz der Generics Ausprägungen des Parametertyps (in diesem Falle: `Person`) liefert würde die abgespeicherte `Mitarbeiter`-Instanz lediglich als Ausprägung der Klasse `Person` aus der Sammlung entnommen werden. Um eine korrekte Weiterverarbeitung (z.B. in Form des Zugriffs auf das Attribut `personalnummer`) als `Mitarbeiter` zu gewährleisten ist daher eine explizite Typkonversion notwendig.

```
(1)import java.util.Vector;
(2)
(3)public class GenTest4 {
(4)     public static void main(String[] args) {
(5)             Vector<Person> persL = new Vector<Person>();
(6)
(7)             Person p = new Person();
(8)             p.name = "Max Mustermann";
(9)             persL.add(p);
(10)
(11)            Mitarbeiter m = new Mitarbeiter();
(12)            m.name = "John Doe";
(13)            m.personalnummer = "1234";
(14)            persL.add(m);
(15)
(16)            Person p2 = persL.get(0);
(17)            System.out.println( p2.name );
(18)
(19)            Mitarbeiter m2 = (Mitarbeiter) persL.get(1);
(20)            System.out.println( m2.name );
(21)            System.out.println( m2.personalnummer );
(22)    } //main()
(23)} //class GenTest4
(24)
(25)class Person {
(26)    public String name;
(27)} //class Person
(28)
(29)class Mitarbeiter extends Person {
(30)    public String personalnummer;
(31)} //class Mitarbeiter
```

Beispiel 79: Explizite Typwandlung trotz Generics GenTest4.java

3.2.6 Reflection API

Mit der JDK-Version 1.1 wurde die Möglichkeit geschaffen zur Laufzeit Informationen über die verwendeten Informationsstrukturen zu gewinnen. Genaugenommen handelt es sich hier um Information über Information, die üblicherweise als *Metainformation* bezeichnet wird.
Bereits seit der Version 1.0 existierte mit der Klasse `Class` ein Mechanismus, der Möglichkeit zur Ermittlung des Typs im Stile der aus C++ bekannten *Runtime Type Identification* bietet.

Der **Begriff Laufzeittyp** meint den konkreten Typen eines Attributes oder einer Variable (im Allgemeinen: einer referenzierten typisierten Speicherposition). Dieser Typ muß nicht zwingend über die gesamte Ausführungszeit mit dem zum Definitionszeitpunkt festgelegten identisch sein. Er kann sich durch typkonforme Neuzuweisung ändern.

Die Klasse `Class`

Zur Motivation: Bisher war es uns mit dem `instanceof`-Operator nur möglich zu entscheiden ob ein gegebenes Objekt Ausprägung einer bestimmten Klasse ist. Hierfür mußte der Namen der Klasse zum Ausführungszeitpunkt des Operators bekannt sein.

Die Methode `getClass()` welche, da von `Object` geerbt, auf allen Java-Objekten zur Verfügung steht behebt dieses Manko.
Sie liefert ein Objekt der Klasse `Class` zurück welches -- neben zahlreichen weiteren Informationen -- den Namen der Klasse zur Laufzeit verfügbar werden läßt.
`Class` ist im Paket `java.lang` angesiedelt, und steht daher ohne explizite `import`-Anweisung in allen Java-Programmen zur Verfügung. Zu jeder Klasse im System existiert je ein solches `Class`-Objekt.

```
(1)public class RefEx1 {
(2)     public static void main(String[] args) {
(3)             Person p1 = new Person();
(4)             Mann m1 = new Mann();
(5)
(6)             System.out.println("runtime class of variable p1="+ p1.getClass().getName() );
(7)             System.out.println("runtime class of variable m1="+ m1.getClass().getName() );
(8)
(9)             p1 = m1; //up-cast is typesafe
(10)            System.out.println("runtime class of variable p1="+ p1.getClass().getName() );
(11)    } //main()
(12)} //class RefEx1
(13)
(14)class Person {
```

```
(15)} //class Person
(16)
(17)class Mann extends Person {
(18)} //class Mann
(19)
(20)class Frau extends Person {
(21)} //class Frau
```

Beispiel 80: Ermittlung der Laufzeitklasse RefEx1.java

Bildschirmausgabe:

```
$java RefEx1
runtime class of variable p1=Person
runtime class of variable m1=Mann
runtime class of variable p1=Mann
```

Die Anwendung definiert neben der Hauptklasse drei weitere Klassen `Person`, `Mann` und `Frau`. Sie sind in der Weise organisiert, daß `Mann` und `Frau` als Subklassen von `Person` realisiert sind.
Zunächst werden Variablen vom Typ `Person` und `Mann` definiert und mit Ausprägungen der Definitionstypen initialisiert. Durch den Methodenaufruf `getName()` auf dem durch `getClass()` zurückgelieferten `Class`-Objekt kann der Name der Laufzeitklasse im Klartext ermittelt werden. So liefert die Variable `p1` welche auf eine `Person`-Ausprägung verweist erwartungsgemäß den Namen dieser Klasse zurück; für die Variable `m1` und die Klasse `Mann` gilt dies analog.
Durch der Zuweisung des `Mann`-Objekts `m1` an die Variable `p1` (vom Typ `Person`) verändert sich deren Laufzeittyp zu `Mann`.

Im vorangegangenen Beispiel wird ein Objekt zur Ermittlung der zugehörigen Laufzeitklasse benutzt. Ist der Klassenname bekannt, so kann durch den Methodenaufruf `forName(String)` das zur Klasse mit dem übergebenen Namen gehörige `Class`-Objekt ermittelt werden.

```
(1)public class RefEx2 {
(2)     public static void main(String[] args) {
(3)             try {
(4)                     Class c1 = Class.forName("Person");
(5)                     System.out.println("Name of class:"+c1.getName() );
(6)             } catch (ClassNotFoundException cnfe) {
(7)                     System.out.println("cannot find class\n"+cnfe.toString()+"\n"+cnfe.
getMessage() );
(8)             } //catch
(9)     } //main()
(10)} //class RefEx2
(11)
(12)class Person {
(13)} //class Person
```

Beispiel 81: Ermittlung eines Class-Objekts über den Klassennamen RefEx2.java

Bildschirmausgabe:

```
$java RefEx2
Name of class:Person
```

Die Klasse `Person` wird geladen, und per `getName()` ihr `Name` -- erwartungsgemäß `Person` -- ausgegeben.
Kann die Klasse nicht gefunden werden, so wird ein `ClassNotFoundException` Ausnahmeereignis-Objekt erzeugt.

Um unnötigen Tippaufwand zu sparen exisistiert mit *T*`.class` eine abkürzende Schreibweise für alle Javatypen *T* aus dem Typsystem.
Beispiel:

```
Class c1 = Person.class;
Class c2 = int.class;
Class c3 = Double[].class;
```

Hinweis: Aus historischen Gründen liefert die Methode `getName()` für alle Array-Typen eine etwas unschöne abkürzende Syntax, die sich am im Class-File verwendeten Format orientiert:
Zunächst das Präfix `[` für jede Array-Schachtelung, gefolgt von ...
`B` für `byte`
`C` für `char`
`D` für `double`
`F` für `float`
`I` für `int`
`J` für `long`
L*classname*; für eine Klasse oder Schnittstelle
`S` für `short`
`Z` für `boolean`.

so liefert `(new int[3][4][5][6][7][8][9]).getClass().getName()` als Namen die Zeichenkette `[[[[[[[I` zurück.

Die **Erzeugung neuer Objekte aus einem existierenden `Class`-Objekt** erfolgt durch die Methode `newInstance()`. Sie ruft intern den parameterlosen Vorgabekonstruktor der durch das `Class`-Objekt vertretenen Klasse auf.
Die `newInstance`-Methode unterstützt keine Aufrufe parametrisierter Konstruktoren. Für diesen Anwendungsfall sollte auf die Klasse `Constructor` der (moderneren) Reflection API zurückgegriffen werden.

```
(1)public class RefEx3 {
(2)     public static void main(String[] args) {
(3)             Person p1 = new Person(); //normal object creation
(4)             p1.sayHello();
(5)
(6)             try {
(7)                     Person p2 = (Person) Class.forName("Person").newInstance();
(8)                     p2.sayHello();
(9)             } catch (ClassNotFoundException cnfe) {
(10)                    System.out.println("cannot find class\n"+cnfe.toString()+"\n"+cnfe.
getMessage() );
(11)            } catch (InstantiationException ie) {
(12)                    System.out.println("an InstantiationException occured\n"+ie.toString()
+"\n"+ie.getMessage() );
(13)            } catch (IllegalAccessException iae) {
(14)                    System.out.println("an IllegalAccessException occured\n"+iae.toString()
+"\n"+iae.getMessage() );
(15)            } //catch
(16)    } //main()
(17)} //class RefEx3
(18)
(19)class Person {
(20)    public void sayHello() {
(21)            System.out.println("hello from person!");
(22)    } //sayHello()
(23)} //class Person
```

Beispiel 82: Erzeugung eines neuen Objekts mit der Methode newInstance RefEx.java

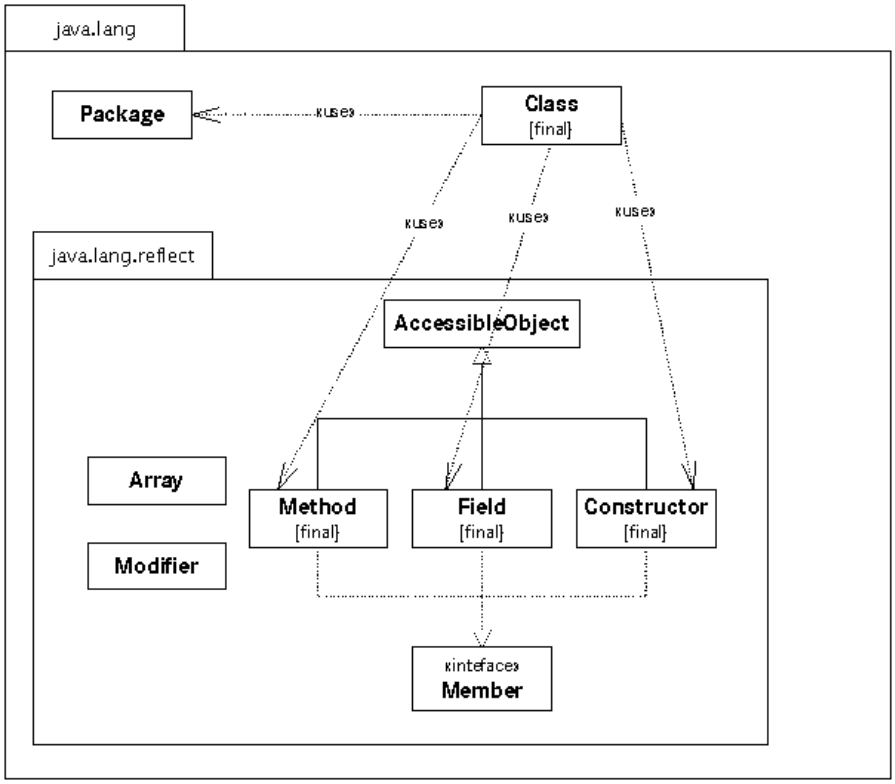
Das Programm gibt zweimal die Zeichenkette *hello from person!* am Bildschirm aus. Die Erzeugung des zweiten Personen-Objektes (`p2`) erfolgt durch die Kopplung der Methoden `forName` und `newInstance`. Letztere liefert eine Ausprägung der Klasse `Object` zurück, weshalb -- um die Methode `sayHello` der davon abgeleiteten Klasse `Person` aufrufen zu können -- die explizite Typumwandlung erfolgen muß.

Ferner stellt die Klasse `Class` weitere Methoden zur Untersuchung der Charakteristika definierter Klassen bereit:

- `getSuperclass()`
  Liefert die Superklasse einer gegebenen Klasse. Für die Klasse `java.lang.Object`, Schnittstellen, alle Primitivtypen und `void` wird `null` zurückgeliefert.
- `getInterfaces()`
  Liefert die durch eine Klasse direkt implementierten Schnittstellen; ererbte Schnittstellen werden nicht geliefert.
- `isInterface()`
  `true` falls das `Class`-Objekt eine Schnittstelle ist, andernfalls `false`.
- `isInstance(Object)`
  Entspricht dem `instanceof`-Operator.
- `isAssignableFrom(Class)`
  Prüft ob die Zuweisung des als Parameter übergebenen Klassentypen an das Objekt auf dem diese Methode aufgerufen wird möglich ist.
  Die Methode bietet damit direkten Zugriff auf den JVM-Opcode `checkcast`; ohne jedoch bei Unmöglichkeit der Zuweisung ein Ausnahmeereignis-Objekt zu generieren.

Das Reflection-Paket

Wie bereits angedeutet wurde die Klasse `Class` mit Einführung der Reflection API in der Java Version 1.1 stark erweitert. Sie schlägt -- über die Rückgabetypen der neu geschaffenen Methoden -- eine Brücke zu den Klassen des Reflection-Paketes `java.lang.reflect`.



Die `Reflection`-API ist als Subpaket von `java.lang` organisiert. Ihre Klassen dienen lediglich zur Inspektion existierender Klassen einer Java-Laufzeit-Umgebung. Daher sind die Konstruktoren der von `AccessibleObject` abgeleiteten Klassen, sowie der der Klasse `Array`, als `private` deklariert; Ausprägungen dieser Klassen können ausschließlich durch die virtuelle Maschine erzeugt werden.

Im Folgenden werden zunächst die Methoden zur Ermittlung verschiedenster Informationen am konkreten Codebeispiel vorgestellt.
Die verschiedenen Methoden zur Einflußnahme auf bereits erzeugte Metainformationsobjekte werden im Anschluß diskutiert.

Ermittlung Klassen- und Schnittstellen-bezogener Information

Klassen und Schnittstellen werden durch die Reflection-API gleichbehandelt. Die Unterscheidung kann durch Methode `isInterface` auf jedem `Class`-Objekt getroffen werden.
Ferner stehen zur Verfügung:

- Modifier -- `getModifiers`
  Der Aufruf liefert die verschiedenen Details als `int`-Wert, der die gesetzten Modifier Bit-codiert enthält. Zur Entschlüsselung der einzelnen Modifier sollten die Methoden der Reflection-Klasse `Modifier` eingesetzt werden, um auch hinsichtlich zukünftiger Java-Spracherweiterungen stabilen Code zu produzieren.
  Aufruf im Beispiel
  Die Umsetzung im Beispiel nutzt die Implementierung der `toString()`-Methode der Klasse `Modifier`, welche die in der *Java Language Specification* definierte kanonische Repräsentation zurückliefert.
- Komponententyp-Ermittlung -- `getComponentType()`
  Ermöglicht die Ermittlung des zugrundeliegenden Typen falls es sich bei der Klasse um einen Array handelt. Bei geschachtelten Arrays ist diese Methode u.U. mehrfach anzuwenden.
  Verwendung im Beispiel
- Intern deklarierte Klassen und Schnittstellen -- `getDeclaredClasses()`
  Liefert die deklarierierten (nicht anonymen) inneren Klassen und inneren Schnittstellen als `Class`-Array zurück.
  Verwendung im Beispiel
- Name -- `getName()`
  Liefert den vollqualifizierten Namen der Klasse oder Schnittstelle.
  Verwendung im Beispiel
- Paketzugehörigkeit -- `getPackage`
  Liefert das definiertende Paket der Klasse als Ausprägung der Klasse `Package`. Bei Klassen im Vorgabepaket, d.h. ohne explizite package-Definition, wird `null` zurückgegeben.
  Verwendung im Beispiel

Ermittlung Attribut-bezogner Informationen

Der Zugriff auf Attribute und Attribut-spezifische Informationen -- in der Java-Terminologie `Felder` genannt -- (wie Namen, Sichtbarkeitsbereich und Typ) wird durch die Methoden `getField(String)`, für ein namentlich bekanntes Feld, und `getFields()` für *alle* Attribute einer Klasse, realisiert. In beiden Fällen ist wird genau ein, bzw. ein Array von Objekten der Reflection-Klasse `Field` zurückgeliefert.
Beide Methoden berücksichtigen dabei *alle* Attribute/Felder einer Klasse, einschließlich der von der Superklasse und deren Superklassen ererbten. Zur Ermittlung der direkt auf dem betrachteten `Class`-Objekt definierten Attribute können Methoden `getDelcaredField(String)` bzw. `getDelcaredFields()` eingesetzt werden.

Verwendung im Beispiel

`Field`-Objekte bieten Zugriff auf die verschiedenen deklarativen Charakteristika eines Attributs:

- Modifier (siehe Kapitel 2) -- `getModifiers()`.
  Der Aufruf liefert die verschiedenen Details als `int`-Wert, der die gesetzten Modifier Bit-codiert enthält. Zur Entschlüsselung der einzelnen Modifier sollten die Methoden der Reflection-Klasse `Modifier` eingesetzt werden, um auch hinsichtlich zukünftiger Java-Spracherweiterungen stabilen Code zu produzieren.
  Die Verwendung im Beispiel nutzt die Implementierung der `toString()`-Methode der Klasse `Modifier`, welche die in der *Java Language Specification* definierte kanonische Repräsentation zurückliefert.
- Typ (einer der Primitivtypen, ein Array, eine beliebige zugreifbare Klasse oder eine Schnittstelle) -- `getType()`.
  Die Methode liefert ein Objekt vom Typ `Class` zurück, auch wenn die Primitivtypen nicht als (first class) Objekte realisiert sind.
  Zur Ermittlung des tatsächlichen Typs sind weitere Operationen auf dem `Class`-Objekt notwendig. `isArray()` klärt ob es sich bei dem Attributtypen um einen Array-Typ handelt. Da Java mehrdimensionale Arrays als geschachtelte eindimensionale Pendants realisiert muß in einem solchen Falle per `getComponentType()` weiternavigiert werden. Diese Methode liefert wiederum ein `Class`-Objekt zurück.
  Der vollqualifizierte (d.h. im Falle der Existenz um den durch das Paket gebildeten Namensraum angereichte) Name eines Typen wird durch `getName()` zurückgeliefert.
  Die Methode printAttribute des Beispiels zeigt die Ermittlung des Typs durch die vorgestellten Methoden im Überblick.
- Name -- `getName()`.
  Liefert den Attributnamen als `String`.

Ermittlung Operationen-bezogener Information

Anmerkung: Zwar spricht die Java-Reflection-API hier konsequent von *Methoden*, bietet jedoch keine Zugriffsmöglichkeiten auf den Implementierungskörper einer Operation. Ermittelbar sind daher nur die Signatur, die Sichtbarkeitsmodifier sowie der Rückgabetyp und, falls existent, im Ausführungsverlauf ausgelößte Ausnahmeereignisse; mithin: die *Operation*.

Die Reflection-API trennt Konstruktoren und sonstige Operationen voneinander. Daher existieren mit `getConstructor(Class[])` und `getConstructors()` bzw. `getMethod(String, Class[])` und `getMethods()` jeweils eigene Methoden zu ihrer Ermittlung.
Zurückgeliefert wird jedoch in allen Fällen genau ein, bzw. ein Array von, `Method`-Objekt(en).

- Modifier (siehe Kapitel 2) -- `getModifiers()`
  Der Aufruf liefert die verschiedenen Details als `int`-Wert, der die gesetzten Modifier Bit-codiert enthält. Zur Entschlüsselung der einzelnen Modifier sollten die Methoden der Reflection-Klasse `Modifier` eingesetzt werden, um auch hinsichtlich zukünftiger Java-Spracherweiterungen stabilen Code zu produzieren.
  Die Umsetzung im Beispiel nutzt die Implementierung der `toString()`-Methode der Klasse `Modifier`, welche die in der *Java Language Specification* definierte kanonische Repräsentation zurückliefert.
- Übergabeparameter -- `getParameterTypes()`
  Liefert einen Array von `Class`-Objekten, die die formalen Parameter der Operation repräsentieren.
  Verwendung im Beispiel
- Rückgabetyp (einer der Primitivtypen, ein Array, eine beliebige zugreifbare Klasse oder eine Schnittstelle) -- `getReturnType()`.
  Verwendung im Beispiel
- Auslösbare Ausnahmeereignisse -- `getExceptionTypes()`
  Liefert einen Array von `Class`-Objekten der deklarierten Exceptions der Operation.
  Verwendung im Beispiel

```
(1)import java.lang.reflect.Constructor;
(2)import java.lang.reflect.Method;
(3)import java.lang.reflect.Modifier;
(4)import java.lang.reflect.Member;
(5)import java.lang.reflect.Field;
(6)import java.lang.ClassNotFoundException;
(7)
(8)public class ShowClass {
(9)     public static void main(String[] args) {
(10)    try {
(11)            printClass( Class.forName(args[0]) );
(12)    } catch (ClassNotFoundException cnfe) {
(13)            System.out.println("cannot find class "+args[0]);
(14)    } //catch
(15)    } //main()
(16)// ----------------------------------------------------------------------------
(17)    public static void printClass(Class c) {
(18)    Package pck = c.getPackage();
(19)    if ( pck != null )
(20)            System.out.println("package "+pck.getName()+";" );
(21)
(22)    if (c.isInterface()) {
(23)            System.out.print(Modifier.toString(c.getModifiers()) + " "+ c.getName());
(24)            }       else
(25)            System.out.print(Modifier.toString(c.getModifiers()) + " class " +
(26)                    c.getName() +
(27)                    " extends " + c.getSuperclass().getName());
(28)
(29)            Class[] interfaces = c.getInterfaces();
```

```
(30)            if ((interfaces != null) && (interfaces.length > 0)) {
(31)            if (c.isInterface())
(32)                    System.out.println(" extends ");
(33)            else
(34)                    System.out.print(" implements ");
(35)            for(int i = 0; i < interfaces.length; i++) {
(36)                    if (i > 0)
(37)                            System.out.print(", ");
(38)                    System.out.print(interfaces[i].getName());
(39)            } //for
(40)            } //if
(41)            System.out.println("\n{");
(42)
(43)            Constructor[] constructors = c.getDeclaredConstructors();
(44)            if ( constructors.length > 0 ) {
(45)                    System.out.println("   //Constructors");
(46)                    for(int i = 0; i < constructors.length; i++)
(47)                    printOperation(constructors[i]);
(48)            } //if
(49)            constructors = null;
(50)
(51)            Class[] classes = c.getDeclaredClasses();
(52)            if (classes.length > 0) {
(53)                    System.out.println("   //Inner Classes");
(54)                    for (int i=0; i < classes.length; i++) {
(55)                            if (classes[i].isInterface() == false ) {
(56)                                    printClass(classes[i] );
(57)                                    System.out.print("\n");
(58)                            } //if
(59)                    } //for
(60)                    System.out.println("   //Inner Interfaces");
(61)                    for (int i=0; i < classes.length; i++) {
(62)                            if (classes[i].isInterface() == true ) {
(63)                                    printClass(classes[i] );
(64)                                    System.out.print("\n");
(65)                            } //if
(66)                    } //for
(67)            } //if
(68)            classes = null;
(69)
(70)            Field[] fields = c.getDeclaredFields();
(71)            if ( fields.length > 0) {
(72)                    System.out.println("   //Attributes");
(73)                    for(int i = 0; i < fields.length; i++)
(74)                    printAttribute(fields[i]);
(75)            } //if
(76)            fields = null;
(77)
(78)
(79)            Method[] operations = c.getDeclaredMethods();
(80)            if ( operations.length > 0 ) {
(81)                    System.out.println("   //Operations");
(82)            for(int i = 0; i < operations.length; i++)
(83)            printOperation(operations[i]);
(84)            } //if
(85)            operations = null;
(86)
(87)            System.out.print("} //end ");
(88)            if ( c.isInterface() )
(89)                    System.out.print("interface ");
(90)            else
(91)                    System.out.print("class ");
(92)            System.out.print( c.getName() );
(93)    } //printClass(Class)
(94)// ----------------------------------------------------------------------------
(95)    public static String typename(Class t) {
(96)    String brackets = "";
(97)            while(t.isArray()) {
(98)            brackets += "[]";
(99)            t = t.getComponentType();
(100)           } //while
(101)      return t.getName() + brackets;
(102)   } //typename(Class)
(103)// ----------------------------------------------------------------------------
(104)   public static String modifiers(int m) {
(105)           if (m == 0)
(106)                   return "";
(107)      else
(108)           return Modifier.toString(m) + " ";
(109)   } //modifiers(int)
(110)// ----------------------------------------------------------------------------
(111)   public static void printAttribute(Field f) {
```

```
(112)           System.out.print("  " + modifiers(f.getModifiers()) + typename(f.getType()) + " " +
f.getName() + ";     //"+f.getType().getName()+" is ");
(113)
(114)           if ( (f.getType()).isInterface() )
(115)                   System.out.println("an interface");
(116)           else {
(117)                   if ( f.getType().isPrimitive() )
(118)                           System.out.println("a primitive type");
(119)                   else {
(120)                           if ( f.getType().isArray() )
(121)                                   System.out.println("an array");
(122)                           else
(123)                                   System.out.println("a class");
(124)                   } //else
(125)           } //else
(126)   } //printAttribute(Field)
(127)// ------------------------------------------------------------------------------
(128)   public static void printOperation(Member member) {
(129)           Class returntype=null, parameters[], exceptions[];
(130)
(131)      if (member instanceof Method) {
(132)           Method m = (Method) member;
(133)         returntype = m.getReturnType();
(134)         parameters = m.getParameterTypes();
(135)         exceptions = m.getExceptionTypes();
(136)      } else {
(137)           Constructor c = (Constructor) member;
(138)         parameters = c.getParameterTypes();
(139)         exceptions = c.getExceptionTypes();
(140)      } //else
(141)
(142)      System.out.print("  " + modifiers(member.getModifiers()) + ((returntype!=null)? typename
(returntype)+" " : "") + member.getName() + "(");
(143)      for(int i = 0; i < parameters.length; i++) {
(144)           if (i > 0)
(145)                   System.out.print(", ");
(146)
(147)         System.out.print(typename(parameters[i]));
(148)      } //for
(149)
(150)      System.out.print(")");
(151)
(152)      if (exceptions.length > 0)
(153)           System.out.print(" throws ");
(154)
(155)      for(int i = 0; i < exceptions.length; i++) {
(156)           if (i > 0)
(157)                   System.out.print(", ");
(158)
(159)         System.out.print(typename(exceptions[i]));
(160)      } //for
(161)
(162)      System.out.println(";");
(163)   } //printOperation(Member)
(164)} //class ShowClass
(165)
(166)// -----------------------------------------
(167)interface TestInterface {
(168)} //interface TestInterface
(169)
(170)class TestClass {
(171)   public          TestInterface  i1;
(172)   private                 TestClass  c1;
(173)   protected       int p1;
(174)   int[]                   ta1;
(175)   int[][]                 tam1;
(176)
(177)   class Inner {
(178)           int i;
(179)           class InnerInner {
(180)                   int j;
(181)           } //class InnerInner
(182)   } //class Inner
(183)   interface InnerInterface {
(184)   } //interface InnerInterface
(185)
(186)   private TestClass testc() {
(187)           return new TestClass() {
(188)                   public void hello() {
(189)                           System.out.println("overridden test!");
(190)                   } //hello()
(191)           }; //class TestClass
```

```
(192)   } //test()
(193)} //class TestClass
```

Beispiel 83: Erfragen verschiedener Klassen-spezifischer Informationen durch die Reflection-API ShowClass.java

Bildschirmausgabe:

```
 class TestClass extends java.lang.Object
{
   //Constructors
  TestClass();
   //Inner Classes
 class TestClass$Inner extends java.lang.Object
{
   //Constructors
  TestClass$Inner(TestClass);
   //Inner Classes
 class TestClass$Inner$InnerInner extends java.lang.Object
{
   //Constructors
  TestClass$Inner$InnerInner(TestClass$Inner);
   //Attributes
  int j;     //int is a primitive type
  final TestClass$Inner this$1;     //TestClass$Inner is a class
} //end class TestClass$Inner$InnerInner
   //Inner Interfaces
   //Attributes
  int i;     //int is a primitive type
  final TestClass this$0;     //TestClass is a class
} //end class TestClass$Inner
   //Inner Interfaces
abstract static interface TestClass$InnerInterface
{
} //end interface TestClass$InnerInterface
   //Attributes
  public TestInterface i1;     //TestInterface is an interface
  private TestClass c1;     //TestClass is a class
  protected int p1;     //int is a primitive type
  int[] ta1;     //[I is an array
  int[][] tam1;     //[[I is an array
   //Operations
  private TestClass testc();
} //end class TestClass
```

Das Ausgabe enthält, erwartungsgemäß, die implizite Superklasse `java.lang.Object`, welche durch den Übersetzer automatisch gesetzt wird, als explizite Angabe nach dem `extends`-Schlüsselwort.
Die inneren Klassen erscheinen in ihrer JVM-internen Namensgebung mit dem trennenden Dollarsymbol $. Auffallend ist die explizite Wiedergabe der `this`-Variable und des automatisch generierten Konstruktors mit dem Typ der äußeren Klasse als Übergabeparameter.
Für Array-Typen erfolgt die Benennung ebenfalls gemäß der JVM-internen Konvention.

Erzeugung neuer Objekte und Modifikationen bestehender mit der Reflection API

Neben den bisher vorgestellten Mechanismen zur Introspektion bestehender Strukturen kann die Reflection API auch zur Erzeugung neuer Objekte und Operation auf diesen eingesetzt werden. Hierfür stehen neben Methoden zum lesenden und schreibenden Zugriff auf Attribute auch Möglichkeiten zur Ausführung von Methoden auf den Objekten zur Verfügung.

Erzeugung von Objekten

Die Erzeugung neuer Objekte aus bestehenden `Class`-Ausprägungen war bereits mit Methoden der `Class`-Klasse möglich, sofern zur Objekterzeugung der parameterlose Standardkonstruktor verwendet wird.
Zur Instanziierung neuer Objekte unter Nutzung eines bestehenden parametrisierten Konstruktors ist die Verwendung der Reflection-Klasse `Constructor` unabdingbar.

Der Aufruf eines parametrisierten Konstruktors vollzieht sich in drei Schritten:

1. Ermittlung des `Class`-Objekts für die neu zu erzeugende Instanz
2. Erzeugung eines neuen `Constructor`-Objekts mit der `getConstructor(Class[])`-Methode der Klasse `Class`
3. Objekterzeugung durch die `newInstance(Object[])`-Methode der `Constructor`-Klasse

```
(1)import java.lang.reflect.Constructor;
(2)import java.lang.reflect.InvocationTargetException;
(3)import java.awt.Rectangle;
(4)
(5)class SampleInstance {
(6)     public static void main(String[] args) {
(7)      Rectangle rectangle;
(8)      Class rectangleDefinition;
(9)      Class[] intArgsClass = new Class[] {int.class, int.class};
(10)      Integer height = new Integer(12);
(11)      Integer width = new Integer(34);
(12)      Object[] intArgs = new Object[] {height, width};
(13)      Constructor intArgsConstructor;
(14)
(15)      try {
(16)                    rectangleDefinition = Class.forName("java.awt.Rectangle");
(17)            intArgsConstructor = rectangleDefinition.getConstructor(intArgsClass);
(18)
(19)          System.out.println ("Constructor: " + intArgsConstructor.toString());
(20)       Object object = null;
(21)
(22)                    object = intArgsConstructor.newInstance(intArgs);
(23)            System.out.println ("Object: " + object.toString());
(24)            rectangle = (Rectangle) object;
(25)      } catch (ClassNotFoundException e) {
(26)          System.out.println("A ClassNotFoundException occurred!\n"+e.toString()+"\n"+e.
getMessage() );
(27)      } catch (NoSuchMethodException e) {
(28)          System.out.println("A NoSuchMethodException occurred!\n"+e.toString()+"\n"+e.
getMessage() );
(29)      } catch (InstantiationException e) {
(30)            System.out.println("An InstantiationException occurred!\n"+e.toString()+"\n"+e.
getMessage() );
(31)      } catch (IllegalAccessException e) {
(32)            System.out.println("An IllegalAccessException occurred!\n"+e.toString()+"\n"+e.
getMessage() );
(33)      } catch (IllegalArgumentException e) {
(34)            System.out.println("An IllegalArgumentException occurred!\n"+e.toString()+"\n"+e.
getMessage() );
(35)      } catch (InvocationTargetException e) {
(36)            System.out.println("An InvocationTargetException occurred!\n"+e.toString()+"\n"+e.
getMessage() );
(37)            } //catch
(38)   } //main()
(39)} //class SampleInstance
```

Beispiel 84: Erzeugung eines AWT-Rechtecks über einen parametrisierten Konstruktor mit der Reflection-API SampleInstance.java

Bildschirmausgabe:

```
Constructor: public java.awt.Rectangle(int,int)
Object: java.awt.Rectangle[x=0,y=0,width=12,height=34]
```

Das Beispiel spiegelt die drei Erstellungsschritte wieder:
Zunächst wird über den Namen (Aufruf: `forName(...)`) das zur Klasse gehörige `Class`-Objekt erfragt.
Die Methode `getConstructor(...)` liefert das Konstruktorenobjekt welches die als Parameter übergebene Signatur aufweist.
Instanziiert wird das neue Objekt durch die Methode `newInstance`, ausgeführt auf dem `Constructor`-Objekt mit den definierten Übergabeparametern.

Attributzugriff

... geschieht über Objekte der Klasse `Field`.
Für alle Primitivtypen existieren lesende Methoden, die einheitlich mit `get`*T*`(Object)` signiert sind. *T* ist einer der Primitivtypen ist. Der Rückgabetyp ist jeweils *T*.
Werte von objektartigen Attributen können mittels `get(Object)` erfragt werden; zurückgegeben wird hier eine Ausprägung von `Object`.

Symmetrisch sind die schreiben Zugriffe als `set`*T*`(Object, T)` realisiert. Mit `set(Object, Object)` steht eine Methode zur Ablage objektartiger Attribute zur Verfügung.

Das nachfolgende Beispiel zeigt die Verwendung der Lese- und Schreibmethoden.

```
(1)import java.lang.reflect.Field;
(2)
(3)public class AttributeAccess {
(4)     public static void main(String[] args) {
(5)             Test o1 = new Test();
(6)             o1.setI(42);
(7)
(8)             System.out.println( "i="+o1.getI() );
(9)
(10)            try {
(11)                    Field iAttribute = (o1.getClass()).getField("i");
(12)                    System.out.println( "i="+iAttribute.getInt(o1) );
(13)
(14)                    iAttribute.setInt(o1, 50 );
(15)                    System.out.println( "i="+iAttribute.getInt(o1) );
(16)            } catch (NoSuchFieldException e) {
(17)                    System.out.println("A NoSuchFieldException occurred!\n"+e.toString()+"\n"+e.
getMessage() );
(18)                    e.printStackTrace();
(19)            } catch (IllegalAccessException e) {
(20)                    System.out.println("A IllegalAccessException occurred!\n"+e.toString()
+"\n"+e.getMessage() );
(21)                    e.printStackTrace();
(22)            } //catch
(23)    } //main()
(24)} //class AttributeAccess
(25)
(26)class Test {
(27)    public int i;
(28)
(29)    public void setI(int i) {
(30)            this.i = i;
(31)    } //setI()
(32)
(33)    public int getI() {
(34)            return i;
(35)    } //getI()
(36)} //class Test
```

Beispiel 85: Attributzugriff mit Methoden der Reflection-Klasse Field AttributeAccess.java

Bildschirmausgabe:

```
i=42
i=42
i=50
```

Ausführen von Methoden

Die Klasse Method weist neben den bisher vorgestellten Funktionalitäten zur Inspektion auch mit der Methode invoke (Object, Object[]) einen generischen Mechanismus zur Ausführung beliebiger Methoden auf.

Das folgende Beispiel zeigt den Aufruf der überladenen Methode hello über die invoke-Methode der Reflection-API.

```
(1)import java.lang.reflect.Method;
(2)import java.lang.reflect.InvocationTargetException;
(3)
(4)public class MethodExecution {
(5)     public static void main(String[] args) {
(6)             test o1 = new test();
(7)
(8)             try {
(9)                     Method helloMethod = (o1.getClass()).getMethod("hello", null);
(10)                    helloMethod.invoke(o1, null);
(11)
(12)                    Class[] formalParameters = { String.class };
(13)                    helloMethod = (o1.getClass()).getMethod("hello", formalParameters );
(14)                    Object[] argumentList = { "stranger" };
(15)                    helloMethod.invoke(o1, argumentList);
(16)            } catch (NoSuchMethodException e) {
(17)                    System.out.println("A NoSuchMethodException occurred!\n"+e.toString()
+"\n"+e.getMessage() );
(18)                    e.printStackTrace();
(19)            } catch (IllegalAccessException e)      {
(20)                    System.out.println("A NoSuchMethodException occurred!\n"+e.toString()
+"\n"+e.getMessage() );
(21)                    e.printStackTrace();
(22)            } catch (InvocationTargetException e) {
(23)                    System.out.println("An InvocationTargetException occurred!\n"+e.toString()
```

```
+"\n"+e.getMessage() );
(24)                    e.printStackTrace();
(25)            } //catch
(26)    } //main()
(27)} //class MethodExecution
(28)
(29)class Test {
(30)    public void hello() {
(31)            System.out.println("simple hello!");
(32)    } //hello()
(33)
(34)    public void hello(String name) {
(35)            System.out.println("hello "+name);
(36)    } //hello()
(37)} //class Test
```

Beispiel 86: Ausführung zweiter Methoden durch die Reflection-API MethodExecution.java

Bildschirmausgabe:

```
simple hello!
hello stranger
```

Weitere Anwendundungsbeispiele

Unter Einsatz der Reflection-API lassen sich generische Programme entwickeln, die auf beliebigen Typen operieren. Mit diesem Mechanismus läßt sich ein Verhalten vergleichbar der parametrischen Polymorphie der C++-Templates nachbilden.

```
(1)import java.lang.reflect.Array;
(2)
(3)public class GrowableArrayTest {
(4)     public static void main(String[] args) {
(5)             int[] ia = {1};
(6)             System.out.println("ia consists of "+ia.length+" elements");
(7)             ia = (int[]) arrayGrow(ia);
(8)             System.out.println("ia consists of "+ia.length+" elements");
(9)
(10)            int[] ib = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
(11)            System.out.println("ib consists of "+ib.length+" elements");
(12)            ib = (int[]) arrayGrow(ib);
(13)            System.out.println("ib consists of "+ib.length+" elements");
(14)
(15)            Person[] pa = new Person[3];
(16)            pa[0] = new Person("hans");
(17)            pa[1] = new Person("franz");
(18)            pa[2] = new Person("fritz");
(19)            System.out.println("pa consists of "+pa.length+" elements");
(20)            pa = (Person[]) arrayGrow(pa);
(21)            System.out.println("pa consists of "+pa.length+" elements");
(22)    } //main()
(23)
(24)    static Object arrayGrow(Object oldArray) {
(25)            Class arrayClass = oldArray.getClass();
(26)            if ( !arrayClass.isArray() )
(27)                    return null;
(28)
(29)            Class componentType = arrayClass.getComponentType();
(30)
(31)            int oldLength = Array.getLength(oldArray);
(32)            int newLength = ( ((oldLength * 1.1) > oldLength + 1) ? (int) (oldLength * 1.1) :
(oldLength + 1) );
(33)
(34)            Object newArray = Array.newInstance(componentType, newLength );
(35)
(36)            System.arraycopy(oldArray, 0, newArray, 0, oldLength);
(37)            return newArray;
(38)    } //arrayGrow()
(39)} //class growableArrayTest
(40)
(41)class Person {
(42)    protected String name;
(43)
(44)    public Person (String name) {
(45)            this.name = name;
(46)    } //constructor
(47)} //class Person
```

Beispiel 87: Ein wachsender Array GrowableArrayTest.java

Bildschirmausgabe:

```
ia consists of 1 elements
ia consists of 2 elements
ib consists of 20 elements
ib consists of 22 elements
pa consists of 3 elements
pa consists of 4 elements
```

Das Beispiel implementiert dynamische, durch expliziten Methodenaufruf, wachsende Arrays.
Nach Ermittlung des Class-Objekts (durch getClass()) und des Komponententyps des Arrays (getComponentType ()) wird ein zweiter -- der neue -- Array durch die newInstance-Methode der Klasse Array erzeugt. Als Übergabeparameter wird der Komponententyp und die Länge des neuen Arrays übergeben. Die Länge des neuen Arrays entspricht der des Alten, um zehn Prozent, jedoch mindestens ein Element, erhöht.
Abschließend wird der Inhalt des alten in den neu erzeugten Array kopiert. Hierzu wird die sehr performant ausgeführte native Methode System.arraycopy eingesetzt.

Durch die Konstruktion des Method-Objekts und der invoke-Methode läßt sich auch das aus C/C++ bekannte Verhalten der **Funktionszeiger** nachbilden.
Das Beispiel zeigt zweimaligen Aufruf derselben Methoden printTable. Im Körper der Methode wird jeweils die als Parameter übergebene Methode ausgeführt.
Im ersten Aufruf wird das Quadrat der Zahlen von Eins bis Zehn gebildet, im Zweiten hingegen die Wurzel gezogen.

```
(1)import java.lang.reflect.Method;
(2)import java.lang.reflect.InvocationTargetException;
(3)
(4)public class MethodPointerTest {
(5)     public static void main(String[] args) {
(6)             try {
(7)                     printTable(1,10, MethodPointerTest.class.getMethod("square", new Class[]
{double.class} ));
(8)                     printTable(1,10, java.lang.Math.class.getMethod("sqrt", new Class[]{ double.
class} ));
(9)             } catch (NoSuchMethodException e) {
(10)                    System.out.println("A NoSuchMethodException occurred!\n"+e.toString()
+"\n"+e.getMessage() );
(11)            } //catch
(12)    } //end main()
(13)
(14)    public static double square (double x) {
(15)            return (x*x);
(16)    } //square()
(17)
(18)    public static void printTable(double from, double to, Method f) {
(19)            System.out.println(f);
(20)            for (double x=from; x <= to; x++) {
(21)                    System.out.print( x );
(22)                    try {
(23)                            Object[] args = { new Double(x) };
(24)                            Double d = (Double) f.invoke(null,args);
(25)                            double y = d.doubleValue();
(26)                            System.out.println("  |  "+y);
(27)                    } catch (InvocationTargetException e) {
(28)                            System.out.println("An InvocationTargetException occurred!\n"+e.
toString()+"\n"+e.getMessage() );
(29)                    } catch (IllegalAccessException e) {
(30)                            System.out.println("An IllegalAccessException occurred!\n"+e.
toString()+"\n"+e.getMessage() );
(31)                    } //catch
(32)            } //for
(33)    } //printTable()
(34)} //class MethodPointerTest
```

Beispiel 88: Verweise auf Methoden in Java MethodPointerTest.java

Bildschirmausgabe:

```
public static double MethodPointerTest.square(double)
1.0  |  1.0
2.0  |  4.0
3.0  |  9.0
4.0  |  16.0
5.0  |  25.0
6.0  |  36.0
7.0  |  49.0
```

```
8.0  |  64.0
9.0  |  81.0
10.0  |  100.0
public static strictfp double java.lang.Math.sqrt(double)
1.0  |  1.0
2.0  |  1.4142135623730951
3.0  |  1.7320508075688772
4.0  |  2.0
5.0  |  2.23606797749979
6.0  |  2.449489742783178
7.0  |  2.6457513110645907
8.0  |  2.8284271247461903
9.0  |  3.0
10.0  |  3.1622776601683795
```

3.2.7 Abstract Windowing Toolkit (AWT)

Das AWT stellt den wohl interessantesten Teil eines (interaktiven) Java-Programmes dar -- die Benutzerinteraktion durch eine graphische Oberfläche.
Ziel der AWT-Entwicklung war es, die verschiedenen graphischen Primitiven wie Buttons, Menüs etc. *plattformunabhängig* anzubieten.
Zur Realisierung des *Look and Feel* bieten sich vier verschiedene Ansätze an:

- Umsetzung ausschließlich der gemeinsamen graphischen Komponenten aller Zielplattformen.
  Hierdurch entsteht automatisch ein GUI des kleinsten gemeinsamen Nenners, welches -- insbesondere wenn man kleinste Endgeräte wie PDAs und Mobiltelephone betrachtet -- praktisch nur schwerlich einsetzbar sein dürfte.
- Umsetzung absolut aller Komponenten der verschiedenen Zielplattformen auf jeder Zielplattform.
  Hierdurch entsteht automatisch ein GUI das die Vereinigungsmenge aller betrachteten Plattformen abbildet. Differerieren die betrachteten Plattformen stark in ihrer Leistungsfähigkeit, so zieht dies mit unter sehr hohen Aufwand in der späteren Umsetzung (Emulation) der einzelnen Primitve auf der realen Plattform nach sich.
- Kapselung der plattformspezifischen API durch die neue Zugriffsschicht.
  Hierdurch steht dem Anwender auf der realen Plattform die dort übliche graphische Schnittstelle in Aussehen und Verhalten zur Verfügung.
  Alternativ könnte auch eine vollständig neue GUI definiert werden, die dann auf allen realen Plattformen unterstützt wird. Diesen Ansatz verfolgt die Java 2 Plattform mit *SWING*.
- Bündelung und Umsetzung einer Menge *sinnvoller*, gängiger Konzepte.
  Hierdurch steht auf der realen Zielplattform immer dieselbe GUI zur Verfügung, die je nach tatsächlicher Ausgestaltung der Plattform mehr oder minder aufwendig zu implementieren ist.

Das AWT bietet neben den graphischen Primitivoperationen zum Zeichnen von einfachen geometrischen Objekten wie Linien, Rechtecken, Kreisen; Fülloperationen und Methoden zur Textausgabe einen Mechanismus zur ereignisbasierten Ablaufsteuerung an, der es erlaubt auf externe Ereignisse wie Mauseingaben zu reagieren. Ferner bietet sie die weitegehend bekannten GUI-Grundelemente wie Fenster, Dialogboxen, Menüs, ... an. Zur Entwicklung komplexerer Anwendungen steht Funktionalität zur Graphikbearbeitung und Audiowiedergabe zur Verfügung.

... und SWING?

Mit der Java Version 1.1 wurde das AWT vollständig überarbeitet, insbesondere die oftmals als Achillesferse praktisch verwendbarer Applikationen empfundene Ereignisbehandlung (engl. *event handling*). Seither verwirklicht auch Java das bereits von anderen Oberflächensystemen wie NextStep oder Windows NT bekannte *Delegation Based Event Handling*. Der Terminus beschreibt die Möglichkeit der AWT externe Ereignisse (Mausclicks, Fenster-Schließen, ...) an beliebige Objekte zur Behandlung weiterzureichen.
Ebenfalls mit der JDK-Version 1.1 wurde eine zweite Bibliothek zur Oberflächenprogrammierung vorgestellt: SWING. Sie geht deutlich über den in der AWT realisierten Funktionsumfang hinaus, und wird daher heute überwiegend zur Implementierung professioneller Anwendungen herangezogen.
Vorwärtsreferenz: Sie wird in Kapitel 3.2.8 Swing beschrieben.

Struktur des AWT

Das gesamte AWT ist im Standard-API-Paket `java.awt` zusammengefaßt. Es stellt die grundlegenden graphischen Primitive zur Verfügung:

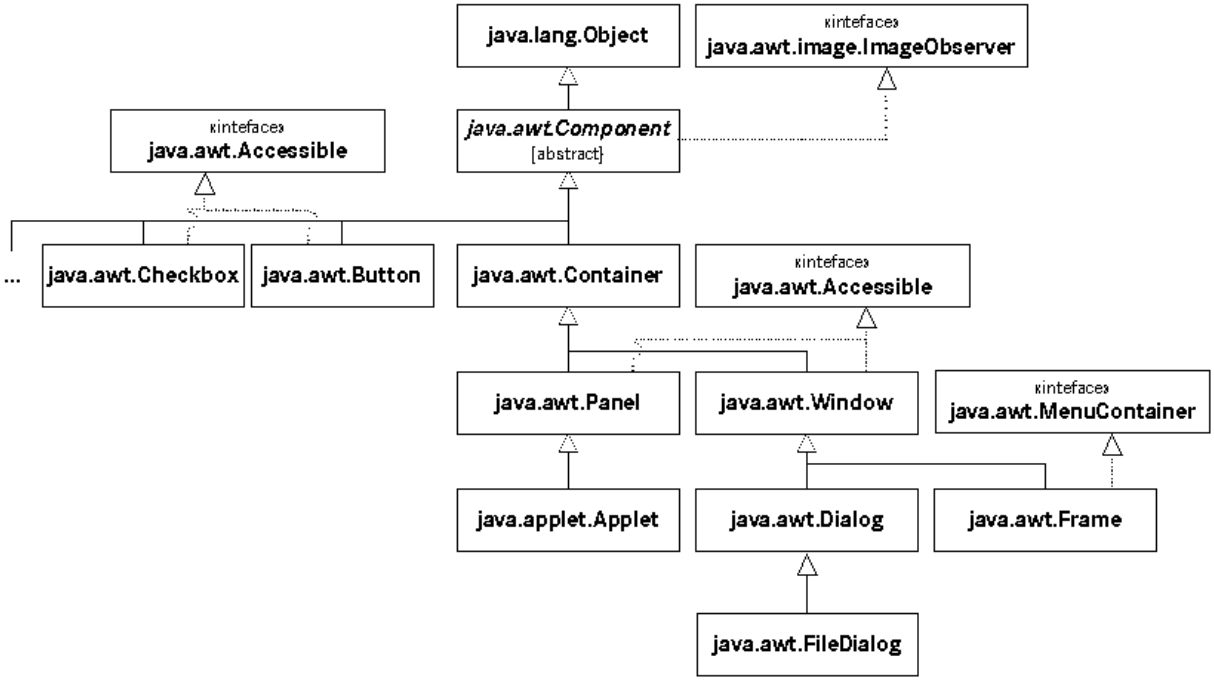
- Auswahlfelder `List`
  enthält auswählbare Elemente.
- Beschriftungstexte `Label`
  Einzeilige Beschriftung.
- Bildlaufleisten (scroll bars)
  Verschiebebalenken zum Scrollen.
- `Choice`
  Drop-down-Liste oder Optionsmenü.
- `FileDialog`
  Dialogbox zur Auswahl von Dateien.
- leere Komponente `Canvas`
  Kann zur Anzeige eigener Zeichnungen verwendet werden.

- Schaltflächen `Button`
  Graphische Schaltfläche, die vom Benutzer angeklickt werden kann.
- Textbereiche (text area)
  Mehrzeiliges Textfeld für Benutzereingaben und zur Textanzeige.
- Textfelder (text field)
  Einzeliges Textfeld für Benutzereingaben.
- Umschaltknopf `Checkbox`
  kann selektiert oder deselektiert werden.

Als Menü-Komponenten stehen zur Verfügung:

- `Menu`
  Pull-down oder frei plazierbares Menü (*Abreißmenü*).
- `CheckboxMenuItem`
  Umschaltknopf in einem Menü.
- `MenuBar`
  ... kann Pull-down-Menüs aufnehmen.
- `MenuItem`
  Menü-Schaltfläche.
- `PopupMenu`
  ... öffnet sich automatisch bei Mauskontakt.

Die dargestellten Komponenten dienen zunächst zum Aufbau der Benutzeroberfläche (buttons, Listen, etc.) oder zur Strukturierung der Fenster (scroll bars, menues, etc.). Die dritte Kategorie bilden die Zeichenbereiche.



Ausgangspunkt jeder AWT-basierten Applikation ist die -- bereits aus der `Applet`-Hierarchie bekannte -- `Container`-Struktur.
In Containern werden die verschiedenen graphischen AWT-Primitive zusammengefaßt; Container können hierarchisch strukturiert sein, d.h. weitere Container enthalten. Der oberste Container einer Applikation bildet gleichsam die sammelnde Ordnungsstruktur. Die AWT verfügt über vier vordefinierte Containertypen:

- `Dialog`
  Leichtgewichtiges Fenster, auch *Pop-Up-Fenster*; zumeist für Meldungen oder Eingaben benutzt.
  Mit der Klasse `FileDialog` ist ein Standard-Dialog zur Erfragung eines Dateinamens realisiert.
- `Frame`
  Vergleichbar mit dem gleichnamigen (X)HTML-Konstrukt ermöglichen es Frames eigenständige Fenster innerhalb einer Applikation zu organisieren.
- `Panel`
  Containertyp, der kein eigenes Fenster öffnet.
- `Window`
  Container zur Aufnahme von Frames.

Anmerkung: Rückblickend mag es (anfänglich) verwundern, daß die Graphikwiedergabe in Applets ohne Kenntnis und Anwendung der Containerstruktur möglich war. Dies ist jedoch nur auf den ersten Blick der Fall. Zunächst stellt der Browser oder Appletviewer das Hauptfenster nebst den Menüleisten zur Verfügung -- insofern existiert ein Container zur Aufnahme des Applets bereits durch seine Ausführungsumgebung. Zusätzlich stellt das Applet selbst, durch seine Plazierung in der Vererbungshierarchie unterhalb der Klasse `java.awt.Panel`, einen eigenständigen Container dar.

Layout-Manager

Ein weiterer Ausweis der Zielsetzung plattformunabhängigen Designs des AWT ist die verwirklichte Philosophie zur Plazierung der graphischen Elemente am Bildschirm. Während plattformspezifische GUI-Bibliotheken zumeist die angebotenen Primitive mit absoluten Koordinaten im Anwendungsfenster verankern verfolgt Java hier einen

gegensätzlichen Weg.
Durch den Programmierer wird ausschließlich die Plazierung der Komponente im Verhältnis zu anderen Komponenten angegeben, die endgültige Ausrichtung am Schirm nimmt eine zusätzliche AWT-Einheit -- der *Layout-Manager* -- vor. Die Orientierung der Primitiven innerhalb einer Komponente erfolgt weiterhin durch absolute Positionierung.

Im AWT stehen folgende vordefinierte Layout-Manager zur Verfügung:

- Flow -- `FlowLayout`
  Ordnet die enthaltenen Komponenten zentriert von links nach rechts an.
  Eignet sich besonders gut zur Ausrichtung von Button-Leisten.
- Border -- `BorderLayout`
  Orientiert Komponenten in fünf verschiedene Bereiche.
- Grid -- `GridLayout`
  Plaziert jedes Element an einer Position innerhalb eines Quadratgitters freiwählbarer Größe.
- Card -- `CardLayout`
  Organisiert enthaltene Container als Stapel von Karten, die jeweils einzeln angezeigt werden können.
- GridBag -- `GridBag`
  Richtet Komponenten verschiedener Dimensionen horizontal und vertikal aus.

Anmerkungen:

- Wird die Methode `setLayout(LayoutManager)` mit dem Übergabeparameter `null` aufgerufen, so wird kein Layoutmanager verwendet, sondern die Positionierung der graphischen Elemente vollständig dem Anwender überlassen.
- In realen Anwendungen wird üblicherweise massiver Gebrauch von der Möglichkeit der Schachtelung von Layoutmanagern gemacht.
  Hierfür wird der Inhalt eines `Frames` durch mehrere `Panels` definiert, die unterschiedliche Layoutmanager nutzen.
- Durch Implementierung der Schnittstelle `LayoutManager` lassen sich eigene Layout-Manager erzeugen.

Ein erstes Fenster

```
(1)import java.awt.Frame;
(2)
(3)public class AWTEx1 {
(4)     public static void main(String[] args) {
(5)             Frame wnd = new Frame("Einfaches Fenster");
(6)
(7)             wnd.setSize(400,300);
(8)             wnd.setVisible(true);
(9)     } //end main()
(10)} //class AWTEx1
```

Beispiel 89: Ein einfaches Fenster mit AWT AWTEx1.java

Bildschirmausgabe unter MS-Windows

Das Beispiel erzeugt einen `Frame` mit dem Titel *Einfaches Fenster*.
Durch die, von `Component` ererbte, Methode `setSize(int, int)` wird dem leeren `Frame` die Größe von 400 mal 300 Pixeln zugewiesen.
Zum Abschluß muß der Anzeigevorgang per `setVisible(boolean)` (ebenfalls von `Component` ererbt) explizit angestoßen werden.

Die Beiden Methoden `setSize` und `setVisible` greifen auf die visuellen Eigenschaften der Komponente zu, diese und weitere Methoden zu Modifikation der graphischen Erscheinung stehen auf allen Subklassen von `Component` zur Verfügung.

- `setSize(Dimension)` und `setSize(int, int)`
  Ändert Größe der Komponente.
- `setVisible(boolean)`
  Zeigt Komponente an, oder verbirgt sie.
- `setLocation(int, int)`

Verschiebt Komponente an angegeben Position.

- `setBounds(int, int, int, int)`
  Kombination aus `setSize` und `setLocation`, ändert Lage und Größe der Komponente.
- `requestFocus()`
  Beantragt Eingabefous.
- `transferFocus()`
  Gibt Eingabefocus an nächste Komponente weiter.
- `setEnabled(boolean)`
  Aktiviert oder deaktiviert Komponente. Nur aktivierte Komponenten reagieren auf Eingaben und können Ereignisse verarbeiten. Vorgabegemäß sind alle Komponenten aktiviert.
- `setFont(Font)`
  Setzt Zeichensatz der Komponente.
- `setComponentOrientation(ComponentOrientation)`
  Setzt sprach- und kulturspezifische Orientierung der Komponente. Nach dem übergebenen Orientierungsschema werden alle Komponentenelemente und Texte ausgerichtet.
  Zur Verfügung stehten derzeit `LEFT_TO_RIGHT` und `RIGHT_TO_LEFT`
- `setCursor(Cursor)`
  Setzt Mauszeiger-Bild.
- `setBackground(Color)`
  Setzt Hintergrundfarbe der Komponente.
- `setForeground(Color)`
  Setzt Vordergrundfarbe der Komponente.

Auf Ausprägungen der Klasse `Frame` sind ferner verfügbar:

- `setState(int)`
  Schaltet zwischen den Zuständen `Frame.NORMAL` (Vorgabe) und `Frame.ICONIFIED` (verkleinert) um.
- `setIconImage(Image)`
  Definiert Icon zur Anzeige bei minimierter Darstellung.
- `setMenuBar(MenuBar)`
  Definiert Pull-down-Menü zur Verwendung innerhalb des Frames.
- `setResizeable(boolean)`
  Erlaubt oder Verbietet Größenänderung durch Anwender.
- `setTitle(String)`
  Setzt Titelzeile des Frames.

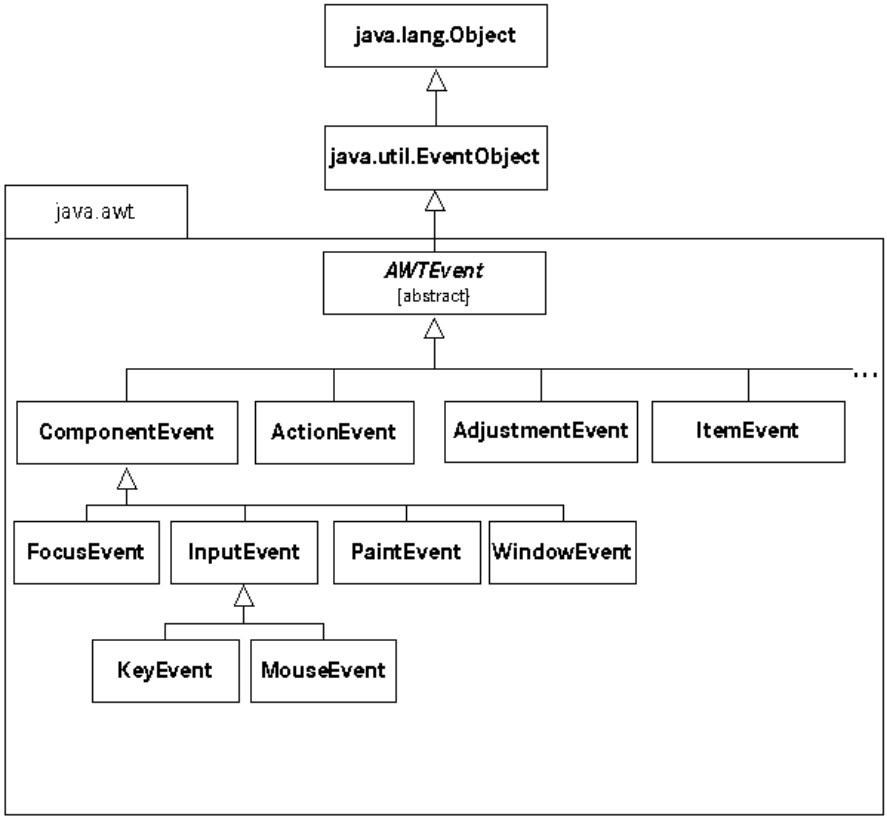
Bei der Ausführung fällt zunächst die Existenz der plattformüblichen Standardmenüs auf. Sie werden automatisch durch die Ausführungsumgebung zur Verfügung gestellt und mit Funktionalität versehen.
Jedoch zeigt sich, daß es nicht möglich ist das Fenster mit den üblichen Betriebssystem-spezifischen Methoden (*Close*-Button, Auswahl des entsprechenden Menüs, Tastenkombination, etc.) zu schließen. Erst der Abbruch des Prozesses der virtuellen Maschine terminiert die Applikation.
Denn anders als die üblichen Vorgabeaktionen ist das hierfür erforderliche Applikationsverhalten durch den Anwender selbst bereitzustellen.

Ereignisbehandlung

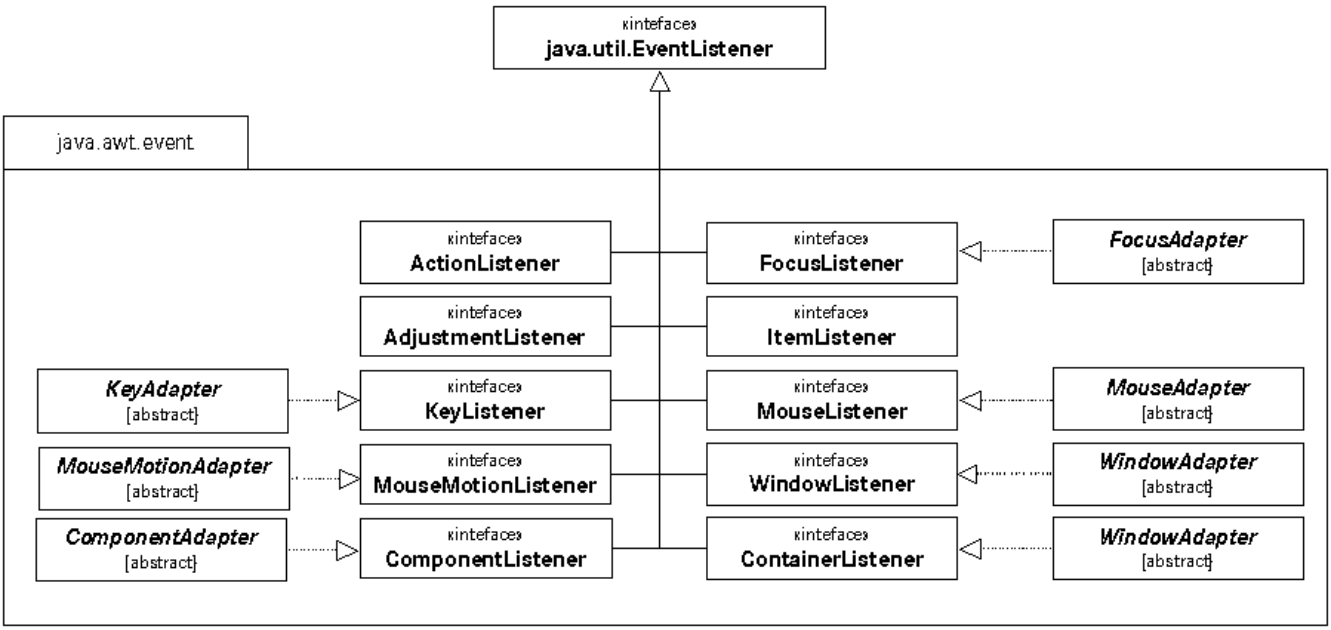
Im Allgemeinen erfolgt die Kommunikation zwischen einer graphischen Oberfläche und dem ausführenden Betriebsystem in Form von Nachrichten, die zwischen diesen beiden Kommunikationspartnern ausgetauscht werden. Das Betriebssystem unterrichtet die Applikation über den Eintritt bestimmter Ereignisse (engl. *event*), hierzu zählen u.a. Mausbewegungen und -klicks, Tastaturanschläge sowie alle Fensteroperationen.

Am Nachrichtenverkehr sind prinzipiell drei Objekte des Systems beteiligt: Die Ereignisquelle, das Objekt bei dem das Ereignis eintrat (z.B. im Falle eines Mausklicks: der entsprechende Button); der Ereignisempfänger, ein Objekt das auf das eingetretene Ereignis reagiert; die Nachricht selbst, in objektorientierten System als eigenständiges Objekt realisiert, welches das ausgelöste Ereignis näher beschreibt.
Dieser Kommunikationsmechanismus ist als *Delegation Based Event Handling* bekannt, da jedes Ereignis an eine konkrete Instanz delegiert wird, die nachfolgend die Behandlung übernimmt. Es bietet zwei entscheidende Vorteile:

- Verringerung des Nachrichtenverkehrs innerhalb der Applikation, durch klare Entscheidbarkeit des Nachrichtenweges. Insbesondere unterbleibt das Senden von Nachrichten, für die kein Empfänger existiert.
- Klare Trennung zwischen GUI-Code und Applikationslogik.



Die Abbilung zeigt eine Auswahl der verschiedenen Ereignistypen. Wie dargestellt sind alle spezifischen Ereignisse von der Klasse `EventObject` abgeleitet. Neben den AWT-Ereignissen existiert eine Fülle weiterer Ereignistypen, die in den verschiedenen spezifischen Paketen (wie `java.beans`, `javax.sound`, etc.) untergebracht sind.
Durch die Superklasse `EventObject` wird auch die an alle Unterklassen vererbte Methode `getSource()` definiert, welche zu jeder `EventObject`-Ausprägung die auslösende Instanz liefert.
Unterhalb der Klasse `ComponentEvent` sind die *low-level*-Ereignisse eingeordnet, die auf der Ebene einer visuellen Komponente aufteten können.
Die Ereignisse `ActionEvent`, `ItemEvent` und `TextEvent` werden als *semantische* Ereignisse bezeichnet, da sie nicht an ein konkretes visuelles Element gebunden sind.



Zur Verarbeitung der verschiedenen Nachrichtentypen muß der Empfänger eine Reihe von Methoden implementieren, die durch entsprechende Schnittstellen definiert sind. Die Abbildung stellt eine Auswahl der angebotenen Schnittstellen nebst den bereits angebotenen Implementierungen dar.

Schablone zur Reaktion auf AWT-Ereignisse

Die Reaktion auf AWT-Ereignisse wird durch sog. *event handler* übernommen; diese Objekte müssen die Benachrichtigung im Falle des Auftretens eines Ereignisses explizit anmelden.
Die Schritte im Einzelnen:

1. Definition einer Event-Handler-Klasse durch Implementierung der entsprechenden Schnittstelle
2. Registrierung des Event-Handlers

Somit kann auch das vorangegangene Beispiel entsprechend erweitert werden, um die noch ausstehende Reaktion auf das *Beenden*-Ereignis zu verwirklichen.

```
(1)import java.awt.Frame;
(2)import java.awt.Window;
(3)import java.awt.event.WindowAdapter;
(4)import java.awt.event.WindowEvent;
(5)
(6)public class AWTEx2 {
(7)     public static void main(String[] args) {
(8)             Frame wnd = new Frame("Einfaches Fenster");
(9)             wnd.addWindowListener( new WindowClosingAdapter() );
(10)            wnd.setSize(400,300);
(11)            wnd.setVisible(true);
(12)    } //main()
(13)} //class AWTEx2
(14)
(15)class WindowClosingAdapter extends WindowAdapter {
(16)    public void windowClosing( WindowEvent we) {
(17)            Window wnd = we.getWindow();
(18)
(19)            wnd.setVisible(false);
(20)            wnd.dispose();
(21)            System.exit(0);
(22)    } //windowClosing()
(23)} //class WindowClosingAdapter
```

Beispiel 90: Ein einfaches Fenster mit AWT AWTEx2.java

Die Applikation erweitert das bisherige Beispiel um eine Adapter-Klasse, welche auf Fensterereignisse reagiert. Hierzu wird eine Ausprägung der Adapter-Klasse `WindowClosingAdapter` als Window Listener mit der Methode `addWindowListener(WindowListener)` registriert.
Die Adapter-Klasse erweitert die bestehende API-Klasse `WindowAdapter` und implementiert damit die Schnittstelle `WindowListener`. Sie definiert die überschriebene Methode `windowClosing(WindowEvent)`, die bei auftreten des Ereignisses automatisch zur Verarbeitung aufgerufen wird.
Nach Ermittlung des Ereingis-sendenden Fensters per `getWindow()` wird dieses Fenster zunächst ausgeblendet (`setVisible(false)`) und im Anschluß zerstört (`dispose()`).
Danach terminiert die Applikation die virtuelle Maschine.

```
(1)import java.awt.Button;
(2)import java.awt.FlowLayout;
(3)import java.awt.Frame;
(4)import java.awt.Label;
(5)import java.awt.Window;
(6)import java.awt.event.MouseAdapter;
(7)import java.awt.event.MouseEvent;
(8)import java.awt.event.MouseEvent;
(9)import java.awt.event.MouseMotionAdapter;
(10)import java.awt.event.WindowAdapter;
(11)import java.awt.event.WindowEvent;
(12)
(13)public class AWTEx3 {
(14)    private Label mousePos,
(15)                                    counterVal;
(16)
(17)    public static void main(String[] args) {
(18)            AWTEx3 app = new AWTEx3();
(19)    } //main()
(20)
(21)    public AWTEx3() {
(22)            Frame wnd = new Frame("Einfaches Fenster");
(23)            mousePos = new Label("mouse pos");
(24)            counterVal = new Label("0");
(25)            Button counter = new Button("add one!");
(26)
(27)            wnd.setSize(200,100);
(28)            wnd.setLayout(new FlowLayout() );
(29)            wnd.add (mousePos);
(30)            wnd.add(counter);
(31)            wnd.add(counterVal);
(32)
(33)            wnd.addWindowListener( new WindowClosingAdapter() );
(34)            wnd.addMouseMotionListener( new MyMouseMotionAdapter(this) );
(35)            counter.addMouseListener( new CounterClicked(this) );
(36)            wnd.addMouseListener( new WindowClicked(this) );
(37)
(38)            wnd.setVisible(true);
(39)    } //constructor
(40)
(41)    public void setMousePos(String newText) {
```

```
(42)            mousePos.setText( newText );
(43)    } //setMousePos()
(44)
(45)    public void incrementCounter() {
(46)            counterVal.setText( ""+(Integer.parseInt(counterVal.getText())+1) );
(47)    } //incrementCounter()
(48)    public void decrementCounter() {
(49)            counterVal.setText( ""+(Integer.parseInt(counterVal.getText())-1) );
(50)    } //decrementCounter()
(51)} //class AWTEx3
(52)
(53)class WindowClosingAdapter extends WindowAdapter {
(54)    public void windowClosing(WindowEvent we) {
(55)            Window wnd = we.getWindow();
(56)
(57)            wnd.setVisible(false);
(58)            wnd.dispose();
(59)            System.exit(0);
(60)    } //windowClosing()
(61)} //class WindowClosingAdapter
(62)
(63)class CounterClicked extends MouseAdapter {
(64)    private AWTEx3 app;
(65)
(66)    public CounterClicked(AWTEx3 app) {
(67)            this.app = app;
(68)    } //constructor
(69)
(70)    public void mouseClicked( MouseEvent me) {
(71)            app.incrementCounter();
(72)    } //mouseClicked()
(73)} //class CounterClicked
(74)
(75)class WindowClicked extends MouseAdapter {
(76)    private AWTEx3 app;
(77)
(78)    public WindowClicked(AWTEx3 app) {
(79)            this.app = app;
(80)    } //constructor
(81)
(82)    public void mouseClicked( MouseEvent me) {
(83)            app.decrementCounter();
(84)    } //mouseClicked()
(85)} //class WindowClicked
(86)
(87)class MyMouseMotionAdapter extends MouseMotionAdapter {
(88)    private AWTEx3 app;
(89)
(90)    public MyMouseMotionAdapter(AWTEx3 app) {
(91)            this.app = app;
(92)    } //constructor
(93)
(94)    public void mouseMoved( MouseEvent me) {
(95)            app.setMousePos( "("+me.getX()+","+me.getY()+")" );
(96)    } //mouseMoved()
(97)} //class myMouseAdapter
```

Beispiel 91: Event-Handling AWTEx3.java

Bildschirmausgabe

Die Applikation definiert, unter Verwendung des `Flow`-Layouts, zwei Beschriftungsfelder (labels) `mousePos` und `counterVal` sowie eine mit `add one!` beschriftete Schaltfläche.
Anmerkung: Die Verwendung des Layoutmanagers ist zur Verarbeitung der Maus-Events zwingend notwendig!
Vor Anzeige des Applikationsfensters werden vier Ereignis-Handler registriert: Der bekannte `WindowClosingListener` zur Umsetzung des *Schließen*-Ereignisses, der `MouseMotionListener` zur Verfolgung der Mausbewegungen im Hautfenster sowie zwei `MouseListener` die auf Mausereignisse -- außer Bewegungen -- auf der definierten Schaltfläche und dem Haupfenster selbst reagieren.
Die Umsetzung des `WindowClosingListeners` ist gegenüber dem vorhergehenden Beispiel unverändert.
Innerhalb der Klassen `counterClicked` und `windowClicked` wird jeweils die Methode `mouseClicked` der Schnittstelle `MouseListener` überschrieben, sie wurde von der Superklasse `MouseAdapter` geerbt. Wird ein Mausclick auf den Button registriert, so wird eine Methode der Hautptklasse `awtEx3` aufgerufen, welche die Beschriftung des Textfeldes entsprechend verändert. Beim Mausklick ist es in der vorliegenden Implementierung unentscheident, mit welcher Maustaste dieser erfolgte. Eine nähere Analyse, beispielsweise hinsichtlich der gedrückten Taste, des aufgetretenen

Ereignisses kann über Attribute der Klasse `MouseEvent` erfolgen.
Die Spezialisierung der Klasse `MouseMotionAdapter` realisiert die Methode `mouseMoved(MouseEvent)`. Sie wird jeweils bei Änderung der Mauskoordinate aufgerufen. Die beiden Koordinaten werden innerhalb des Methodenkörpers mittels der angebotenen API-Methoden extrahiert und an die Hauptklasse zur Anzeige übergeben.

Event Handling und anonyme innere Klassen

Bei näherer Betrachtung der verschiedenen Event-Handler des vorhergehenden Beispiels fällt deren immergleiche Grundstruktur auf: Zunächst das Erben von einer passenden Adapterklasse um eine oder mehrere Methoden zu überschreiben. Im konkreten Beispiel wird sogar zweimal dieselbe Adapterklasse überschrieben; jedoch mit unterschiedlichem Verhalten.
Die durch die Spezialisierung entstehenden Klassen werden in allen Fällen zur Erzeugung genau eines Objekts -- des jeweiligen Listeners -- herangezogen; eine sonstige Weiter- oder Wiederverwendung in der Applikation liegt nicht vor, und ist unter Berücksichtigung des Klassendesigns auch für die Zukunft nicht anzunehmen.

Nimmt man die beiden Randbedingungen -- *Vererbung und Überschreiben ererbter Methoden* und *keine Wiederverwendung* -- zusammen, so offenbart sich das Sprachmittel der anonymen inneren Klassen als wesentlich adäquater zur Lösung der vorliegenden Problemstellung.

Unter Nutzung dieses Mechanismus ergibt läßt sich der Code des Beispiels wie folgt modifizieren:

```
(1)import java.awt.Button;
(2)import java.awt.FlowLayout;
(3)import java.awt.Frame;
(4)import java.awt.Label;
(5)import java.awt.Window;
(6)import java.awt.event.MouseAdapter;
(7)import java.awt.event.MouseEvent;
(8)import java.awt.event.MouseMotionAdapter;
(9)import java.awt.event.WindowAdapter;
(10)import java.awt.event.WindowEvent;
(11)
(12)public class AWTEx3i {
(13)    private Label mousePos,
(14)                                    counterVal;
(15)
(16)    private static AWTEx3i app;
(17)
(18)    public static void main(String[] args) {
(19)            app = new AWTEx3i();
(20)    } //main()
(21)
(22)    public AWTEx3i() {
(23)            Frame wnd = new Frame("Einfaches Fenster");
(24)            mousePos = new Label("mouse pos");
(25)            counterVal = new Label("0");
(26)            Button counter = new Button("add one!");
(27)
(28)            wnd.setSize(200,100);
(29)            wnd.setLayout(new FlowLayout() );
(30)            wnd.add (mousePos);
(31)            wnd.add(counter);
(32)            wnd.add(counterVal);
(33)
(34)            wnd.addWindowListener( new WindowAdapter() {
(35)                    public void windowClosing(WindowEvent we) {
(36)                            Window wnd = we.getWindow();
(37)
(38)                            wnd.setVisible(false);
(39)                            wnd.dispose();
(40)                            System.exit(0);
(41)                    } //windowClosing()
(42)            }//anonymous inner class
(43)            );
(44)
(45)            wnd.addMouseMotionListener( new MouseMotionAdapter() {
(46)                    public void mouseMoved( MouseEvent me) {
(47)                            app.setMousePos( "("+me.getX()+","+me.getY()+")" );
(48)                    } //mouseMoved()
(49)            } //anonymous inner class
(50)            );
(51)
(52)            counter.addMouseListener( new MouseAdapter() {
(53)                    public void mouseClicked( MouseEvent me)        {
(54)                            app.incrementCounter();
(55)                    } //mouseClicked()
(56)            } //anonymous inner class
(57)            );
(58)
(59)            wnd.addMouseListener( new MouseAdapter() {
```

```
(60)                    public void mouseClicked( MouseEvent me) {
(61)                            app.decrementCounter();
(62)                    } //mouseClicked()
(63)            } //anonymous inner class
(64)            );
(65)
(66)            wnd.setVisible(true);
(67)    } //constructor
(68)
(69)    public void setMousePos(String newText) {
(70)            mousePos.setText( newText );
(71)    } //setMousePos()
(72)
(73)    public void incrementCounter() {
(74)            counterVal.setText( ""+(Integer.parseInt(counterVal.getText())+1) );
(75)    } //incrementCounter()
(76)
(77)    public void decrementCounter() {
(78)            counterVal.setText( ""+(Integer.parseInt(counterVal.getText())-1) );
(79)    } //incrementCounter()
(80)} //class AWTEx3i
```

Beispiel 92: Das Beispiel AWTEx3 unter Nutzung anonymer innerer Klassen AWTEx3i.java

Der entstehende Code wird deutlich kompakter, und lokalisiert zusätzlich die Handlermethoden in den Registrierungsaufrufen.
Da anonyme innere Klassen (naturgemäß) keine Konstruktoren besitzen können wurden die Methoden entsprechend modifizert.

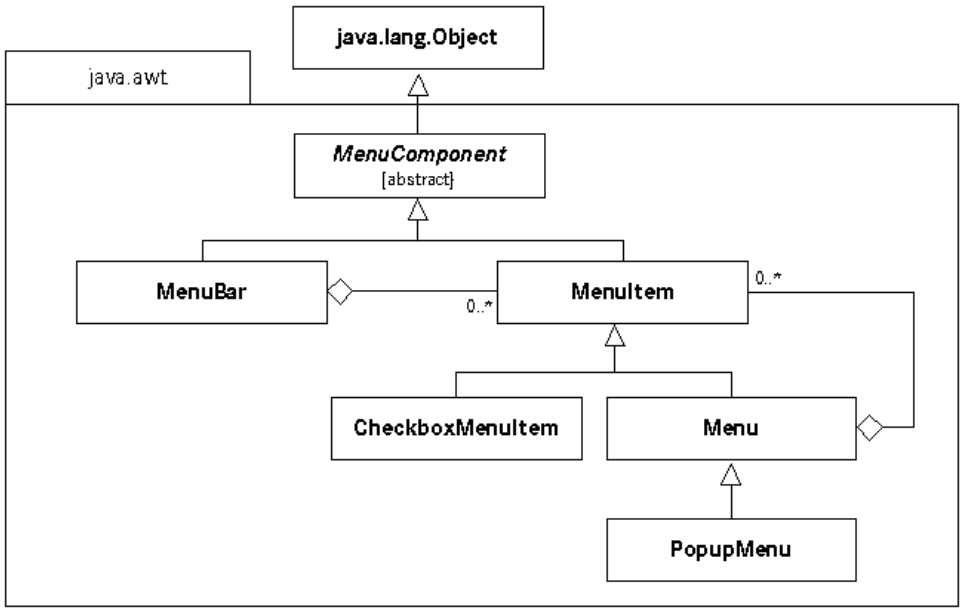
Event-Handling im Überblick

Schnittstelle	Methoden	Parameter/Zugriffsmethoden	Eventauslöser
ActionListener	actionPerformed(ActionEvent)	ActionEvent getActionCommand() getModifiers()	Button List MenuItem TextField
AdjustmentListener	adjustmentValueChanged(AdjustmentEvent)	AdjustmentEvent getAdjustable() getAdjustmentType() getValue()	Scrollbar
ItemListener	itemStateChanged(ItemEvent)	ItemEvent getItem() getItemSelectable() getStateChange()	Checkbox CheckboxMenuItem Choice List
TextListener	textValueChanged(TextEvent)	TextEvent	TextComponent
ComponentListener	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)	ComponentEvent getComponent()	Component
ContainerListener	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)	ContainerEvent getChild() getContainer()	Container
FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)	FocusEvent isTemporary()	Component
KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)	KeyEvent getKeyChar() getKeyCode() getKeyModifiersText(int) getKeyText() isActionKey()	Component
MouseListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)	MouseEvent getClickCount() getPoint() getX() getY() isPopupTrigger()	Component

MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)	MouseEvent getClickCount() getPoint() getX() getY() isPopupTrigger()	Component
WindowListener	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)	WindowEvent getWindow()	Window

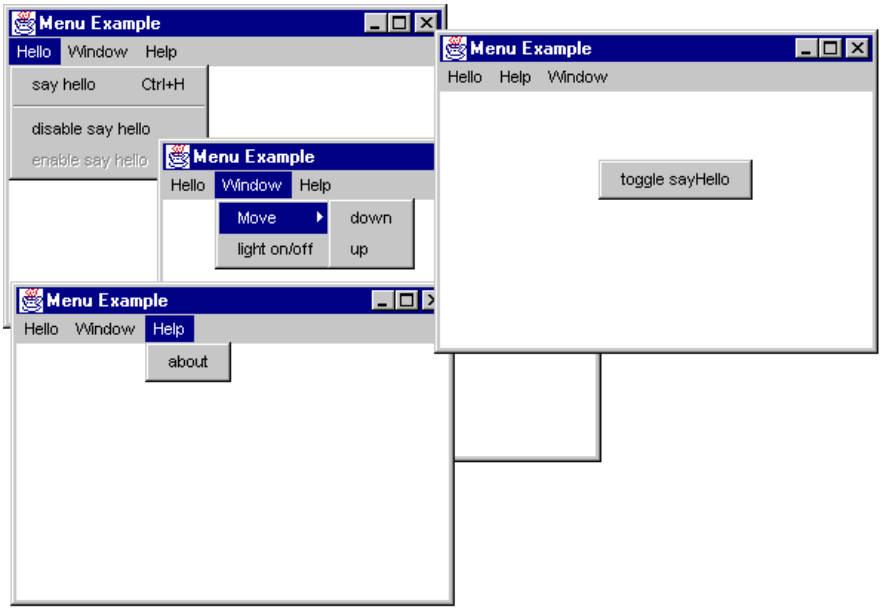
Menüs

Das AWT bietet die auch von anderen graphischen Benutzerschnittstellen bekannten Menüprimitive und -Grundfunktionen an. Darunter verschachtelete Menüs (Submenüs), Short-Cuts, frei plazierbare Kontextmenüs, etc. Die Abbildung zeigt die verschiedenen Primitive und ihre Beziehungen zueinander.

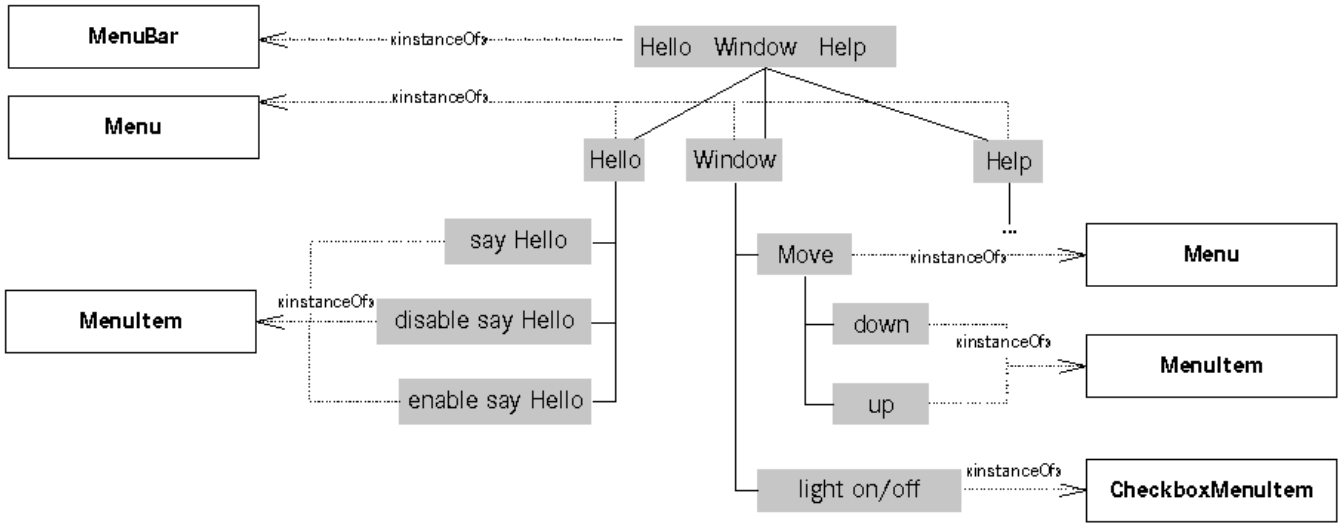


Die abstrakte Klasse MenuComponent bildet die Wurzel der Menüklassenhierarchie.
Ein MenuBar stellt eine Zusammenfassung von Menüs dar. Objekte diese Klasse bilden den Ausgangspunkt einer Menüstruktur.
Innerhalb eines MenuBars können beliebig viele MenuItem-Ausprägungen organisiert werden. Ein MenuItem ist ein beliebiger Eintrag eines Menüs. Hierbei kann es sich um einen anklickbaren Menüeintrag oder ein weiteres Menü handeln, welches sich bei Mausberührung öffnet.
Üblicherweise werden konkrete Menüeinträge in Menüs zusammengefaßt. Dem Menü selbst entspricht die AWT-Klasse Menu. Jedes Menu-Objekt kann aus weiteren MenuItems bestehen. Mithin kann ein Menü Submenüs oder direkte Menüeinträge in beliebiger Reihenfolge enthalten.
Als Spezialisierung der durch die Klasse MenuItem definierten Menüeinträge stehen umschaltbare Menüpunkte durch die Klasse CheckboxMenuItem zur Verfügung. Gegenüber den herkömmlichen Menüeinträgen verfügt dieser Typ über ein zusätzliches -- in der visuellen Darstellung plattformabhängig variierendes -- Symbol welches den Eintragszustand „aktiviert“ oder „deaktiviert“ anzeigt.
Menüleisten-unabhängige Kontextmenüs werden durch die Klasse PopupMenu realisiert. Sie sind frei plazierbar, verhalten sich jedoch ansonsten wie die bekannten Menüs.

Neue (klickbare) Menüeinträge werden als Objekte der Klasse MenuItem erzeugt. Die dem Konstruktor übergebene Zeichenkette legt den späteren angezeigten Eintrag im Menü fest. (Verwendung im Beispiel).



Die Abbildung stellt einen Ausschnitt der Menüstruktur der Beispielapplikation dar:



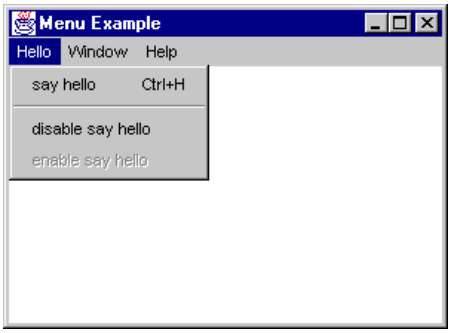
Erzeugung von Menüstrukturen

Ausgangspunkt der Menüerzeugung ist immer ein Objekt der Klasse `MenuBar`. Durch den parameterlosen Konstruktor wird eine leere Menüleiste erzeugt, die als Container zur Aufnahme der weiteren Menükomponenten dient. (Verwendung im Beispiel).
Die einzelnen Menüs werden durch Objekte der entsprechenden Klasse `Menu` repräsentiert. Der in der Menüleistete aufgeführte Menüname wird als Zeichenkettenparameter (Klasse `java.lang.String`) übergeben. (Verwendung im Beispiel.)
Durch die Methode `add(MenuItem)` werden Menüeinträge oder ganze Menüs -- allgemein: Subklassen von `MenuItem` -- in eine bestehende Menüleiste eingehängt. (Verwendung im Beispiel).
Zusätzlich kann dem Konstruktor eine Instanz der Klasse `MenuShortcut` übergeben werden. Hierdurch wird die in GUIs übliche direkte Anwahlmöglichkeit einzelner Menüeinträge über Tastenkürzel realisiert. (Verwendung im Beispiel).
Zur Erzeugung von `CheckboxMenuItems`, Menüeinträgen die ihren zweiwertigen Zustand automatisch darstellen, muß ein Objekt der Klasse `CheckboxMenuItem` erzeugt und in ein bestehendes Menü eingehängt werden. (Verwendung im Beispiel).
Die Variante der Pop-up Menüs wird durch die gleichnamige Klasse `PopupMenu` erzeugt. Im Unterschied zu den herkömmlichen Menüs wird dieser Menütyp nicht in der Menüleiste, sondern dem beherbergenden Fenster direkt, durch die dortige add-Methode, eingehängt. (Verwendung im Beispiel).

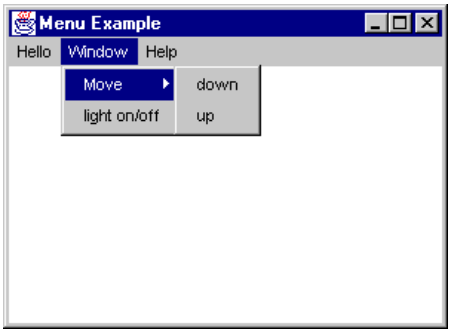
Die **Ereignisbehandlung** für Menüstrukturen folgt dem in der Übersicht *Event-Handling im Überblick* gegebenen Schema.
Für jeden Menüeintrag vom Typ `MenuItem` muß ein Objekt vom Typ `ActionListener` über die Methode `addActionListener(ActionListener)` der Klasse `MenuItem` registriert werden. Dieser muß die Methode `actionPerformed(ActionEvent)` überschreiben. Dort findet sich der Code, welcher bei Eintreten des Ereignisses (Mausclick auf Menüeintrag) abgearbeitet wird. (Verwendung im Beispiel).
Menüpunkte der Klasse `CheckboxMenüItem` werden durch Objekte die die Schnittstelle `ItemListener` implementieren überwacht. Tritt die Zustandsänderung ein, so wird die Methode `itemStateChanged(ItemEvent)` aufgerufen. (Verwendung im Beispiel).

```
(1)import java.awt.Button; (2)import java.awt.CheckboxMenuItem; (3)import java.awt.Color; (4)import java.awt.Dialog; (5)
import java.awt.FlowLayout; (6)import java.awt.Frame; (7)import java.awt.Label; (8)import java.awt.Menu; (9)import java.awt.
MenuBar; (10)import java.awt.MenuItem; (11)import java.awt.MenuShortcut; (12)import java.awt.Point; (13)import java.awt.
PopupMenu; (14)import java.awt.Window; (15)import java.awt.event.ActionEvent; (16)import java.awt.event.ActionListener; (17)
import java.awt.event.InputEvent; (18)import java.awt.event.ItemEvent; (19)import java.awt.event.ItemListener; (20)import
java.awt.event.KeyEvent; (21)import java.awt.event.MouseAdapter; (22)import java.awt.event.MouseEvent; (23)import java.awt.
event.WindowAdapter; (24)import java.awt.event.WindowEvent; (25) (26)public class AWTEx4 { (27) private static AWTEx4 app;
(28) private Label lbl_hello; (29) private Frame wnd; (30) private MenuItem mi_sayHello, mi_disable, mi_enable, mi_toggle,
mi_up, mi_down; (31) private CheckboxMenuItem mi_light; (32) private PopupMenu myPopupMenu; (33) (34) class
ToggleSayHelloState implements ActionListener { (35) public void actionPerformed(ActionEvent ae) { (36) mi_sayHello.
setEnabled( !mi_sayHello.isEnabled() ); (37) mi_enable.setEnabled( !mi_sayHello.isEnabled() ); (38) mi_disable.setEnabled
( mi_sayHello.isEnabled() ); (39) } //actionPerformed() (40) } //class ToggleSayHelloSate (41) (42) public static void main
(String[] args) { (43) app = new AWTEx4(); (44) } //main() (45) (46) public AWTEx4() { (47) wnd = new Frame("Menu
Example"); (48) lbl_hello = new Label("Hello!"); (49) wnd.add(lbl_hello); (50) (51) wnd.setSize(300,200); (52) wnd.setLayout
(new FlowLayout() ); (53) (54) MenuBar myMenuBar = new MenuBar(); (55) Menu m_hello = new Menu("Hello"); (56) myMenuBar.add
(m_hello); (57) mi_sayHello = new MenuItem("say hello", new MenuShortcut (KeyEvent.VK_H)); (58) m_hello.add( mi_sayHello );
(59) mi_disable = new MenuItem("disable say hello"); (60) m_hello.addSeparator(); (61) mi_enable = new MenuItem("enable say
hello"); (62) m_hello.add( mi_disable ); (63) m_hello.add( mi_enable ); (64) (65) Menu m_windowAction = new Menu("Window");
(66) Menu m_windowActionMove = new Menu("Move"); (67) mi_light = new CheckboxMenuItem("light on/off"); (68) mi_down = new
MenuItem("down"); (69) mi_up = new MenuItem ("up"); (70) (71) m_windowActionMove.add(mi_down); (72) m_windowActionMove.add
(mi_up); (73) m_windowAction.add(m_windowActionMove); (74) m_windowAction.add(mi_light); (75) myMenuBar.add
(m_windowAction); (76) (77) //pop-up menu (78) myPopupMenu = new PopupMenu(); (79) MenuItem mi_toggle = new MenuItem
("toggle sayHello"); (80) myPopupMenu.add( mi_toggle ); (81) wnd.add( myPopupMenu ); (82) (83) Menu m_hlp = new Menu
("Help"); (84) m_hlp.add(new MenuItem("about")); (85) (86) myMenuBar.setHelpMenu(m_hlp); (87) (88) wnd.setMenuBar
(myMenuBar); (89) (90) mi_up.addActionListener( new ActionListener() { (91) public void actionPerformed(ActionEvent ae)
{ (92) Point pnt = wnd.getLocation(); (93) (94) for (int x=pnt.x; x >= 0 ; x--) (95) wnd.setLocation(x, pnt.y); (96) for
(int y=pnt.y; y >= 0; y--) (97) wnd.setLocation(0, y); (98) } //actionPerformed() (99) } //anonymous inner class (100) );
(101) (102) mi_down.addActionListener( new ActionListener() { (103) public void actionPerformed(ActionEvent ae) { (104)
final Dialog dlg_notImplemented = new Dialog (wnd,true); (105) Label lbl = new Label("function not implemented, yet!");
(106) dlg_notImplemented.setTitle("Sorry, I'm afraid I can't to that ..."); (107) dlg_notImplemented.setLayout( new
FlowLayout() ); (108) dlg_notImplemented.setResizable(false); (109) dlg_notImplemented.add(lbl); (110) dlg_notImplemented.
setSize(250,80); (111) (112) Button btn_ok = new Button("ok"); (113) btn_ok.addActionListener( new ActionListener() { (114)
public void actionPerformed(ActionEvent ae) { (115) dlg_notImplemented.setVisible(false); (116) dlg_notImplemented.dispose
(); (117) } //actionPerformed() (118) } //anonymous inner class (119) ); (120) (121) dlg_notImplemented.add(btn_ok); (122)
(123) dlg_notImplemented.show(); (124) } //actionPerformed() (125) } //anonymous inner class (126) ); (127) (128) mi_light.
addItemListener( new ItemListener() { (129) public void itemStateChanged(ItemEvent ie) { (130) if (mi_light.getState())
(131) wnd.setBackground( new Color(33,33,33) ); (132) else (133) wnd.setBackground( new Color(255,255,255) ); (134) } //
actionPerformed() (135) } //anonymous inner class (136) ); (137) (138) mi_sayHello.addActionListener( new ActionListener()
{ (139) public void actionPerformed(ActionEvent ae) { (140) lbl_hello.setVisible(true); (141) try { (142) Thread.sleep
(1500); (143) } catch (InterruptedException e) { (144) //ignore it (145) } //catch (146) lbl_hello.setVisible(false);
(147) } //actionPerformed() (148) } //anonymous inner class (149) ); (150) (151) ActionListener toggler = new
ToggleSayHelloState(); (152) mi_disable.addActionListener( toggler ); (153) mi_enable.addActionListener( toggler ); (154)
mi_toggle.addActionListener( toggler ); (155) (156) wnd.addWindowListener( new WindowAdapter() { (157) public void
windowClosing(WindowEvent we) { (158) Window wnd = we.getWindow(); (159) (160) wnd.setVisible(false); (161) wnd.dispose();
(162) System.exit(0); (163) } //windowClosing() (164) }//anonymous inner class (165) ); (166) (167) wnd.addMouseListener
( new MouseAdapter() { (168) public void mouseClicked( MouseEvent me) { (169) if ( (me.getModifiers() & InputEvent.
BUTTON1_MASK) == 0 ) (170) myPopupMenu.show( me.getComponent(), me.getX(), me.getY() ); (171) } //mouseClicked() (172) } //
anonymous inner class (173) ); (174) (175) wnd.setVisible(true); (176) lbl_hello.setVisible(false); (177) mi_enable.
setEnabled(false); (178) } //main() (179)} //class AWTEx4
```

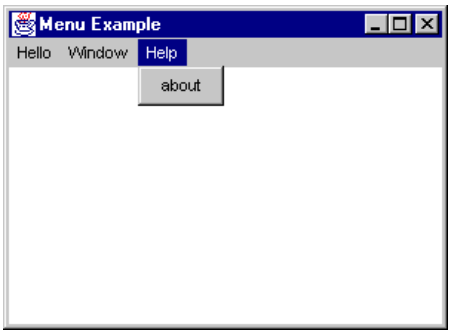
Beispiel 93: Beispiel einer Menüstruktur AWTEx4.java



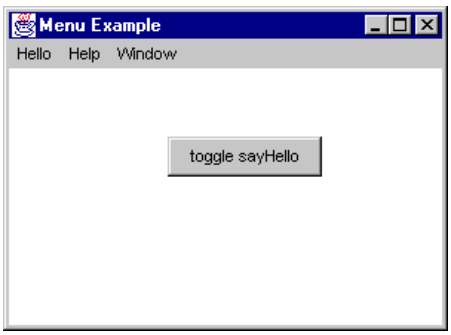
Das Menü *Hello* der Beispielapplikation besteht aus drei Menüeinträgen -- *say hello*, *enable say hello* und *disable say hello*. Für den ersten Menüpunkt ist das Tastaturkürzel `CTRL+H` definiert (Definition im Code). Bereits zum Startzeitpunkt der Applikation ist der Menüpunkt *enable say hello* deaktiviert, da der entsprechende Menüpunkt vorgabegemäß aktiviert ist. Die Instanzenmethode `setEnabled` der Klasse `MenuItem` erlaubt den Wechsel zwischen den beiden Zuständen „aktiviert“, mit normaler visueller Darstellung des Menüpunktes, und „deaktiviert“, mit entsprechender *aus-gegrauter* Darstellung. Standardmäßig sind alle Menüeintrage nach ihrer Ereugung aktiviert, daher muß die gewünschte Deaktivierung explizit erfolgen. (Stelle im Code).
Wird der Menüpunkt *disable say hello* angewählt, so zieht dies die Deaktivierung des Menüpunktes *say hello* und gleichzeitige (Re-)Aktivierung von *enable say hello* nach sich.
Die Selektierung des Menüpunktes *enable say hello* vollführt hingegen die duale Operation dazu, Aktivierung der Menüpunkte *say hello* und *disable say hello*. Aus diesem Grunde teilen sich beide Menüpunkte dieselbe Ereignisbehandlungs-Routine. Als Konsequenz dieser Forderung wird die Initialisierung der beiden `ActionListener` mit demselben Objekt notwendig (Codestelle)-- daher kann die Umsetzung an dieser Stelle nicht mehr als anonyme innere Klasse realisiert werden, sondern erfolgt als „normale“ innere Klasse (Codestelle).
Die Anwahl des Menüpunktes *say hello* zeigt für eineinhalb Sekunden den Schriftzug *Hello!* am Bildschirm an. (Ereignisbehandlungsroutine).



Das zweite Menü der Menüleiste, mit *Window* betitelt, enthält ein Untermenü *Move* und eine Ausprägung von `CheckboxMenuItem`.
Die Ereignisbehandlung für Submenüeinträge erfolgt analog der auf höheren Ebenen; durch Definition des entsprechenden `ActionListener`-Instanz. (Implementierung der Ereignisbehandlugn des Menüpunktes *up*). Bei Auswahl dieses Menüpunkts wird das Fenster, durch mehrmalige Änderung der Bildschirmkoordinaten in die linke obere Bildschirmecke verschoben.
Der Eintragstyp `CheckboxMenuItem` erfordert eine Behandlungsroutine vom Typ `ItemListener`. Die Anzeige des Zustands wird durch die AWT automatisch vorgenommen, und bedarf keiner Anwenderinteraktion. (Behandlungsroutine des Menüeintrages *light on/off*). Der Menüpunkt ändert den Bildschirmhintergrund je nach Zustand von hell nach dunkel oder umgekehrt.



Das dritte dargestellte Menü wurde als Standard-Hilfe-Menü deklariert. Existiert bereits ein Menü dieser Eigenschaft, so wird es durch das neu definierte ersetzt.
Die Aufnahme in die Menüleiste erfolgt nicht durch die bekannte add-Methode, sondern über den separaten Aufruf `setHelpMenu(Menu)` (Codestelle).



Zusätzlich definiert die Applikation ein Kontextmenü welches die Funktionalität der Menüeinträge *enable say hello* und *disable say hello* vereinigt. Auch es nutzt die Ereignisbehandlungsroutine der beiden genannten Menüeinträge mit. (Codestelle).
Die Aktivierung und Anzeige am Punkt der Aktivierung muß durch den Anwender selbst realisiert werden. Hierfür ist ein enstprechender `MouseListener` umzusetzen. Die Methode `show(Component, int, int)` erlaubt hierfür die Übergabe von Koordinaten, an denen das Kontextmenü aufgeklappt wird.

Textfelder

Editierbare Textfelder für Benutzereingaben stellt die Klasse `TextField` bereit.

Zur Reaktion auf Tastenanschläge kann -- wie in Übersicht dargestellt -- jedes Objekt genutzt werden, das die von `Component` ererbte Schnittstelle `KeyListener` implementiert.

```
(1)import java.awt.Frame; (2)import java.awt.GridLayout; (3)import java.awt.Label; (4)import java.awt.TextField; (5)import
java.awt.Window; (6)import java.awt.event.KeyAdapter; (7)import java.awt.event.KeyEvent; (8)import java.awt.event.
WindowAdapter; (9)import java.awt.event.WindowEvent; (10) (11)public class AWTEx5 { (12) private static AWTEx5 app; (13)
private TextField tf_dm, tf_eur; (14) (15) public static void main(String[] args) { (16) app = new AWTEx5(); (17) } //main
() (18) (19) public AWTEx5() { (20) Frame wnd = new Frame("Euro-Konverter"); (21) (22) wnd.setSize(200,100); (23) wnd.
setLayout(new GridLayout(2,2) ); (24) (25) Label lbl_dm = new Label ("DM"); (26) Label lbl_eur = new Label ("Euro"); (27)
tf_dm = new TextField("1", 7); (28) tf_eur = new TextField("1.95583", 7 ); (29) (30) wnd.add(lbl_dm); (31) wnd.add
(lbl_eur); (32) wnd.add(tf_dm); (33) wnd.add(tf_eur); (34) wnd.show(); (35) (36) (37) tf_dm.addKeyListener( new KeyAdapter
() { (38) public void keyReleased(KeyEvent e) { (39) tf_eur.setText( ""+Double.parseDouble(tf_dm.getText())*1.95583 );
(40) } //keyPressed() (41) } //anonymous inner class (42) ); (43) (44) tf_eur.addKeyListener( new KeyAdapter() { (45)
public void keyReleased(KeyEvent e) { (46) tf_dm.setText(""+Double.parseDouble(tf_eur.getText())/1.95583 ); (47) } //
keyPressed() (48) } //anonymous inner class (49) ); (50) (51) wnd.addWindowListener( new WindowAdapter() { (52) public void
windowClosing(WindowEvent we) { (53) Window wnd = we.getWindow(); (54) (55) wnd.setVisible(false); (56) wnd.dispose(); (57)
System.exit(0); (58) } //windowClosing() (59) }//anonymous inner class (60) ); (61) } //constructor (62)} //end class AWTEx5
```

Beispiel 94: Ein einfacher Euro-Umrechner AWTEx5.java

Bildschirmausgabe:

Die Applikation nutzt das `GridLayout` zur Ausrichtung der vier visuellen Komponenten. Hierfür wird der entsprechende Layoutmanager dahingehend parametrisiert, daß er ein quadratisches Layout mit jeweils zwei horizontalen und vertikalen Einträgen erlaubt. (Codestelle).
Die Beschriftungen der Spalten sind als `Labels` realisiert.
Zum Erzeugungszeitpunkt wird den Text-Feldern ein Startwert, und die horizontale Ausdehnung von sieben Zeichen zugewiesen. (Codestelle).
In den (anonymen inneren) Klassen zur Ereignisbehandlung, wird die Methode `keyTyped(KeyEvent)` überschrieben. Sie wird *nach* Abschluß des Tastendrucks aufgerufen.
In der Implementierung dieser Methode wird die tatsächliche Umrechung vorgenommen, und daß Umrechnungsergebnis im jeweils anderen Feld ausgegeben. (Codestelle).
Hinweis: Eine Behandlung des möglicherweise generierten `NumberFormatException`-Ausnahmeereignisses findet aus Übersichtlichkeitsgründen nicht statt.

Dialoge

Ausprägungen der Klasse `Dialog` stellen eine Möglichkeit zur Realisierung einfacher Ein- und Ausgabefenster dar. Diese Art Fenster eignet sich besonders für kurze Anfragen, oder Meldungen, an den Anwender.

Beispiel *awtEx4* enthält eine solche Nachrichtenbox. Dialogfenster können, im Gegensatz zu den bekannten Applikationsfenstern, *modal* sein. Dies bedeutet, daß ein solches Fenster nicht durch den Anwender in den Hintergrund versetzt werden kann; es erzwingt eine Reaktion -- zumeist in Form der Bestätigung einer Meldung o.ä. -- bevor die Applikation fortfährt.

3.2.8 Swing

Mit dem JDK v1.1 wurde als zusätzliche API zur Erstellung von graphischen Oberflächen `Swing` vorgestellt. Bis zur aktuellen Version 1.3 ist diese API weiterhin Bestandteil des Extension Paketes, und steht daher nicht auf allen Plattformen zur Verfügung. Des weiteren können sich noch zukünftige Änderungen an den derzeit publizierten APIs ergeben, so daß bestehender Code überarbeitet werden muß.

Folgende Überlegungen führten zur Entwicklung von Swing:

- Durch die Übernahme des plattformspezifischen look-and-feels der AWT-Applikationen entsteht teilweise großer Portierungsaufwand zur Realisierung nicht-nativ verfügbarer Primitive.
- Jede Änderung an den realen GUIs der Zielsysteme läßt Modifikationen an der AWT-Implementierung notwendig werden.
  Dies konterkariert die angestrebte Plattformunabhängigkeit auf der Ebene der angebotenen GUIs zunehmend.
- Aufwendige und komplexe Benutzeroberflächen sind mit dem AWT u. U. sehr schwer zu realisieren.

Daher wurde mit den *Java Foundation Classes*, deren Bestandteil Swing ist, versucht eine echte Alternative zum bestehenden -- und weiter angebotenen -- AWT zu schaffen. Dabei erhöht Swing massiv die Anzahl der angebotenen visuellen Komponenten, und fügt einige sehr mächtige Primitive wie beispielsweise zur Darstellung von Bäumen hinzu.

Anders als die AWT setzt Swing nicht auf den Möglichkeiten des zugrundeliegenden GUI-Systems auf, sondern stellt selbst die gesamte Verwaltung und Verarbeitung der angebotenen Primitven zur Verfügung. Als Folge dieses Ansatzes realisiert Swing ein eigenes look-and-feel, das auf allen unterstützten Plattformen unverändert präsentiert wird. Mit diesem -- *Metal* genannten -- Layout entsteht erstmals ein typisches Aussehen nativer Java-Applikationen.
Technisch ist Swing mit den seit JDK v1.1 angebotenen *leichtgewichtigen Komponenten*, den Java Beans, realisiert.
Die Realisierung graphischer Primitivoperationen erfolgt daher nicht mehr durch Operationen des zugrundeliegenden

GUI-Systems, sondern durch die angebotenen Swing-Komponenten selbst.
Dies führt dazu, daß die gesamte Swing-API ohne native Methoden -- vollständig in Java codiert -- realisiert werden konnte. Hieraus ergeben sich weitere Vorteile in der Anwendung, insbesondere in der Fehlersuche.

Jedoch führt die verfolgte *pure Java*-Implementierung auch zu Problemen. So erhöht sich durch den de-facto emulierenden Ansatz der Speicherplatzbedarf, da nicht mehr auf die evtl. durch die Plattform angebotenen Routinen zurückgegriffen wird. Flankierend sinkt die Ausführungsgeschwindigkeit durch die gestiegene Menge an auszuführendem Java-Code.
Anmerkung:
Aus rein praktischen Erwängungen sprechen derzeit noch zwei weitere Punkte gegen den breiten Swing-Einsatz: zunächst der Reifegrad und die Entwicklungsstabilität. Als Bestandteil des Java Extension Paketes ist Swing explizit als experimenteller Bestandteil der derzeit verfügbaren Java-API gekennzeichnet. In der verbreiteten Implementierung finden sich noch kleinere und größere Fehler und Ungereimtheiten, die gegen die Verwendung in Produktivappliaktaionen sprechen. Wie bei allen anderen Bestandteilen des *javax*-Paketes auch, behält sich SUN explizit das Recht vor Schnittstellen und Funktionalität ohne Ankündigung zu verändern oder aus dem Angebot herauszunehmen. Daher verbietet sich, unter Berücksichtigung einer zukünftigen Pflege des entstehenden Applikationscodes, die Nutzung dieser Klassen schon fast.
Ergänzend sei noch erwähnt, daß seitens der aktuell verfügbaren Web-Browser noch keine Swing-Unterstützung umgesetzt ist. Dieses Manko läßt sich zwar durch die zusätzliche manuelle Installation des *Java-Plugin*s beheben, verlangt dem Anwender jedoch einen zusätzlichen Installationsvorgang ab.

Swing verwirklicht durchgängig das von T. Renskaug initiierte *Model-View-Controller*-Konzept. Dessen hervorstechendstes Kennzeichen die Separierung des Codes in drei verschiedene Verwendungskategorien ist:

- **Model** -- Enthält die Daten und den Zustand des Dialogelements
- **View** -- Übernimmt die graphische Darstellung der visuellen Komponente
- **Controller** -- Verbindet *Model* und *View*; empfängt Ereignisse und leitet entsprechende Maßnahmen an Model und View ein

Die gesamte Verarbeitungslogik ist dabei in der *Model*-Klasse zentralisiert. Zu jedem *Model* können dabei gleichzeitig verschiedene *View*-Klassen existieren.

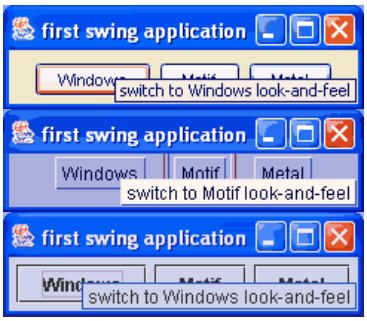
Swing reduziert diesen Mechanismus auf zwei verschiedenen Klassen-Typen. Hierbei wird die *View*- und die *Controller*-Komponente zu einer Einheit verschmolzen. Das entstehende Design wird *Model-Delegate-Prinzip* genannt.

```
(1)import java.awt.Window;
(2)import java.awt.event.ActionEvent;
(3)import java.awt.event.ActionListener;
(4)import java.awt.event.WindowAdapter;
(5)import java.awt.event.WindowEvent;
(6)import javax.swing.JButton;
(7)import javax.swing.JFrame;
(8)import javax.swing.JPanel;
(9)import javax.swing.SwingUtilities;
(10)import javax.swing.UIManager;
(11)import javax.swing.UnsupportedLookAndFeelException;
(12)
(13)public class SwingEx1 extends JFrame implements ActionListener {
(14)    public SwingEx1() {
(15)            super("first swing application");
(16)            JPanel myPanel = new JPanel();
(17)            JButton btn_win = new JButton("Windows");
(18)            JButton btn_motif = new JButton("Motif");
(19)            JButton btn_metal = new JButton("Metal");
(20)
(21)            btn_win.setToolTipText("switch to Windows look-and-feel");
(22)            btn_motif.setToolTipText("switch to Motif look-and-feel");
(23)            btn_metal.setToolTipText("switch to Metal look-and-feel");
(24)
(25)            myPanel.add( btn_win );
(26)            myPanel.add( btn_motif );
(27)            myPanel.add( btn_metal );
(28)
(29)            btn_win.addActionListener(this);
(30)            btn_motif.addActionListener(this);
(31)            btn_metal.addActionListener(this);
(32)
(33)            getContentPane().add("South", myPanel);
(34)
(35)            addWindowListener( new WindowAdapter() {
(36)                    public void windowClosing(WindowEvent we) {
(37)                            Window wnd = we.getWindow();
(38)
(39)                            wnd.setVisible(false);
(40)                            wnd.dispose();
(41)                            System.exit(0);
(42)                    } //windowClosing()
(43)            }//anonymous inner class
(44)            );
(45)    } //constructor
(46)
```

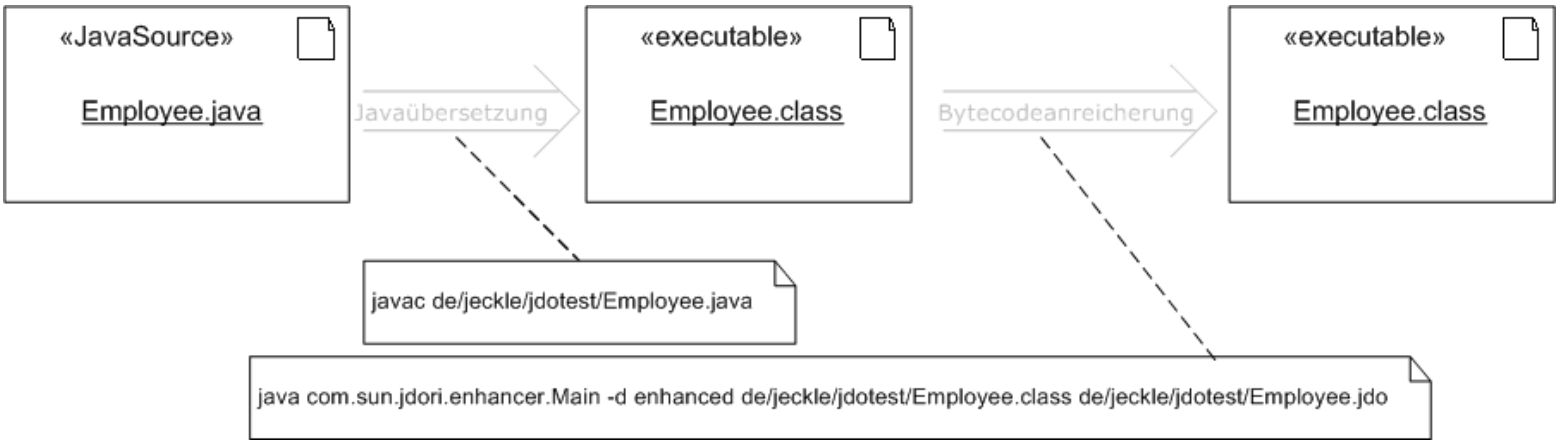
```
(47)    public void actionPerformed(ActionEvent ae) {
(48)            String cmd = ae.getActionCommand(),
(49)                            plaf="";
(50)
(51)            if ( cmd.equals("Metal") ) {
(52)                    plaf = "javax.swing.plaf.metal.MetalLookAndFeel";
(53)            } else {
(54)                    if ( cmd.equals("Motif") ) {
(55)                            plaf = "com.sun.java.swing.plaf.motif.MotifLookAndFeel";
(56)                    } else {
(57)                            if( cmd.equals("Windows") ) {
(58)                                    plaf = "com.sun.java.swing.plaf.windows.WindowsLookAndFeel";
(59)                            } //else
(60)                    } //else
(61)            } //else
(62)
(63)            try {
(64)                    UIManager.setLookAndFeel(plaf);
(65)                    SwingUtilities.updateComponentTreeUI(this);
(66)            } catch (UnsupportedLookAndFeelException e) {
(67)                    System.err.println(e.toString());
(68)    } catch (ClassNotFoundException e) {
(69)                    System.err.println(e.toString());
(70)            } catch (InstantiationException e) {
(71)                    System.err.println(e.toString());
(72)            } catch (IllegalAccessException e) {
(73)                    System.err.println(e.toString());
(74)            } //catch
(75)    } //actionPerformed()
(76)
(77)    public static void main(String[] args) {
(78)            SwingEx1 wnd = new SwingEx1();
(79)            wnd.setSize(300,200);
(80)            wnd.pack();
(81)            wnd.setVisible(true);
(82)    } //end main()
(83)} //class SwingEx1
```

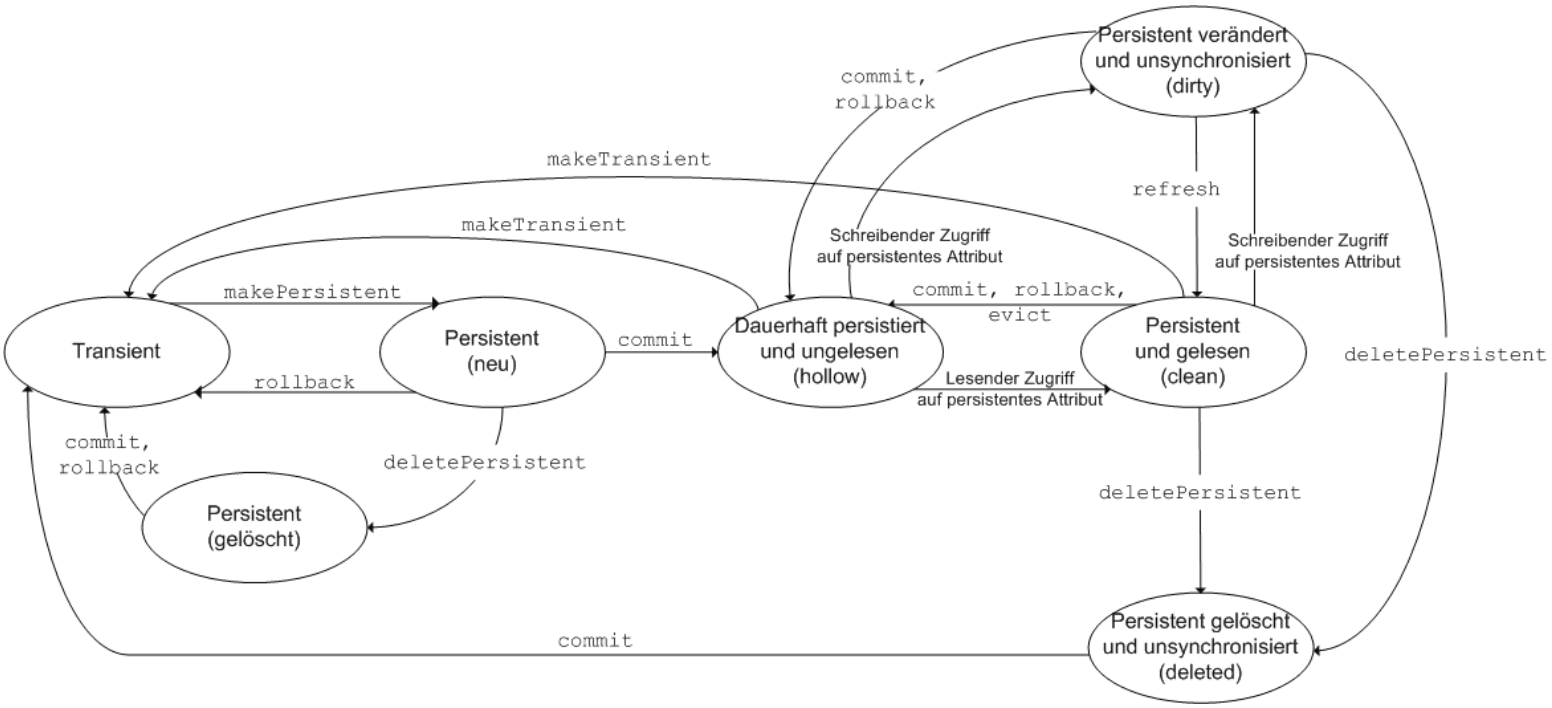
Beispiel 95: Eine erstes Swing-Applikation SwingEx1.java

Bildschirmausgabe:



Diese Swing-Applikation definiert ein einfaches Fenster mit drei Buttons.
Deutlichstes Kennzeichen der Nutzung von Swing ist das Präfix *J* vor den bekannten AWT-Komponentennamen. So erweitert die Applikation das Swing-Analogon der AWT-Klasse `Frame`, jetzt unter dem Namen `JFrame`.
Die Schaltflächen sind entsprechend Ausprägungen von `JButton`. Mit der Methode `setToolTipText(String)` wird die rein Swing-interne Möglichkeit zur Definition eines Hilfstexts genutzt, der bei Überstreichen einer Komponente mit der Maus automatisch angezeigt wird. Die Implementierung greift nicht auf Mechanismen des zugrundeliegenden GUI-Systems zurück, sondern ist vollständig innerhalb von Swing realisiert.
Die Ereingisbearbeitung ist identisch zur AWT v1.1 realisiert, und kann daher naherzu unverändert übernommen werden.
Als Behandlungsroutine der drei Schaltflächen ist der Wechsel des Bildschirmlayouts zur Laufzeit realisiert. Nach Aktivierung der entsprechenden Schaltfläche wird das entsprechende, innerhalb der Swing-API vordefinierte Layouts referenziert, und durch die statische Methode `setLookAndFeel(String)` geladen. Nach der Aktualisierung der Bildschirmdarstellung präsentiert sich die gesamte Applikation im neuen Layout.


«JavaSource»
Employee.java
Javaübersetzung
«executable»
Employee.class
Bytecodeanreicherung
«executable»
Employee.class
javac de/jeckle/jdotest/Employee.java
java com.sun.jdori.enhancer.Main -d enhanced de/jeckle/jdotest/Employee.class de/jeckle/jdotest/Employee.jdo


Persistent verändert
und unsynchronisiert
(dirty)
commit,
rollback
makeTransient
refresh
makeTransient
Schreibender Zugriff
auf persistentes Attribut
Schreibender Zugriff
auf persistentes Attribut
makePersistent
commit, rollback,
evict
Transient
Persistent
(neu)
commit
Dauerhaft persistiert
und ungelesen
(hollow)
Persistent
und gelesen
(clean)
deletePersistent
rollback
Lesender Zugriff
auf persistentes Attribut
commit,
rollback
deletePersistent
deletePersistent
Persistent
(gelöscht)
Persistent gelöscht
und unsynchronisiert
(deleted)
commit

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;
import java.util.Vector;

import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;

import de.jeckle.jdotest.Employee;

public class JDOStoreObj {
	public static void main(String args[]) {
		Properties props = new Properties();
		try {
			InputStream is =
				ClassLoader.getSystemResourceAsStream("jdo.properties");
			props.load(is);
		} catch (IOException ioe) {
			System.out.println("Error loading properties");
			System.exit(1);
		}
		PersistenceManagerFactory pmf =
			JDOHelper.getPersistenceManagerFactory(props);
		PersistenceManager pm = pmf.getPersistenceManager();

		Vector empCol = new Vector();

		Employee emp1 = new Employee();
		emp1.setName("Mario Jeckle");
		emp1.setDepartment("D001");
		emp1.addProject("P001");
		emp1.addProject("P002");
		empCol.add(emp1);

		Employee emp2 = new Employee();
		emp2.setName("John DoeX");
		emp2.setDepartment("D003");
		emp2.addProject("P001");
		emp2.addProject("P042");
		empCol.add(emp2);

		Employee emp3 = new Employee();
		emp3.setName("John Doe");
		emp3.setDepartment("B042");
```

```
            empCol.add(emp3);

            Employee emp4 = new Employee();
            emp4.setName("Barnie Bar");
            emp4.setDepartment("B042");

            pm.currentTransaction().begin();
            pm.makePersistentAll(empCol);
            pm.makePersistent(emp4);

            pm.currentTransaction().commit();
            pm.close();
        }
}
```

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Iterator;
import java.util.Properties;

import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;

import de.jeckle.jdotest.Employee;

public class JDORollback {

	public static void main(String args[]) {
		Properties props = new Properties();
		try {
			InputStream is =
				ClassLoader.getSystemResourceAsStream("jdo.properties");
			props.load(is);
		} catch (IOException ioe) {
			System.out.println("Error loading properties");
			System.exit(1);
		}
		PersistenceManagerFactory pmf =
			JDOHelper.getPersistenceManagerFactory(props);
		PersistenceManager pm = pmf.getPersistenceManager();

		Employee e = new Employee();
		e.setName("Marta Mayer");

		pm.currentTransaction().begin();
		pm.makePersistent(e);
		pm.currentTransaction().commit();
		displayPersistedObjects(pm);

		System.out.println("Martha gets married and changes her name");

		pm.currentTransaction().begin();
		e.setName("Marta Smith");
		pm.makePersistent(e);
		displayPersistedObjects(pm);
		System.out.println("Suppose and error happens now ...\nRolling back");
		pm.currentTransaction().rollback();

		displayPersistedObjects(pm);
```

```
	}
	private static void displayPersistedObjects(PersistenceManager pm) {
		Iterator i = pm.getExtent(Employee.class, false).iterator();
		while (i.hasNext()) {
			System.out.println((Employee) i.next());
		}
	}
}
```

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Iterator;
import java.util.Properties;

import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;

import de.jeckle.jdotest.Employee;
public class JDONonTransact {
	public static void main(String args[]) {
		Properties props = new Properties();
		try {
			InputStream is =
				ClassLoader.getSystemResourceAsStream("jdo.properties");
			props.load(is);
		} catch (IOException ioe) {
			System.out.println("Error loading properties");
			System.exit(1);
		}
		PersistenceManagerFactory pmf =
			JDOHelper.getPersistenceManagerFactory(props);
		PersistenceManager pm = pmf.getPersistenceManager();

		Employee e = new Employee();
		e.setName("Marta Mayer");
		pm.currentTransaction().setNontransactionalWrite(true);

		pm.currentTransaction().begin();
		pm.makePersistent(e);
		pm.currentTransaction().commit();
		displayPersistedObjects(pm);

		//martha gets married and changes her name

		e.setName("Marta Smith");

		displayPersistedObjects(pm);
	}
	private static void displayPersistedObjects(PersistenceManager pm) {
		Iterator i = pm.getExtent(Employee.class, false).iterator();
		while (i.hasNext()) {
			System.out.println((Employee) i.next());
		}
```

```
        }
}
```

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Iterator;
import java.util.Properties;

import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;

import de.jeckle.jdotest.Employee;

public class JDOListObj {
	public static void main(String args[]) {
		Properties props = new Properties();
		try {
			InputStream is =
				ClassLoader.getSystemResourceAsStream("jdo.properties");
			props.load(is);
		} catch (IOException ioe) {
			System.out.println("Error loading properties");
			System.exit(1);
		}
		PersistenceManagerFactory pmf =
			JDOHelper.getPersistenceManagerFactory(props);
		PersistenceManager pm = pmf.getPersistenceManager();

		Iterator i = pm.getExtent(Employee.class, false).iterator();
		while (i.hasNext()) {
			System.out.println((Employee)i.next());
		}
	}
}
```

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Collection;
import java.util.Iterator;
import java.util.Properties;

import javax.jdo.Extent;
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;
import javax.jdo.Query;

import de.jeckle.jdotest.Employee;
public class JDOQuery {
	public static void main(String args[]) {
		Properties props = new Properties();
		try {
			InputStream is =
				ClassLoader.getSystemResourceAsStream("jdo.properties");
			props.load(is);
		} catch (IOException ioe) {
			System.out.println("Error loading properties");
			System.exit(1);
		}
		PersistenceManagerFactory pmf =
			JDOHelper.getPersistenceManagerFactory(props);
		PersistenceManager pm = pmf.getPersistenceManager();

		Extent ext = pm.getExtent(Employee.class, false);
		String filter = "department == \"B042\"";
		Query qry = pm.newQuery(ext, filter);
		qry.setOrdering("name ascending");
		qry.compile();
		Collection c = (Collection) qry.execute();

		Iterator i = c.iterator();
		while (i.hasNext()) {
			System.out.println(i.next());
		}
	}

}
```

```
import java.io.IOException;
import java.io.InputStream;
import java.util.Collection;
import java.util.Properties;

import javax.jdo.Extent;
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;
import javax.jdo.Query;

import de.jeckle.jdotest.Employee;

public class JDODeleteObj {
	public static void main(String args[]) {
		Properties props = new Properties();
		try {
			InputStream is =
				ClassLoader.getSystemResourceAsStream("jdo.properties");
			props.load(is);
		} catch (IOException ioe) {
			System.out.println("Error loading properties");
			System.exit(1);
		}
		PersistenceManagerFactory pmf =
			JDOHelper.getPersistenceManagerFactory(props);
		PersistenceManager pm = pmf.getPersistenceManager();

		Extent ext = pm.getExtent(Employee.class, false);
		String filter = "name == \"Marta Smith\"";
		Query qry = pm.newQuery(ext, filter);
		Collection c = (Collection) qry.execute();

		pm.currentTransaction().begin();
		pm.deletePersistentAll(c);
		pm.currentTransaction().commit();
		System.out.println("Object deleted");
	}
}
```

```
javax.jdo.PersistenceManagerFactoryClass=com.triactive.jdo.PersistenceManagerFactoryImpl
javax.jdo.option.ConnectionURL=jdbc:mysql://localhost/jdotest/
javax.jdo.option.ConnectionDriverName=com.mysql.jdbc.Driver
javax.jdo.option.ConnectionUserName=mario
javax.jdo.option.ConnectionPassword=thePassword
com.triactive.jdo.autoCreateTables=true
```


Java Database Connectivity
Enterprise Java Bean
(bean managed persistence)
Java Data Objects
Transparenz
im Zugriff
Eingriffsmöglichen
für den Programmierer

Welcome to the TJDO Project Page

Links

- SourceForge Site
- Download
- Tutorial
- Docs
- JDO Javadocs
- TJDO Javadocs
- TJDO Roadmap
- Contributing Guidelines
- Browse CVS
- Sun's JDO Site

Overview

TriActive JDO (TJDO) is an open source implementation of Sun's JDO specification (JSR 12), designed to support transparent persistence using any JDBC-compliant database. TJDO has been deployed and running successfully in many commercial installations since 2001.

The current version has been tested successfully using Cloudscape, DB2, Firebird, MySQL, Oracle, PostgreSQL, SAP DB, and MS SQL Server.

Features

- Supports JDO 1.0.1.
- Implements the entire JDOQL query language, including several useful method enhancements.
- Auto-creates all necessary schema elements (tables, foreign keys, indexes) according to your app's classes and JDO metadata.
- Auto-validates expected schema structure at runtime, reducing failures from schema evolution.
- Can map Java classes to SQL views, and allows for direct SQL queries, to leverage SQL capabilities not normally accessible through JDO.
- Designed to be lightweight and fast.

Design Goals

As an O-R mapper TJDO falls under the category of *schema generators*, meaning it takes your user-written Java classes and, to the degree possible, fully automates the tasks required to transparently persist objects to a database. Although there are several ways to "tweak" the mapping strategy, for the most part the developer delegates these tasks to the JDO layer. Schema generators are best suited to projects whose persistent data model is being newly developed, or perhaps fully refactored, and whose developers would prefer to design mostly from an object rather than a relational point of view.

If things like DBMS portability and rapid development turnaround are more important to you than fine-grained control over exact data representations, then TJDO is probably what you're looking for. Conversely, if you're one of those who tend to spend long hours fussing over where you should "denormalize for performance", it probably is not.

Getting Started

New users can start by downloading the latest version and including the required jar files in their classpath. The tutorial, documentation, and the JDO API Javadocs are good places to begin when exploring TJDO. If you have any questions, comments, or problems please consult the documentation or use the TJDO forums on SourceForge.

Being an open source project, we are always interested in finding people to help with the project. If you are interested in contributing, contact the TJDO Project Administrator. You can find information about tasks, bugs, obtaining source code, etc... on the TJDO SourceForge Site.

Also, check out the TJDO roadmap to get an idea where we're heading. If you're interested in contributing to TJDO, please read the contributing guidelines. As always, you can post to the forums and let us know what you think.

We hope you find TJDO useful. Please let us know what you think of it.

News

March 31, 2004

TJDO 2.1 has been released! Binary and source distributions are available on SourceForge. This release includes several new features:

- Transient Transactional instances.
- Non-persistent transactional fields.
- Fields of type `byte[]` and `java.util.Hashtable`.
- Support for Oracle 9i.

As well as a few bug fixes. See the release notes for a full list of changes.

November 11, 2003

TJDO 2.0 Final was released today! Binary and source distributions are available for this release. This is the first TJDO release to be labeled as stable. Many new features have been added to the release, and many bugs have been fixed. Notable changes are: JDO 1.0.1 support, package-level metadata files, support for fields of type `HashSet`, the ability to pass persistent and transient `Sets` as parameters into `Querys`, the option to turn off table and constraint validation, the ability to utilize cascaded deletes, allowing non-nullable owner fields for inverse collections, RestoreValues mode, use of a validating XML parser, work towards passing the TCK test suite, many bug fixes, and much more. See the release notes for a full list of the changes.

Direct questions/comments to the TJDO Project Administrator.
Last Updated: $Date: 2004/04/01 07:18:50 $

JDOcentral.com
Developer's Community for Java Data Objects

JDO 1.0.1 is Now Available for Download!
http://access1.sun.com/jdo

Login
Password
Login | Join Now
Forgot your password?

Search JDOcentral.com

- JDO Success Stories
- JDO Books
- Meet the JDO Experts
- JDO Vendor Directory
- About Us
- Contact Us

Charter Sponsors

Latest Commentary [More]

Comparing SDO (JSR-235) to Detachment in JDO 2.0 (JSR-243) NEW

In response to remarks at TheServerSide.com I took some time to review the JSR-243 proposal for Service Data Objects. Based on the contents of this proposal I now offer a comparison between SDO and the Detachment capabilities of JDO 2.0 (JSR-243). I must admit that I have not had the time to read the preliminary SDO specification which was published by IBM and BEA prior to the submission of the SDO JSR, and that naturally my knowledge of JDO Detachment is substantially more detailed than the high-level view I have of SDO. The content here is offered in good faith, and may be refined over time in conjunction with the JSR-235 members as further information about SDO becomes available. [More]

Express Yourself! NEW

There is a heated debate going on over JDO 2.0, due to be released as a public draft very soon. IBM, Oracle, and BEA voted against JDO 2.0 as a JSR, but it still passed. They are asking developers to put their faith in EJB 3.0, with no delivery date in sight. JDO 2.0 is moving ahead, despite their attempts to stifle its momentum. Go to theserverside and "express yourself". JDO vendors have done a good job of speaking up, but having JDO users speak out will carry even more weight. [More]

JDO != Lowest Common Denominator - by Robin Roos

In the 2 years since JDO 1.0 was adopted as a standard by the JCP back in March 2002 the technology has attracted more than its fair share of criticism. The Java community has now accepted the wisdom of bytecode enhancement over alternative approaches, which was the biggest shouting point in the early months. It is important to note that the JDO 2.0 specification will no longer mandate bytecode enhancement (an implementation detail), but most vendors expect to continue along that path. [More]

JSR 243 for Java Data Objects 2.0 - by Craig Russell

On Tuesday 20-Apr-2004, Sun submitted Java Specification Request 243 to the Java Community Process for Java Data Objects 2.0. The submission of this JSR represents a significant commitment by Sun to the effort to promote database-independent programming standards for Java. Sun also recently renewed its charter membership in http://www.JDOCentral.com. JDOCentral is a strong voice for vendors and users alike to promote the vision of JDO. [More]

How Do You Spell Freedom? J-D-O

JDO provides freedom, freedom from storage dependencies. With a simple change to a couple of properties in a property file, an application can be switched between an object database and a relational database, without needing to re-compile or re-enhance the classfiles. Be sure and read the latest article published by JavaPro proclaiming the portability benefits of JDO. [More]

JDO, then and now

In the past five years, I've seen the software industry from a variety of perspectives...Over the last decade, I've witnessed remarkable change in our overall approach to persistence. In this article, I'd like to share my experiences, and how I see the future of JDO. [More]

JBoss Chooses Java Data Objects

Several months ago, Marc Fleury announced that JBoss would make their CMP implementation available to developers working on persistence solutions outside the server environment. With no details, rumors flew about what this actually meant for persistence and the open source community. [More]

JDO or CMP?

Importance of JDO 2.0

How Important is JDO 2.0 to You?

- Very important
- I'd rather wait for EJB 3.0
- Don't know

Vote

"Last week I gave a presentation on JDO at the New England Java Users Group ... There were approximately 200 attendees who stayed the entire 2 hours and asked a lot of questions. ... I think the interest was more than polite. People are starting to factor JDO into their development plans."
-- [David Ezzio, JDO Expert]

New Content Around the Site

JDO Article: New Features in the JDO 2.0 Query Language

[By David Jordan] Query language enhancements are one of the primary areas being addressed in the JDO 2.0 specification. This article provides a high level overview of the capabilities being introduced in JDO 2.0. Since the JDO 2.0 specification has not yet reached public draft stage, nothing is cast in stone, so things are still subject to change. However, the JDO 2.0 specification should be completed fairly soon and these query features have been the most stable set of features the expert group has dealt with. So I am very confident that the final specification will be very similar to the information presented here. [More] [Chinese Translation]

JDO Success Story: Rapid - An Application Framework Built On JDO

This paper presents a case study of how Anaglyph Solutions developed an entire application framework using Java Data Objects (JDO) as a base layer. Our framework combines a good commercial JDO implementation with open-source libraries for XML parsing, JavaBean handling, and other important architectural functions. The end result is a tool that allows us to develop complex applications in a very small amount of time, providing greater value to our clients. [More]

JDO FREE Download: WebApp-Emp Application

Amir Firdus has designed and developed a complete web-based application that uses JDO and Struts. The application implements the Party pattern with the roles employer and employee, which can be associated with a number of addresses and can share an employment relationship. He has graciously decided to make the software available to the JDO community. [More]

JDO Article: Managing Your Relationships

If you are like most developers trying out JDO for the first time, you are building an application that will manage data in a relational database. This may be an existing relational database with a schema already defined, in which case you need to develop a corresponding Java model. Or, you may be building a brand new Java application and do not have an existing relational schema that you need to map to. [More]

Java Data Objects (JDO) and Enterprise JavaBeans (EJB) Container Managed Persistence (CMP) were developed concurrently, but took different paths. To see where each technology is applicable, read the following article. You can also read a Chinese translation of this article by clicking here. Check out the discussion of this article on TheServerSide and support JDO by adding your own views to the discussion thread.

2004 Featured Events [More]

5/24 - 5/26	JAOO Symposium - Cannes, France
5/24 - 5/26	JDO Hands-On Workshop - Huntington Beach, CA
6/14 - 6/15	JDO Introduction Workshop - London, England
6/15 - 6/16	JDO Introduction Workshop - Paris, France
6/28 - 6/30	JDO Hands-On Workshop - Huntington Beach, CA

In the News [More]

5/14/04	Riflexo released JCredo 1.3 to keep JDO developers current with JDO 2.0 NEW
3/24/04	SolarMetric's Kodo JDO Named Finalist for "Best Java Persistence Architecture" and "Best Java Data Access Tool" Awards by the Readership of Java Developers Journal
3/01/04	ObjectStore Announces Version 6.1, Incorporating New Standards to Ease Development and Porting of Java-Based Applications: Small Footprint Version of ObjectStore®, PSE Pro™, Adds Support for Java® Data Objects
2/25/04	JDO Genie 3.0.0beta Press Release - Use JDK 1.5 Generics With JDO
1/15/04	Stellar Performance With New JDO Genie 2.2.0
1/14/04	Riflexo Released JCredo 1.2
12/15	Versant Corp. Becomes Charter Member of JDOcentral.com
12/12	Riflexo Releases JCredo JDO Professional & Enterprise Editions
12/11	SolarMetric Releases Latest Version of Popular Kodo JDO Product: Introduces Preview Features of the JDO 2.0 Standard
10/29	JInspired Announces JDBInsight 2.0 Release
10/08	Versant Delivers JDO for Versant Object Database
10/07	Riflexo Released JCredo 1.0, Free Compliant JDO Implementation

Whitepapers [More]

Best Practices for JDO Performance - FastObjects NEW

JDOQL: The JDO Query Language - David Jordan/Object Identity, Inc.

A Comparison Between Java Data Objects (JDO), Serialization and JDBC for Java Persistence - David Jordan/Object Identity, Inc.

Buy Versus Build - Neelan Choski/SolarMetric, Inc.

Java Data Objects - The Future for Java Object Persistence - Keiron McCammon/Versant Corporation

Why JDO is a Critical Component of J2EE? - LIBeLIS

OpenFusion - Java Data Objects - PrismTech

Articles [More]

4/20/04	JDO O/R mapping DAO ValueObject Hibernate EJB [Chinese] NEW
4/5/04	[Chinese Translation]: New Features in the JDO 2.0 Query Language NEW
3/29/04	New Features in the JDO 2.0 Query Language
3/10/04	How Do You Spell Freedom? J-D-O
2/24/04	Managing Your Relationships
2/3/04	The Perfect Database Round-Trip Using JAXB & JDO
2/3/04	Tech Talk: Bruce Tate on EJB, JDO, and J2EE Best Practices - [Streaming Video]
1/26/04	Versant Edition Of Kodo JDO Now Available
1/16/04	JDO Architecture
1/15/04	Kodo 3.1 to Get JMX
1/1/04	Java Data Objects - An Introduction
11/21	JDO 2.0 Preview: Java Object Persistence Made Easy
11/17	JDO Meets Maven
9/30	Versant Includes JDO in Object Database
9/15	Sun Back on JDO Bandwagon
9/9	Sample Of a Persistence Capable Class Wrapping an Image In a Gallery
8/21	JDO 2.0: Lot's of Changes Discussed at the Kickoff Meeting
8/7	Y'all Bought Too Much Oracle

JDO Success Story: Cardif Standardizes on the use of LIDO

Today, ease of communication between data and applications must be straightforward. Up to now, mapping between object and relations databases was limited to experts. JDO and more specifically LiDO bring an east thus powerful solution. Cardif confirmed this fact. [More]

JDO Meets Maven

Maven is a tool for managing and building projects providing an alternative to the accepted Ant build process. Maven is almost totally plug-in-driven, and provides plug-ins for many common tasks (for example: building EJB components such as WARs and EARs, generating documentation, running unit tests) and related software (for example: Checkstyle, PMD, Clover, JCoverage). [More]

JDO 2.0: Lot's of Changes Discussed at the Kickoff Meeting

"...Working groups with competing vendors got together over the 3 day meeting to work on things. Vendors were discussing what they did right now, and what they were planning to do in the future. I probably learnt much about the different JDO vendor products just via this information alone..." [More]

Dedicated JDO Books Page Now Available

More and more JDO books are hitting the book shelves. You can find an overview what is available today right here... [More]

JDO Success Story: Building Multimedia Informatics Systems Using JDO

Multimedia Informatics: Learn How JDO and FastObjects are being used at UCLA. [More]

JDO Success Story: CastorTPV 1.2 - A Case Study of a JDO Based Application

Advanced Quality Solutions is a Spanish IT provider whose portfolio includes commercial software products, custom development, and system maintenance. [More]

More JDO Success Stories... [More]

JDO Expert Corner

Tutorial of WebApp on JDO & JSP 2.0 (by Bin Sun)

JThis tutorial will introduce the development process of a web based database application conforming J2EE standard. For the purpose of “simpler, better”, the application will be built on JSP2.0 and take advantage of the currently fast-spreading Java Data Object specification. JSP2.0’s JSP EL can greatly simplize JSP development and hence economize debug time and increase readability of JSP pages.

Sample Of a Persistence Capable Class Wrapping an Image In a Gallery

(by Stephen Kolaroff)

This example demonstrates a strategy for dealing with user-specified pictures (or "avatars") and photos, and how to store them in a gallery. The class itself may be re-used in different contexts with quite little effort. We will generalize this strategy and provide hints about its applicability. The code example, which is available on JDO Central Code Exchange page, is an extract of a full JDO code sample showing a web application called "Senat" discussion forum (http://jcredo.com/senat), which full code can be downloaded together with JCredo - Riflexo's free JDO implementation - on jcredo.com.

Creating and Using Simple Extended Optimistic Transactions (by David Ezzio)

JDO 1.0 provides two kinds of transactions, but both assume that the persistence manager is open during the duration of the transaction. In web and enterprise applications, the persistence manager may close between the view and the update requests. In these architectures, the application may need a technique to verify that the version of the object when changed is the same as the version when viewed.

HOT TOPICS in the JDO Community Forums

EJB vs. JDO, Why JDO is better than EJB

Hello: I have a problem...Why JDO is better than EJB...convide me...

The JDO is client Side, How it acts when in a distributed application? how it would be the integration of jdo with ejb if ejb is distributed? it considers that EJB is version java of the CORBA, and JDO is only persisted of objects, which the beneficios to join the two technologies

Did you understand??

Step by Step help to use jdori, Step by Step help to use jdori

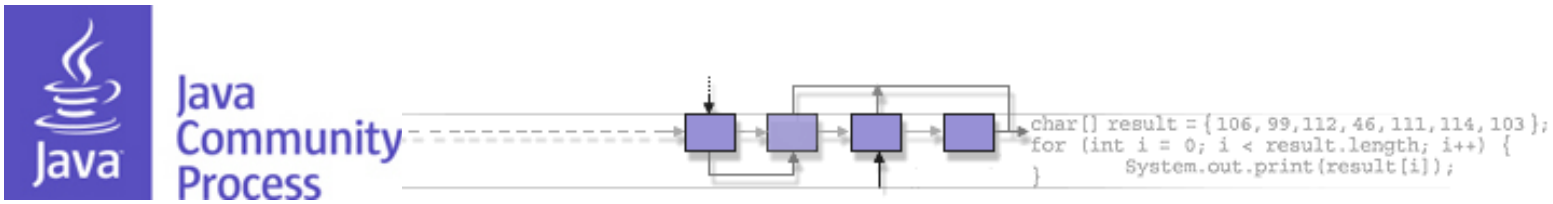
Hello, I am working with java 1.4.1 SE and need to make instance classes persistent for further computation.
I found jdo interesting but find implementation hard to carry and time consuming as i should learn about ant, xml, etc.

Does someone know a step by step procedure to use jdori.

Best regards,
Farah

Persistent Identity: The Persistent Object's Essential Attribute (by David Ezzio)

Persistent identity is the essential attribute of persistent objects. JDO provides both datastore identity and application identity. Because JDO makes datastore identity opaque, the application needs a technique to capture it. The data object's equals method uses the persistent identity to determine when two memory objects refer to the same persistent object.



JSR-000012 Java™ Data Objects (JDO) Specification

(Maintenance Release)

This is an updated Final Release of this Specification, as described in Section 4.2 of the Java Community Process℠ Program, version 2.1.

Specification:

- Download

Maintenance Lead:

- Craig Russell
  Sun Microsystems, Inc.

  Please send comments to jdo-comments@sun.com.

Reference Implementation and Technology Compatibility Kit:

- JDO Reference Implementation Source
- JDO Reference Implementation Binary
- JDO Technology Compatibility Kit

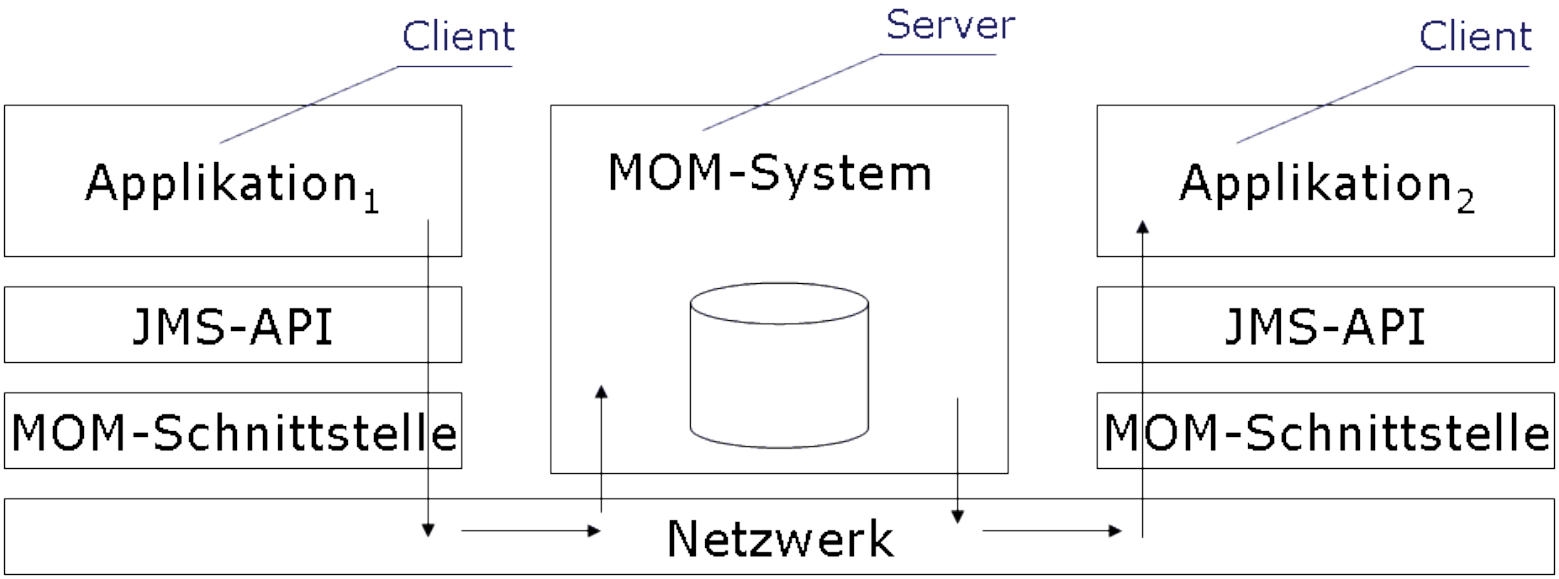
Change Log:

- JSR 12 Change Log

See Also

JSR-000012 Java™ Data Objects Page
Previous Final Release of JSR-000012 Java™ Data Objects Specification
JCP Final Releases Page
Community Process Main Page

Search


Client
Server
Client
Applikation1
MOM-System
Applikation2
JMS-API
JMS-API
MOM-Schnittstelle
MOM-Schnittstelle
Netzwerk

Web Services Architecture

W3C Working Group Note 11 February 2004

This version:
http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
Latest version:
http://www.w3.org/TR/ws-arch/
Previous version:
http://www.w3.org/TR/2003/WD-ws-arch-20030808/
Editors:
David Booth, W3C Fellow / Hewlett-Packard
Hugo Haas, W3C
Francis McCabe, Fujitsu Labs of America
Eric Newcomer (until October 2003), Iona
Michael Champion (until March 2003), Software AG
Chris Ferris (until March 2003), IBM
David Orchard (until March 2003), BEA Systems

This document is also available in these non-normative formats: PostScript version and PDF version.

Abstract

This document defines the Web Services Architecture. It identifies the functional components and defines the relationships among those components to effect the desired properties of the overall architecture.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is a public Working Group Note produced by the W3C Web Services Architecture Working Group, which is part of the W3C Web Services Activity. This publication as a Working Group Note coincides with the end of the Working Group's charter period, and represents the culmination of the group's work.

Discussion of this document is invited on the public mailing list www-ws-arch@w3.org (public archives). A list of remaining open issues is included in **4 Conclusions**.

Patent disclosures relevant to this specification may be found on the Working Group's patent disclosure page.

Publication as a Working Group Note does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress. Other documents may supersede this document.

Table of Contents

Appendices

---

1 Introduction

1.1 Purpose of the Web Service Architecture

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. This document (WSA) is intended to provide a common definition of a Web service, and define its place within a larger Web services framework to guide the community. The WSA provides a conceptual model and a context for understanding Web services and the relationships between the components of this model.

The architecture does not attempt to specify how Web services are implemented, and imposes no restriction on how Web services might be combined. The WSA describes both the minimal characteristics that are common to all Web services, and a number of characteristics that are needed by many, but not all, Web services.

The Web services architecture is an *interoperability* architecture: it identifies those global elements of the global Web services network that are required in order to ensure interoperability between Web services.

1.2 Intended Audience

This document is intended for a diverse audience. Expected readers include Web service specification authors, creators of Web service software, people making decisions about Web service technologies, and others.

1.3 Document Organization

This document has two main sections: a core concepts section (**2 Concepts and Relationships**) and a stakeholder's perspectives section (**3 Stakeholder's Perspectives**).

2 Concepts and Relationships provides the bulk of the conceptual model on which conformance constraints could be based. For example, the resource concept states that resources have identifiers (in fact they have URIs). Using this assertion as a basis, we can assess conformance to the architecture of a particular resource by looking for its identifier. If, in a given instance of this architecture, a resource has

no identifier, then it is not a valid instance of the architecture.

While the concepts and relationships represent an enumeration of the architecture, the stakeholders' perspectives approaches from a different viewpoint: how the architecture meets the goals and requirements. In this section we elucidate the more global properties of the architecture and demonstrate how the concepts actually achieve important objectives.

A primary goal of the Stakeholder's Perspectives section is to provide a top-down view of the architecture from various perspectives. For example, in the **3.6 Web Services Security** section we show how the security of Web services is addressed within the architecture. The aim here is to demonstrate that Web services can be made secure and indicate which key concepts and features of the architecture achieve that goal.

The key stakeholder's perspectives supported in this document reflect the major goals of the architecture itself: interopability, extensibility, security, Web integration, implementation and manageability.

1.4 What is a Web service?

For the purpose of this Working Group and this architecture, and without prejudice toward other definitions, we will use the following definition:

[Definition: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.]

1.4.1 Agents and Services

A Web service is an abstract notion that must be implemented by a concrete agent. (See Figure 1-1) The agent is the concrete piece of software or hardware that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided. To illustrate this distinction, you might implement a particular Web service using one agent one day (perhaps written in one programming language), and a different agent the next day (perhaps written in a different programming language) with the same functionality. Although the agent may have changed, the Web service remains the same.

1.4.2 Requesters and Providers

The purpose of a Web service is to provide some functionality on behalf of its owner -- a person or organization, such as a business or an individual. The *provider entity* is the person or organization that provides an appropriate agent to implement a particular service. (See Figure 1-1: Basic Architectural Roles.)

A *requester entity* is a person or organization that wishes to make use of a provider entity's Web service. It will use a *requester agent* to exchange messages with the provider entity's *provider agent*.

(In most cases, the requester agent is the one to initiate this message exchange, though not always. Nonetheless, for consistency we still use the term "requester agent" for the agent that interacts with the provider agent, even in cases when the provider agent actually initiates the exchange.)

> **Note:**
>
> A word on terminology: Many documents use the term service provider to refer to the provider entity and/or provider agent. Similarly, they may use the term service requester to refer to the

requester entity and/or requester agent. However, since these terms are ambiguous -- sometimes referring to the agent and sometimes to the person or organization that owns the agent -- this document prefers the terms *requester entity*, *provider entity*, *requester agent* and *provider agent*.

In order for this message exchange to be successful, the requester entity and the provider entity must first agree on both the semantics and the mechanics of the message exchange. (This is a slight simplification that will be explained further in **3.3 Using Web Services**.)

1.4.3 Service Description

The mechanics of the message exchange are documented in a Web service description (WSD). (See Figure 1-1) The WSD is a machine-processable specification of the Web service's interface, written in WSDL. It defines the message formats, datatypes, transport protocols, and transport serialization formats that should be used between the requester agent and the provider agent. It also specifies one or more network locations at which a provider agent can be invoked, and may provide some information about the message exchange pattern that is expected. In essence, the service description represents an agreement governing the mechanics of interacting with that service. (Again this is a slight simplification that will be explained further in **3.3 Using Web Services**.)

1.4.4 Semantics

The semantics of a Web service is the shared expectation about the behavior of the service, in particular in response to messages that are sent to it. In effect, this is the "contract" between the requester entity and the provider entity regarding the purpose and consequences of the interaction. Although this contract represents the overall agreement between the requester entity and the provider entity on how and why their respective agents will interact, it is not necessarily written or explicitly negotiated. It may be explicit or implicit, oral or written, machine processable or human oriented, and it may be a legal agreement or an informal (non-legal) agreement. (Once again this is a slight simplification that will be explained further in **3.3 Using Web Services**.)

While the service description represents a contract governing the mechanics of interacting with a particular service, the semantics represents a contract governing the meaning and purpose of that interaction. The dividing line between these two is not necessarily rigid. As more semantically rich languages are used to describe the mechanics of the interaction, more of the essential information may migrate from the informal semantics to the service description. As this migration occurs, more of the work required to achieve successful interaction can be automated.

1.4.5 Overview of Engaging a Web Service

There are many ways that a requester entity might engage and use a Web service. In general, the following broad steps are required, as illustrated in Figure 1-1: (1) the requester and provider entities become known to each other (or at least one becomes know to the other); (2) the requester and provider entities somehow agree on the service description and semantics that will govern the interaction between the requester and provider agents; (3) the service description and semantics are realized by the requester and provider agents; and (4) the requester and provider agents exchange messages, thus performing some task on behalf of the requester and provider entities. (I.e., the exchange of messages with the provider agent represents the concrete manifestation of interacting with the provider entity's Web service.) These steps are explained in more detail in **3.4 Web Service Discovery**. Some of these steps may be automated, others may be performed manually.

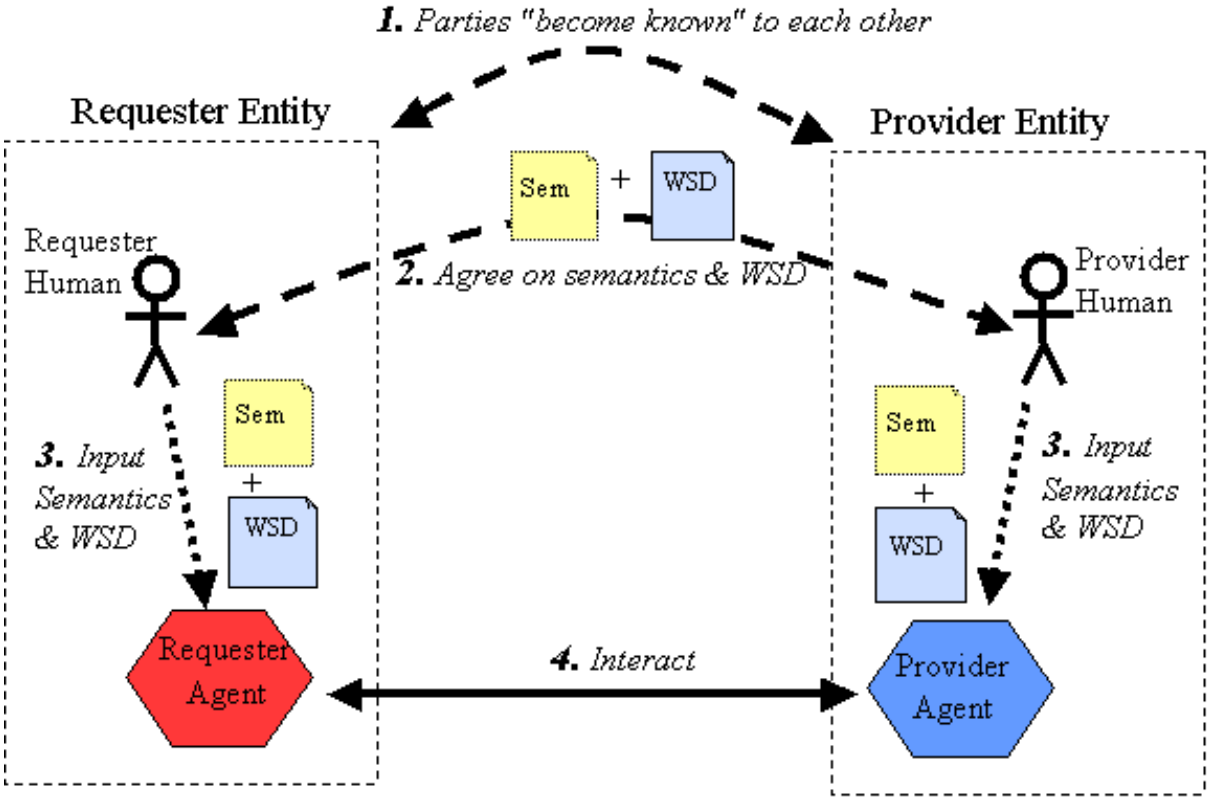


Figure 1-1. The General Process of Engaging a Web Service

1.5 Related Documents

The Working Group produced the following companion documents in the process of defining this architecture:

- Requirements Document [WSA Reqs]
- Usage Scenarios [WSAUS]
- Glossary [WS Glossary]
- OWL Ontology [OWLO]

2 Concepts and Relationships

2.1 Introduction

The formal core of the architecture is this enumeration of the concepts and relationships that are central to Web services' interoperability.

2.2 How to read this section

The architecture is described in terms of a few simple elements: concepts, relationships and models. Concepts are often noun-like in that they identify things or properties that we expect to see in realizations of the architecture, similarly relationships are normally linguistically verbs.

As with any large-scale effort, it is often necessary to structure the architecture itself. We do this with the larger-scale meta-concept of model. A model is a coherent portion of the architecture that focuses on a particular theme or aspect of the architecture.

2.2.1 Concepts

A concept is expected to have some correspondence with any realizations of the architecture. For example, the message concept identifies a class of object (not to be confused with Objects and Classes as are found in Object Oriented Programming languages) that we expect to be able to identify in any Web services context. The precise form of a message may be different in different realizations, but the message concept tells us what to look for in a given concrete system rather than prescribing its precise form.

Not all concepts will have a realization in terms of data objects or structures occurring in computers or communications devices; for example the person or organization refers to people and human organizations. Other concepts are more abstract still; for example, message reliability denotes a property of the message transport service — a property that cannot be touched but nonetheless is important to Web services.

Each concept is presented in a regular, stylized way consisting of a short definition, an enumeration of the relationships with other concepts, and a slightly longer explanatory description. For example, the concept of agent includes as relating concepts the fact that an agent is a computational resource, has an identifier and an owner. The description part of the agent explains in more detail why agents are important to the archicture.

2.2.2 Relationships

Relationships denote associations between concepts. Grammatically, relationships are verbs; or more accurately, predicates. A statement of a relationship typically takes the form: concept predicate concept. For example, in agent, we state that:

An agent is

> a computational resource

This statement makes an assertion, in this case about the nature of agents. Many such statements are descriptive, others are definitive:

A message has

> a message sender

Such a statement makes an assertion about valid instances of the architecture: we expect to be able to identify the message sender in any realization of the architecture. Conversely, any system for which we cannot identify the sender of a message is not conformant to the architecture. Even if a service is used anonymously, the sender has an identifier but it is not possible to associate this identifier with an actual person or organization.

2.2.3 Concept Maps

Many of the concepts in the architecture are illustrated with *concept maps*. A concept map is an informal, graphical way to illustrate key concepts and relationships. For example the diagram:

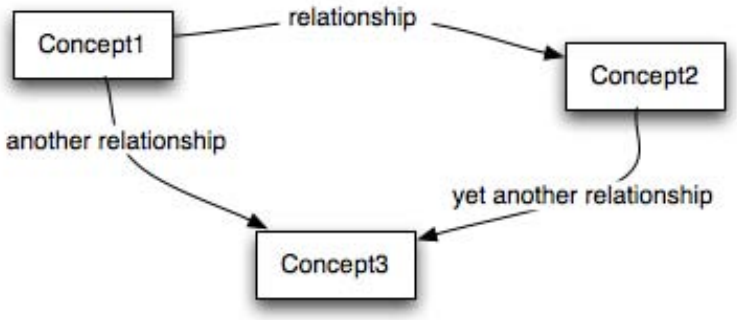


Figure 2-1. Concept Map

shows three concepts which are related in various ways. Each box represents a concept, and each arrow (or labeled arc) represents a relationship.

The merit of a concept map is that it allows rapid navigation of the key concepts and illustrates how they relate to each other. It should be stressed however that these diagrams are primarily navigational aids; the written text is the definitive source.

2.2.4 Model

A model is a coherent subset of the architecture that typically revolves around a particular aspect of the overall architecture. Although different models share concepts, it is usually from different points of view; the major role of a model is to explain and encapsulate a significant theme within the overall Web services architecture.

For example, the Message Oriented Model focuses and explains Web services strictly from a message passing perspective. In particular, it does not attempt to relate messages to services provided. The Service Oriented Model, however, lays on top of and extends the Message Oriented Model in order to explain the fundamental concepts involved in service - in effect to explain the purpose of the messages in the Message Oriented Model.

Each model is described separately below, in terms of the concepts and relationships inherent to the model. The ordering of the concepts in each model section is alphabetical; this should not be understood to imply any relative importance. For a more focused viewpoint the reader is directed to the Stakeholder's perspectives section which examines the architecture from the perspective of key stakeholders of the architecture.

The reason for choosing an alphabetical ordering is that there is a large amount of cross-referencing between the concepts. As a result, it is very difficult, if not misleading, to choose a non-alphabetic ordering that reflects some sense of priority between the concepts. Furthermore, the optimal ordering depends very much on the point of view of the reader. Hence, we devote the Stakeholders perspectives section to a number of prioriterized readings of the architecture.

2.2.5 Conformance

Unlike language specifications, or protocol specifications, conformance to an architecture is necessarily a somewhat imprecise art. However, the presence of a concept in this enumeration is a strong hint that, in any realization of the architecture, there should be a corresponding feature in the implementation. Furthermore, if a relationship is identified here, then there should be corresponding relationships in any realized architecture. The consequence of non-conformance is likely to be reduced interoperability: The absence of such a concrete feature may not prevent interoperability, but it is likely to make such interoperability more difficult.

A primary function of the Architecture's enumeration in terms of models, concepts and relationships is to

give guidance about conformance to the architecture. For example, the architecture notes that a message has a message sender; any realization of this architecture that does not permit a message to be associated with its sender is not in conformance with the architecture. For example, SMTP could be used to transmit messages. However, since SMTP (at present) allows forgery of the sender's identity, SMTP by itself is not sufficient to discharge this responsibility.

2.3 The Architectural Models

This architecture has four models, illustrated in Figure 2-2. Each model in Figure 2-2 is labeled with what may be viewed as the key concept of that model.

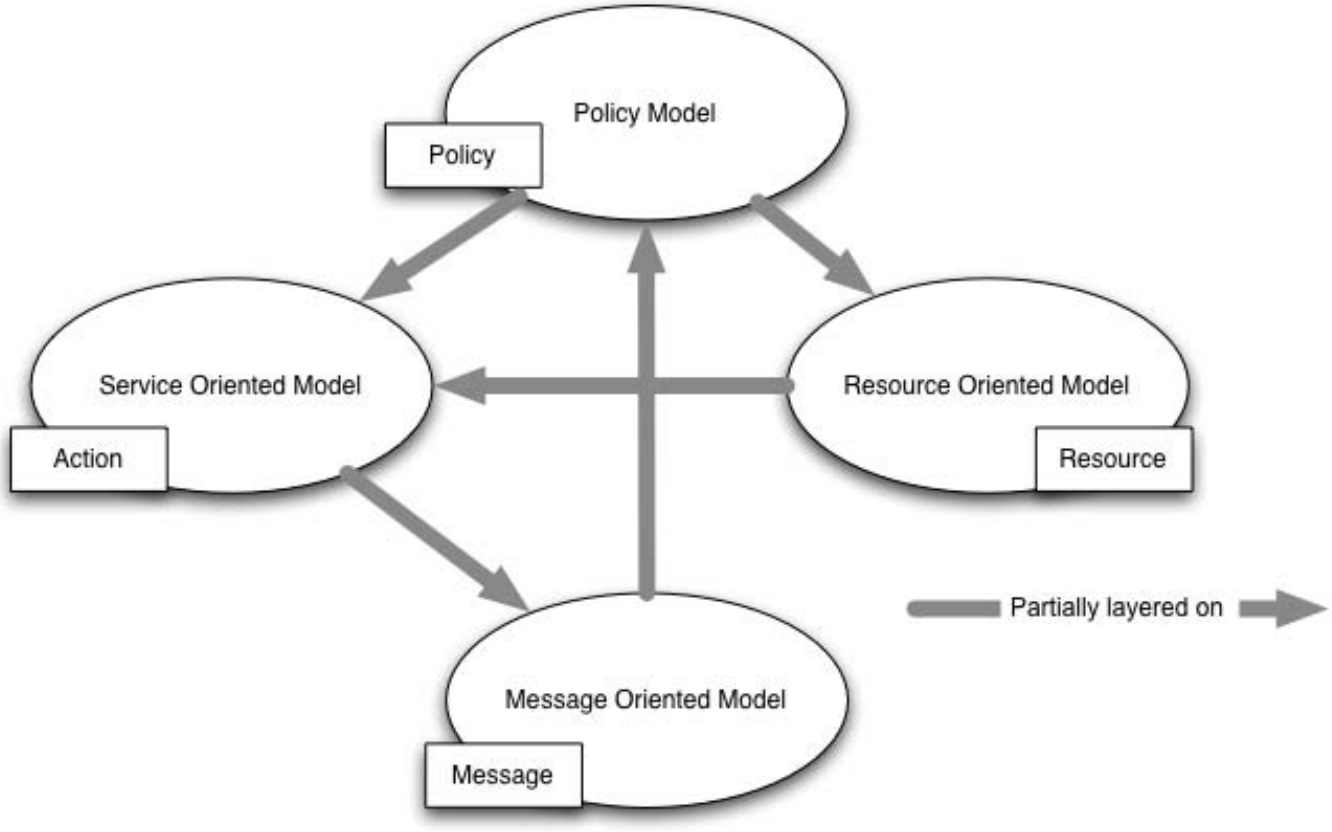


Figure 2-2. Meta Model of the Architecture

The four models are:

- The Message Oriented Model focuses on messages, message structure, message transport and so on — without particular reference as to the reasons for the messages, nor to their significance.

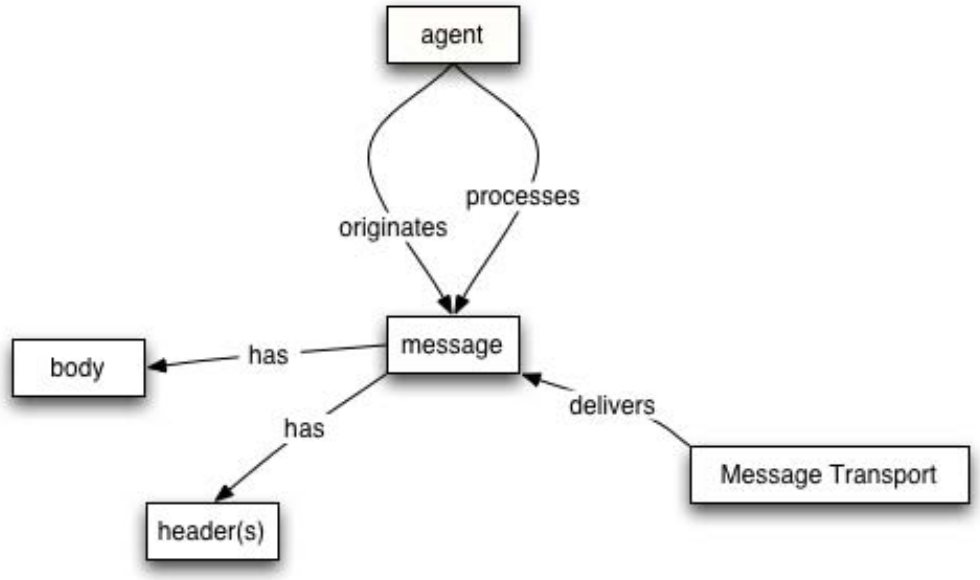


Figure 2-3. Simplified Message Oriented Model

The essence of the message model revolves around a few key concepts illustrated above: the agent that sends and receives messages, the structure of the message in terms of message headers and bodies and the mechanisms used to deliver messages. Of course, there are additional details to consider: the role of policies and how they govern the message level model. The abridged diagram shows the key concepts; the detailed diagram expands on this to include many more concepts and relationships.

- The Service Oriented Model focuses on aspects of service, action and so on. While clearly, in any distributed system, services cannot be adequately realized without some means of messaging, the converse is not the case: messages do not need to relate to services.

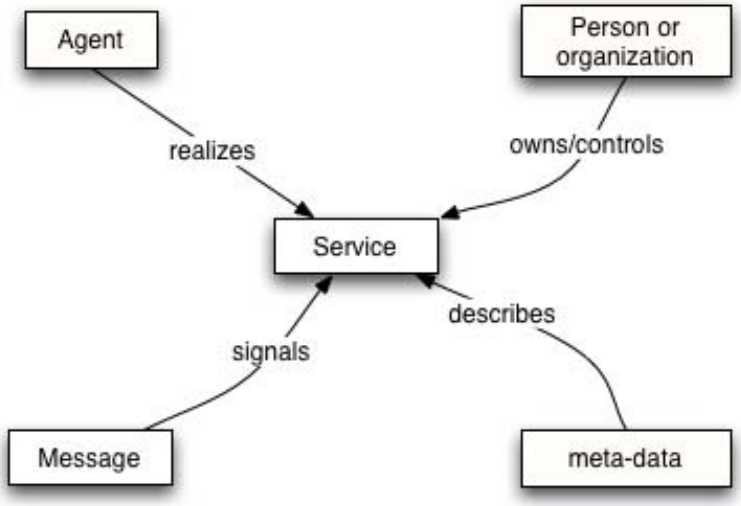


Figure 2-4. Simplified Service Oriented Model

The Service Oriented Model is the most complex of all the models in the architecture. However, it too revolves around a few key ideas. A service is realized by an agent and used by another agent. Services are mediated by means of the messages exchanged between requester agents and provider agents.

A very important aspect of services is their relationship to the real world: services are mostly

deployed to offer functionality in the real world. We model this by elaborating on the concept of a service's owner — which, whether it is a person or an organization, has a real world responsibility for the service.

Finally, the Service Oriented Model makes use of meta-data, which, as described in **3.1 Service Oriented Architecture**, is a key property of Service Oriented Architectures. This meta-data is used to document many aspects of services: from the details of the interface and transport binding to the semantics of the service and what policy restrictions there may be on the service. Providing rich descriptions is key to successful deployment and use of services across the Internet.

- The Resource Oriented Model focuses on resources that exist and have owners.

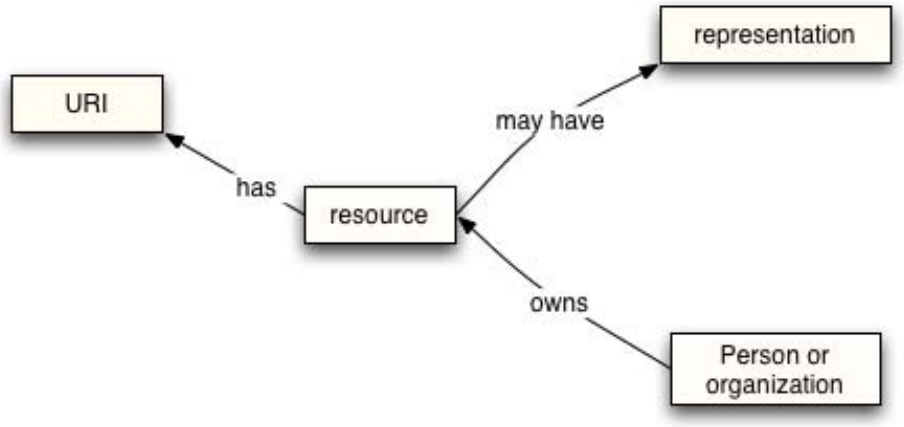


Figure 2-5. Simplified Resource Oriented Model

The resource model is adopted from the Web Architecture concept of resource. We expand on this to incorporate the relationships between resources and owners.

- The Policy Model focuses on constraints on the behavior of agents and services. We generalize this to resources since policies can apply equally to documents (such as descriptions of services) as well as active computational resources.

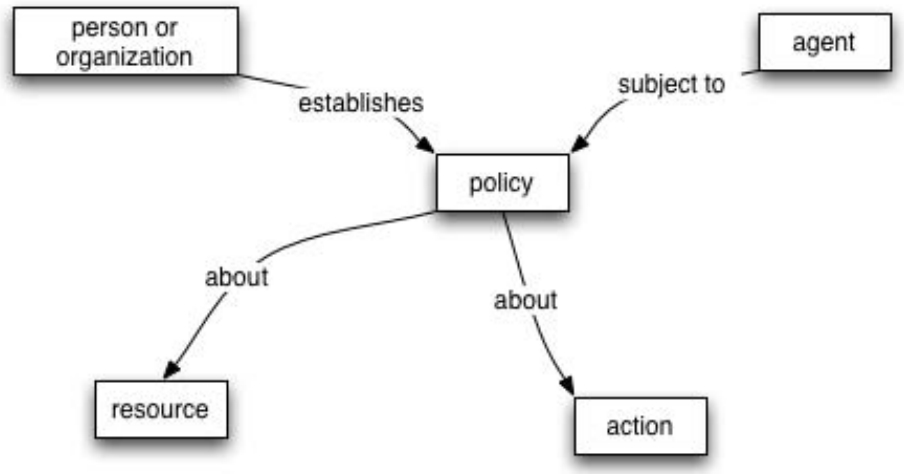


Figure 2-6. Simplified Policy Model

Policies are about resources. They are applied to agents that may attempt to access those

resources, and are put in place, or established, by people who have responsibility for the resource.

Policies may be enacted to represent security concerns, quality of service concerns, management concerns and application concerns.

2.3.1 Message Oriented Model

The Message Oriented Model focuses on those aspects of the architecture that relate to messages and the processing of them. Specifically, in this model, we are not concerned with any semantic significance of the content of a message or its relationship to other messages. However, the MOM does focus on the structure of messages, on the relationship between message senders and receivers and how messages are transmitted.

The MOM is illustrated in the Figure 2-7:

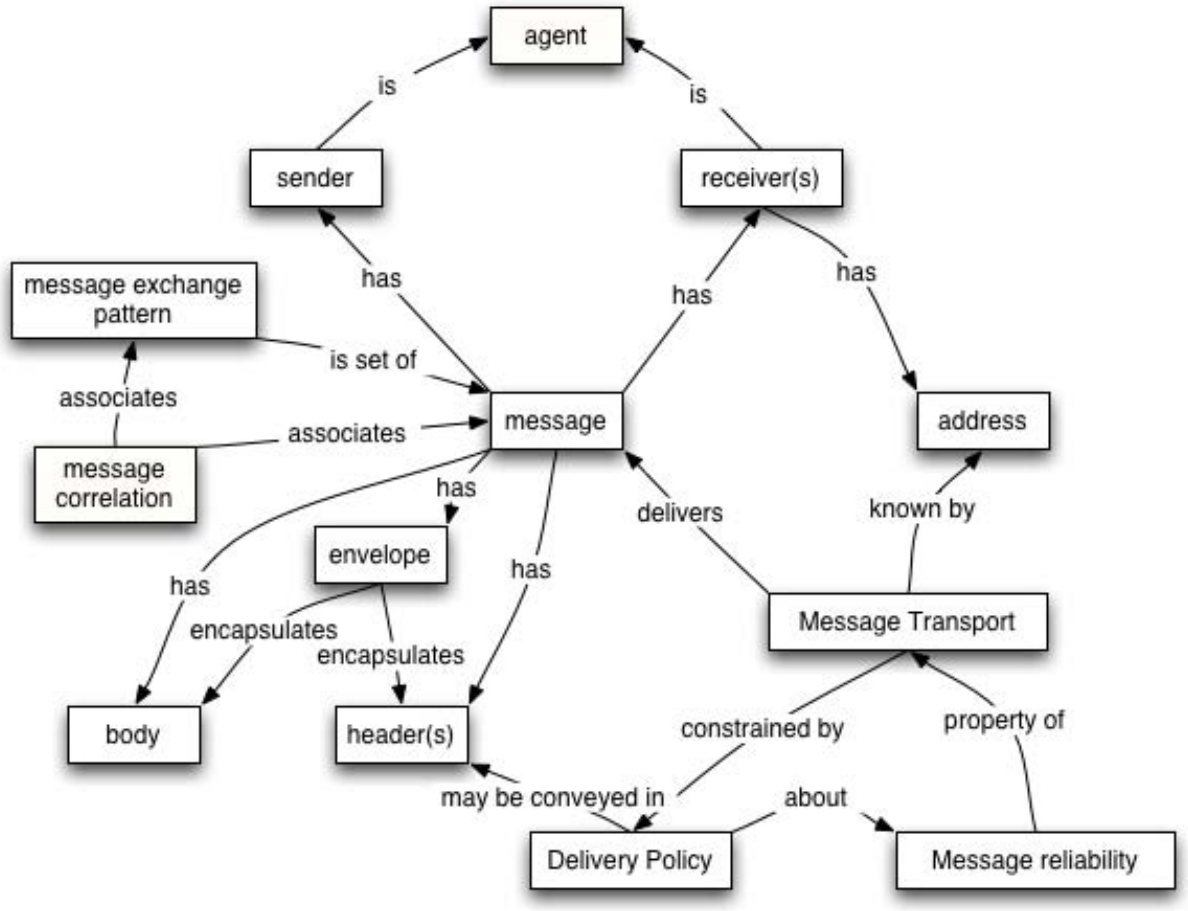


Figure 2-7. Message Oriented Model

2.3.1.1 Address

2.3.1.1.1 Definition

An address is that information required by a message transport mechanism in order to deliver a message appropriately.

2.3.1.1.2 Relationships to other elements

An address is

information used to describe how and where to deliver messages.

An address may be

a URI.

An address is

typically transport mechanism specific.

An address may be contained

in the message envelope.

2.3.1.1.3 Explanation

In order for message transport mechanisms to function, it is normally necessary to provide information that allows messages to be delivered. This is called the address of the message receiver.

Typically, the form of the address information will depend of the particular message transport. In the case of an HTTP message transport, the address information will take the form of a URL.

The precise method that a message sender uses to convey address information will also depend on the transport mechanism used. On occasion, the address information may be provided as additional arguments to the invoking procedure. Or the address information may be located within the message itself; typically in the message envelope.

2.3.1.2 Delivery Policy

2.3.1.2.1 Definition

A delivery policy is a policy that constrains the methods by which messages are delivered by the message transport.

2.3.1.2.2 Relationships to other elements

Delivery policy is

a policy

Delivery policy constrains

message transport

Delivery policy may be expressed

in a policy description language

Delivery policy may express

the quality of service associated with delivering a message by a message transport mechanism

2.3.1.2.3 Explanation

Delivery policies are those policies that relate to the delivery of messages.

Typically, a delivery policy applies to the combination of a particular message and a particular message transport mechanism. The policies that apply, however, may originate from descriptions in the message itself, or be intrinsic to the transport mechanism, or both.

Examples of delivery policies include quality of service assurances — such as reliable versus best effort message delivery — and security assurances — such as encrypted versus unencrypted message transport. Another kind of delivery policy could take the form of assertions about recording an audit of how the message was delivered.

2.3.1.3 Message

2.3.1.3.1 Definition

A message is the basic unit of data sent from one Web services agent to another in the context of Web services.

2.3.1.3.2 Relationships to other elements

a message is

a unit of data sent from one agent to another

a message may be part of

a message sequence

a message may be described using

a service description language

a message has

a message sender

a message has

one or more message recipients

a message may have

an identifier

a message has

a message body

a message has

zero or more message headers

a message has

a message envelope

a message is delivered by

a message transport system

a message may have

a delivery policy associated with it

2.3.1.3.3 Explanation

A message represents the data structure passed from its sender to its recipients. The structure of a message is defined in a service description.

The main parts of a message are its envelope, a set of zero or more headers, and the message body.

The envelope serves to encapsulate the component parts of the message and it serves as a well-known location for message transport services to locate necessary addressing information. The header holds ancillary information about the message and facilitates modular processing. The body of the message contains the message content or URIs to the actual data resource.

A message can be as simple as an HTTP GET request, in which the HTTP headers are the headers and the parameters encoded in the URL are the content. Note that extended Web services functionality in this architecture is not supported in HTTP headers.

A message can also simply be a plain XML document. However, such messages do not support extended Web services functionality defined in this architecture.

A message can be a SOAP XML, in which the SOAP headers are the headers. Extended Web services functionality are supported in SOAP headers.

2.3.1.4 Message Body

2.3.1.4.1 Definition

A message body is the structure that represents the primary application-specific content that the message sender intends to deliver to the message recipient.

2.3.1.4.2 Relationships to other elements

a message body is contained by

the message envelope.

a message body is

the application-specific content intended for the message recipient.

2.3.1.4.3 Explanation

The message body provides a mechanism for transmitting information to the recipient of the message. The form of the message body, and other constraints on the body, may be expressed as part of the service description.

In many cases, the precise interpretation of the message body will depend on the message headers that are in the message.

2.3.1.5 Message Correlation

2.3.1.5.1 Definition

Message correlation is the association of a message with a context. Message correlation ensures that a requester agent can match the reply with the request, especially when multiple replies may be possible.

2.3.1.5.2 Relationships to other elements

Message Correlation is

a means of associating a message within a specific conversational context.

Message correlation may be realized

by including message identifiers to enable messages to be identified.

2.3.1.5.3 Explanation

Message correlation allows a message to be associated with a particular purpose or context. In a conversation, it is important to be able to determine that an actual message that has been received is the expected message. Often this is implicit when conversations are relayed over stream-oriented message transports; but not all transports allow correlation to be established so implicitly.

For situations where correlation must be handled explicitly, one technique is to associate a message identifier with messages. The message identifier is an identifier that allows a received message to be correlated with the originating request. The sender may also add an identifier for a service, not necessarily the originating sender, who will be the recipient of the message (see asynchronous messaging).

Correlation may also be realized by the underlying protocol. For example, HTTP/1.1 allows one to correlate a request with its response.

2.3.1.6 Message Envelope

2.3.1.6.1 Definition

A message envelope is the structure that encapsulates the component parts of a message: the message body and the message headers.

2.3.1.6.2 Relationships to other elements

a message envelope may contain

address information about the intended recipients of its associated message

a message envelope contains

the message body.

a message envelope contains

the message headers.

2.3.1.6.3 Explanation

Issue (message_with_address):

How is a message associated with its destination address?

There is an unresolved issue here. A message somehow must be associated with its destination address. This combination of the message with its destination address seems to be a significant architectural concept, yet SOAP does not require that the address be included in the message header.

Resolution:

None recorded.

The message envelope may contain information needed to actually deliver messages. If so, it must at least contain sufficient address information so that the message transport can deliver the message. Typically this information is part of the service *binding* information found in a WSDL document.

Other metadata that may be present in an envelope includes security information to allow the message to be authenticated and quality of service information.

A correctly design message transport mechanism should be able to deliver a message based purely on the information in the envelope. For example, an encrypted message that fully protects the identities of the sender, recipient as well as the message content, may still be delivered using only the address information (and the encrypted data stream itself).

2.3.1.7 Message Exchange Pattern (MEP)

2.3.1.7.1 Definition

A Message Exchanage Pattern (MEP) is a template, devoid of application semantics, that describes a generic pattern for the exchange of messages between agents. It describes relationships (e.g., temporal, causal, sequential, etc.) of multiple messages exchanged in conformance with the pattern, as well as the normal and abnormal termination of any message exchange conforming to the pattern.

2.3.1.7.2 Relationships to other elements

a message exchange pattern describes

a generic pattern for the exchange of messages between agents.

a message exchange pattern should have

a unique identifier

a message exchange pattern may realize

message correlation

a message exchange pattern may describe

a service invocation

2.3.1.7.3 Explanation

Distributed applications in a Web services architecture communicate via message exchanges. These message exchanges are logically factored into patterns that may be composed at different levels to form larger patterns. A Message Exchange Pattern (MEP) is a template, devoid of application semantics, that describes a generic pattern for the exchange of (one-way) messages between agents. The patterns can be described by state machines that define the flow of the messages, including the handling of faults that may arise, and the correlation of messages.

> **Issue (mep_vs_chor):**
>
> **What is the difference between an MEP and a Choreography?**
>
> The precise difference between an MEP and a choreography is unresolved. Some view MEPs as being atomic patterns, and a choreography as including composition of patterns. Also, a choreography generally describes patterns that include application semantics (choreography = MEPs + application semantics), whereas an MEP is devoid of application semantics. Finally, there is usually a difference in scale between an MEP and a choregraphy: A choreography often makes use of MEPs as building blocks.
>
> **Resolution:**
>
> None recorded.

Messages that are instances of an MEP are correlated, either explicitly or implicitly. The exchanges may be synchronous or asynchronous.

In order to promote interoperability, it is useful to define common MEPs that are broadly adopted and unambiguously identified. When a MEP is described for the purpose of interoperability, it should be associated with a URI that will identify that MEP.

Some protocols may natively support certain MEPs, e.g., HTTP natively supports request-response. In other cases there is may be additional glue needed to map MEPs onto a protocol.

Web service description languages at the level of WSDL view MEPs from the perspective of a particular service actor. A simple request-reponse MEP, for example, appears as an incoming message which invokes an operation and an associated outgoing message with a reply.

An MEP is not necessarily limited to capturing only the inputs and outputs of a single service. Consider the pattern:

1. agent A uses an instance of an MEP (possibly request-response) to communicate initially with B.
2. agent B then uses a separate, but related instance of an MEP to communicate with C.
3. agent A uses another instance of an MEP to communicate with C but gets a reply only after C has processed (2).

This example makes it clear that the overall pattern cannot be described in terms of the inputs and outputs of any single interaction. The pattern involves constraints and relationships among the messages in the various MEP instances. It also illuminates the fact that exchange (1) is in in-out MEP from the perspective of actor B, and mirrored by an out-in MEP from the perspective of actor A. Finally, an actual application instantiates this communication pattern and completes the picture by adding computation at A, B and C to carry out application-specific operations.

It is instructive to consider the kinds of fault reporting that occur in such a layering. Consider a fault at the transport protocol level. This transport level may itself be able to manage certain faults (e.g., re-tries), but it may also simply report the fault to the binding level. Similarly the binding level may manage the fault (e.g., by re-initiating the underlying protocol) or may report a SOAP fault. The choreography and application layers may be intertwined or separated depending on how they are architected. There is also no rigid distinction between the choreography and binding layers; binding-level MEPs are essentially simple choreographies. Conceptually, the choreographic level can enforce constraints on message order, maintain state consistency, communicate choreographic faults to the application, etc. in ways that transcend particular bindings and transports.

2.3.1.8 Message Header

2.3.1.8.1 Definition

A message header is the part of the message that contains information about a specific aspect of the message.

2.3.1.8.2 Relationships to other elements

a message header is contained in

a message envelope

a message header may be

a specific well known types

Editorial note	
The "is-a" relationship here is used in a different way than elsewhere in the document.	

a message header may identify

a service role, which denotes the kind of processing expected for the header.

a message header may be processed

independently of the message body

2.3.1.8.3 Explanation

Message headers represent information about messages that is independently standardized (such as WS-Security) — and may have separate semantics -- from the message body. For example, there may be standard forms of message header that describe authentication of messages. The form of such headers is defined for all messages; although, of course, a given authentication header will be specific to the particular message.

The primary function of headers is to facilitate the modular processing of the message, although they can also be used to support routing and related aspects of message processing. The header part of a message can include information pertinent to extended Web services functionality, such as security, transaction context, orchestration information, message routing information, or management information.

Message headers may be processed independently of the message body, each message header may have an identifying service role that indicates the kind of processing that should be performed on messages with that header. Each message may have several headers, each potentially identifying a different service role.

Although many headers will relate to infrastructure facilities, such as security, routing, load balancing and so on; it is also possible that headers will be application specific. For example, a purchase order processing Web service may be structured into layers; corresponding to different functions within the organization. These stakeholders may process headers of different messages in standardized ways: the customer information may be captured in one standardized header, the stock items by a different standardized header and so on.

2.3.1.9 Message Receiver

2.3.1.9.1 Definition

A message receiver is an agent that receives a message.

2.3.1.9.2 Relationships to other elements

a message receiver is

a agent

a message receiver is

the recipient of a message

2.3.1.9.3 Explanation

The message receiver is an agent that is intended to receive a message from the message sender.

Messages may be passed through intermediaries that process aspects of the message, typically by examining the message headers. The message recipient may or may not be aware of processing by such intermediaries.

Often a specific message receiver, the ultimate recipient, is identified as the final recipient of a message. The ultimate recipient will be responsible for completing the processing of the message.

2.3.1.10 Message Reliability

2.3.1.10.1 Definition

Message reliability is the degree of certainty that a message will be delivered and that sender and receiver will both have the same understanding of the delivery status.

2.3.1.10.2 Relationships to other elements

message reliability is

a property of message delivery.

message reliability may be realized by

a combination of message acknowledgement and correlation.

message reliability may be realized by

a transport mechanism

2.3.1.10.3 Explanation

The goal of reliable messaging is to both reduce the error frequency for messaging and to provide sufficient information about the status of a message delivery. Such information enables a participating agent to make a compensating decision when errors or less than desired results occur. High level correlation such as "two-phase commit" is needed if more than two agents are involved. Note that in a distributed system, it is theoretically not possible to guarantee correct notification of delivery; however, in practice, simple techniques can greatly increase the overall confidence in the message delivery.

It is important to note that a guarantee of the delivery of messages alone may not improve the overall reliability of a Web service due to the need for end-to-end reliability. (See "End-to-End Arguments in System Design".) It may, however, reduce the overall cost of a Web service.

Message reliability may be realized with a combination of message receipt acknowledgement and correlation. In the event that a message has not been properly received and acted upon, the sender may attempt a resend, or some other compensating action at the application level.

2.3.1.11 Message Sender

2.3.1.11.1 Definition

A message sender is the agent that transmits a message.

2.3.1.11.2 Relationships to other elements

a message sender is

an agent

a message sender is

the originator of a message

2.3.1.11.3 Explanation

A message sender is an agent that transmits a message to another agent. Although every message has a sender, the identity of the sender may not be available to others in the case of anonymous

interactions.

Messages may also be passed through intermediaries that process aspects of the message; typically by examining the message headers. The sending agent may or may not be aware of such intermediaries.

2.3.1.12 Message Sequence

2.3.1.12.1 Definition

A message sequence is a sequence of related messages.

2.3.1.12.2 Relationships to other elements

a message sequence is

a sequence of related messages

a message sequence may realize

a documented message exchange pattern

2.3.1.12.3 Explanation

A requester agent and a provider agent exchange a number of messages during an interaction. The ordered set of messages exchanged is a message sequence.

This sequence may be realizing a well-defined MEP, usually identified by a URI.

2.3.1.13 Message Transport

2.3.1.13.1 Definition

A Message Transport is a mechanism that may be used by agents to deliver messages.

2.3.1.13.2 Relationships to other elements

a message transport is

a mechanism that delivers messages

a message transport has

zero or more capabilities

a message transport is constrained by

various delivery policies

a message transport must know

sufficient address information in order to deliver a message.

2.3.1.13.3 Explanation

The message transport is the actual mechanism used to deliver messages. Examples of message transport include HTTP over TCP, SMTP, message oriented middleware, and so on.

The responsibility of the message transport is to deliver a message from a sender to one or more recipient, i.e. transport a SOAP Infoset from one agent to another, possibly with some implied semantics

(e.g. HTTP methods semantics).

Message transports may provide different features (e.g. message integrity, quality of service guaranties, etc.).

For a message transport to function, the sending agent must provide the address of the recipient.

2.3.2 The Service Oriented Model

The Service Oriented Model (SOM) focuses on those aspects of the architecture that relate to service and action.

The primary purpose of the SOM is to explicate the relationships between an agent and the services it provides and requests. The SOM builds on the MOM, but its focus is on action rather than message.

The concepts and relationships in the SOM are illustrated in Figure 2-8:

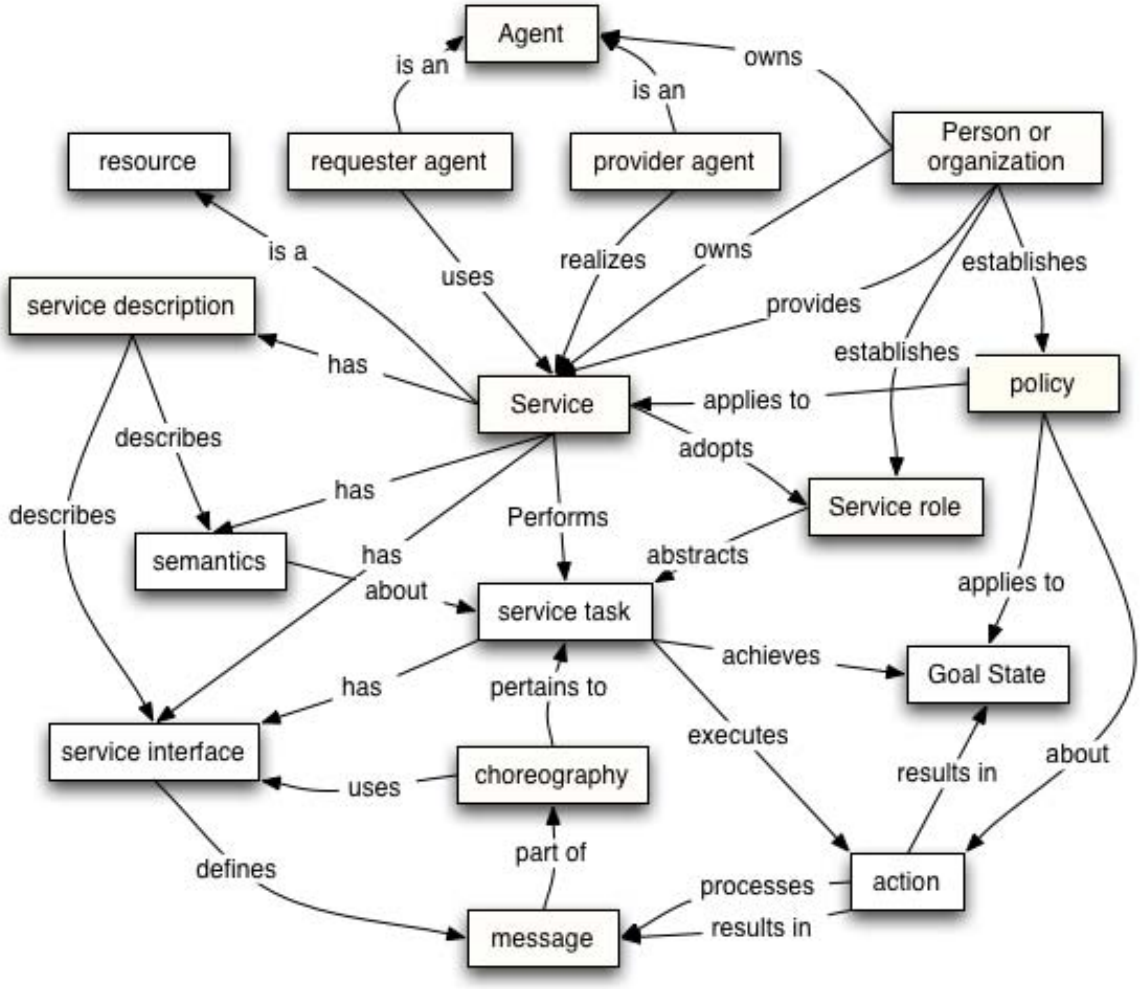


Figure 2-8. Service Oriented Model

2.3.2.1 Action

2.3.2.1.1 Definition

An action, for the purposes of this architecture, is any action that may be performed by an agent,

possibly as a result of receiving a message, or which results in sending a message or another observable state change.

2.3.2.1.2 Relationships to other elements

An action may result in

a desired goal state

An action may be

the sending of a message

An action may be

the processing of a received message

2.3.2.1.3 Explanation

At the core of the concept of service is the notion of one party performing action(s) at the behest of another party. From the perspective of requester and provider agents, an action is typically performed by executing some fragment of a program.

In the WSA, the actions performed by requester and provider agents are largely out of scope, except in so far as they are the result of messages being sent or received. In effect, the programs that are executed by agents are not in scope of the architecture, however the resulting messages are in scope.

2.3.2.2 Agent

2.3.2.2.1 Definition

An agent is a program acting on behalf of person or organization. (This definition is a specialization of the definition in [Web Arch]. It corresponds to the notion of *software agent* in [Web Arch].)

2.3.2.2.2 Relationships to other elements

An agent is

a computational resource

An agent has

an owner that is a person or organization

An agent may realize

zero or more services

An agent may request

zero or more services

2.3.2.2.3 Explanation

Agents are programs that engage in actions on behalf of someone or something else. For our purposes, agents realize and request Web services. In effect, software agents are the running programs that *drive* Web services — both to implement them and to access them.

Software agents are also *proxies* for the entities that own them. This is important as many services

involve the use of resources which also have owners with a definite interest in their disposition. For example, services may involve the transfer of money and the incurring of legal obligations as a result.

We specifically avoid any attempt to govern the implementation of agents; we are only concerned with ensuring interopability between systems.

2.3.2.3 Choreography

2.3.2.3.1 Definition

A choreography defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state.

Editorial note	
This is a different level of abstraction from the definition used by the W3C Web Services Choreography Working Group.	

2.3.2.3.2 Relationships to other elements

A choreography uses

one or more service interfaces .

A choreography defines

the pattern of possible interactions between a set of agents .

A choreography may be expressed in

a choreography description language

A choreography pertains to

a given task

A choreography defines

the relationship between exchanged messages and tasks of a service.

2.3.2.3.3 Explanation

A choreography is a model of the sequence of operations, states, and conditions that control the interactions involved in the participating services. The interaction prescribed by a choreography results in the completion of some useful function. Examples include the placement of an order, information about its delivery and eventual payment, or putting the system into a well-defined error state.

A choreography can be distinguished from an orchestration. An orchestration defines the sequence and conditions in which *one* Web service invokes other Web services in order to realize some useful function.

A choreography may be described using a choreography description language. A choreography description language permits the description of how Web services can be composed, how service roles and associations in Web services can be established, and how the state, if any, of composed services is to be managed.

2.3.2.4 Capability

2.3.2.4.1 Definition

A capability is a named piece of functionality (or feature) that is declared as supported or requested by an agent.

2.3.2.4.2 Relationships to other elements

a capability has a

identifier which is a URI

a capability has a

a description of its semantics

a capability can be

advertised by an agent that supports it

a capability can be

required by agent that wishes to use it

a capability may be referenced by

a service description

2.3.2.4.3 Explanation

Agents participating in an exchange may implement a wide variety of features. For example, there may be different ways to achieve the reliable delivery of a message, or there may be several mechanisms available to support security. A Web service may advertise that it supports a particular capability, and an agent requiring that capability might select the service on that basis. Or a Web service may indicate that it requires a particular capability of any requester agent that uses it, and a requester agent may select it or avoid it on that basis. There may also be a negotiation -- either manual or automatic -- about which capabilities to select.

The concept of capability encompasses SOAP features, but is broader.

2.3.2.5 Goal State

2.3.2.5.1 Definition

A goal state is a state of some service or resource that is desireable from some person or organization's point of view.

2.3.2.5.2 Relationships to other elements

a goal state is

a state of the real world, which includes the state of relevant resources

a goal state is

desired by some person or organization which has an interest in defining it.

a goal state may be characterized

informally, or formally with a formal expression.

2.3.2.5.3 Explanation

Goal states are associated with tasks. Tasks are the unit of action associated with services that have a

measurable meaning. Typically measured from the perspective of the owner of a service, a goal state is characterized by a predicate that is true of that state — for example, a book selling service may have as its goal state that a book has been purchased by a legitimate customer.

It is difficult to be formal about vague properties such as desireable, however, it is also clear that services are deployed and used with an intention. An e-commerce service is oriented towards buying and selling, a stock ticker service is oriented towards giving timely information. A goal state is simply a way of being able to declare success when a task has completed sucessfully.

2.3.2.6 Provider Agent

2.3.2.6.1 Definition

A provider agent is an agent that is capable of and empowered to perform the actions associated with a service on behalf of its owner — the provider entity.

2.3.2.6.2 Relationships to other elements

a provider agent is

a Web service software agent

a provider agent realizes

one or more services

a provider agent performs, or causes to perform

the actions associated with a task

a provider agent acts on behalf of

a provider entity

2.3.2.6.3 Explanation

The provider agent is the software agent that realizes a Web service by performing tasks on behalf of its owner — the provider entity.

A given service may be offered by more than one agent, especially in the case of composite services, and a given provider agent may realize more than one Web service.

2.3.2.7 Provider Entity

2.3.2.7.1 Definition

The provider entity is the person or organization that is providing a Web service.

2.3.2.7.2 Relationships to other elements

a provider entity

is a person or organization

a provider entity

offers a Web service

a provider entity owns

a provider agent

2.3.2.7.3 Explanation

The provider entity is the person or organization that is offering a Web service. The provider agent acts on behalf of the provider entity that owns it.

2.3.2.8 Requester Agent

2.3.2.8.1 Definition

A requester agent is a software agent that wishes to interact with a provider agent in order to request that a task be performed on behalf of its owner — the requester entity.

2.3.2.8.2 Relationships to other elements

a requester agent is

an agent

a requester agent uses

a service

a requester agent acts on behalf of

a requester entity

2.3.2.8.3 Explanation

The requester agent is the software agent that requires a certain function to be performed on behalf of its owner — the requester entity. From an architectural perspective, this is the agent that is looking for and invoking or initiating an interaction with a provider agent.

2.3.2.9 Requester Entity

2.3.2.9.1 Definition

The requester entity is the person or organization that wishes to use a provider entity's Web service.

2.3.2.9.2 Relationships to other elements

a requester entity

is a person or organization

a requester entity owns

a requester agent

2.3.2.9.3 Explanation

The requester entity is the person or organization that wishes to make use of a Web service. The requester entity is the counterpart to the provider entity.

2.3.2.10 Service

2.3.2.10.1 Definition

A service is an abstract resource that represents a capability of performing tasks that represents a coherent functionality from the point of view of provider entities and requester entities. To be used, a service must be realized by a concrete provider agent.

2.3.2.10.2 Relationships to other elements

a service is a

resource

a service performs

one or more tasks

a service has

a service description

a service has a

service interface

a service has

service semantics

a service has

an identifier

a service has

a service semantics

a service has

one or more service roles in relation to the service's owner

a service may have

one or more policies applied to it.

a service is owned by

a person or organization.

a service is provided by

a person or organization.

a service is realized by

a provider agent.

a service is used by

a requester agent.

2.3.2.10.3 Explanation

A service is an abstract resource that represents a person or organization in some collection of related tasks as having specific service roles. The service may be realized by one or more provider agents that act on behalf of the person or organization — the provider entity.

The critical distinction of a Web service, compared to other Web resources, is that Web services do not necessarily have a representation; however, they *are* associated with actions.

Issue (ws_get):

What should be the representation returned by an HTTP "GET" on a Web service URI?

What should be the representation of a Web service? Should a service description be available at the service URI?

Resolution:

None recorded.

For a Web service to be compliant with this architecture there must be sufficient service descriptions associated with the service to enable its use by other parties. Ideally, a service description will give sufficient information so that an automatic agent may not only use the service but also decide if the service is appropriate; that in turn implies a description of the semantics of the service.

We distinguish a number of things in their relation to a service: a service has an owner; a service must be realized by a (software) provider agent; a requester agent may interact with a provider agent; and a provider agent has an owner (the provider entity). Web services are inherently *about* computer-to-computer interactions between requester and provider agents; yet they are also ultimately deployed in human service because the requester and provider agents act on behalf of their owners.

Web services are focused on actions. It is convenient, for the purposes of characterizing their semantics, to capture this in terms of tasks. The semantics of any computational system is bound with the behavior of the system: and the intended semantics is bound with some desired behavior. Tasks combine the concept of action with intention: i.e., Web services are conventionally invoked with a given purpose in mind. The purpose can be expressed as an intended goal state: such as a book being delivered or a temperature reading being taken.

There is *no* requirement for there to be a one-to-one correspondence between messages and services. A given message may be processed by more than one service, especially in the situation where there are service intermediaries, and a given service may, of course, process more than one kind of message. We formalize this by asserting that a service adopts one or more service roles. The service role identifies the intended role as determined by the owner of the service. A given role is characterized by the aspects of messages it is concerned with.

2.3.2.11 Service Description

2.3.2.11.1 Definition

A service description is a set of documents that describe the interface to and semantics of a service.

2.3.2.11.2 Relationships to other elements

a service description is

a machine-processable description of a service

a service description is

a machine-processable description of the service's interface

a service description contains

a machine-processable description of the messages that are exchanged by the service

a service description may include

a description of the service's semantics

a service description is expressed in

a service description language

2.3.2.11.3 Explanation

A service description contains the details of the interface and, potentially, the expected behavior of the service. This includes its data types, operations, transport protocol information, and address. It could also include categorization and other metadata to facilitate discovery and utilization. The complete description may be realized as a set of XML description documents.

There are many potential uses of service descriptions: they may be used to facilitate the construction and deployment of services, they may be used by people to locate appropriate services, and they may be used by requester agents to automatically discover appropriate provider agents in those case where requester agents are able to make suitable choices.

2.3.2.12 Service Interface

2.3.2.12.1 Definition

A service interface is the abstract boundary that a service exposes. It defines the types of messages and the message exchange patterns that are involved in interacting with the service, together with any conditions implied by those messages.

2.3.2.12.2 Relationships to other elements

a service interface defines

the messages relevant to the service

2.3.2.12.3 Explanation

A service interface defines the different types of messages that a service sends and receives, along with the message exchange patterns that may be used.

2.3.2.13 Service Intermediary

2.3.2.13.1 Definition

A service intermediary is a Web service whose main role is to transform messages in a value-added way. (From a messaging point of view, an intermediary processes messages en route from one agent to another.) Specifically, we say that a service intermediary is a service whose outgoing messages are equivalent to its incoming messages in some application-defined sense.

2.3.2.13.2 Relationships to other elements

A service intermediary is

a service.

A service intermediary adopts

a specific service role.

A service intermediary preserves

the semantics of messages it receives and sends.

2.3.2.13.3 Explanation

A service intermediary is a specific kind of service which typically acts as a kind of filter on messages it handles. Normally, intermediaries do not consume messages but rather forward them to other services. Of course, intermediaries do often *modify* messages but, it is of the essence that *from some application specific perspective* they do not modify the meaning of the message.

Of course, if a message is altered in any way, then from some perspectives it is no longer the same message. However, just as a paper document is altered whenever anyone writes a comment on the document, and yet it is still the same document, so an intermediary modifies the messages that it receives, forwarding the same message with some changes.

Coupled with the concept of service intermediary is the service role is adopts. Typically, this involves one or more of the messages' headers rather than the bodies of messages. The specification of the header is coupled with the permissable semantics of the intermediary should make it clear to what extent the messages forwarded by an itnermediary are the same message and what modifications are permitted.

There are a number of situations where additional processing of messages is required. For example, messages that are exchanged between agents within an enterprise may not need encryption; however, if a message has to leave the enterprise then good security may suggest that it be encrypted. Rather than burden every software agent with the means of encrypting and decrypting messages, this functionality can be realized by means of an intermediary. The main responsiblity of the software agents then becomes ensuring that the messages are routed appropriately and have the right headers targetted at the appropriate intermediaries.

2.3.2.14 Service Role

2.3.2.14.1 Definition

A service role is an abstract set of tasks which is identified to be relevant by a person or organization offering a service. Service roles are also associated with particular aspects of messages exchanged with a service.

2.3.2.14.2 Relationships to other elements

a service role is

a set of service tasks

a service role may be defined

in terms of particular properties of messages.

a service role may be established by

a service owner.

2.3.2.14.3 Explanation

A service role is an intermediate abstraction between service and task. A given message that is received by a service may involve processing associated with several service roles. Similarly, messages emitted may also involve more than one service role.

We can formalize the distinction by noting that a service role is typically associated with a particular property of messages. For *ultimate* processing, the service role may be to determine some final disposition of messages received. However, other service roles may be associated with more generic properties of messages — such as their encryption, or whether they reference a customer or inventory item.

Service roles identify the points of interest that a service owner has in the processing of messages. As such, they are established by the party that offers in the service.

2.3.2.15 Service Semantics

2.3.2.15.1 Definition

The semantics of a service is the behavior expected when interacting with the service. The semantics expresses a contract (not necessarily a legal contract) between the provider entity and the requester entity. It expresses the intended real-world effect of invoking the service. A service semantics may be formally described in a machine readable form, identified but not formally defined, or informally defined via an "out of band" agreement between the provider entity and the requester entity.

2.3.2.15.2 Relationships to other elements

a service semantics is

the contract between the provider entity and the requester entity concerning the effects and requirements pertaining to the use of a service

a service semantics describes

the intended effects of using a service

a service semantics is about

the service tasks that constitute the service.

a service semantics should be identified

in a service description

a service semantics may be described

in a formal, machine-processable language

2.3.2.15.3 Explanation

Knowing the type of a data structure is not enough to understand the intent and meaning behind its use. For example, methods to deposit and withdraw from an account typically have the same type signature, but with a different effect. The effects of the operations are the semantics of the operation. It is good practice to be *explicit* about the intended effects of using a Web service; perhaps even to the point of constructing a machine readable description of the semantics of a service.

Machine processable semantic descriptions provide the potential for sophisticated usage of Web services. For example, by accessing such descriptions, a requester agent may autonomously choose which provider agent to use.

Apart from the expected behavior of a service, other semantic aspects of a service include any policy restrictions on the service, the relationship between the provider entity and the requester entity, and what manageability features are associated with the service.

2.3.2.16 Service Task

2.3.2.16.1 Definition

A service task is an action or combination of actions that is associated with a desired goal state. Performing the task involves executing the actions, and is intended to achieve a particular goal state.

2.3.2.16.2 Relationships to other elements

a service task is

an action or combination of actions.

a service task is associated with

one or more intended goal states.

a service task is performed by

executing the actions associated with the task.

a service task has a

service interface

2.3.2.16.3 Explanation

A service task is an abstraction that encapsulates some intended effect of invoking a service.

Tasks are associated with goal states — characterized by predicates that are satisfied on successful completion.

The performance of a task is made observable by the exchange of messages between the requester agent and the provider agent. The specific pattern of messages is what defines the choreography associated with the task.

In addition to exchanged messages, there may be other private actions associated with a task. For example, in a database update task, the task may be signaled by an initiating message and a completion message, which are public, and the actual database update, which is typically private.

In the case of a service oriented architecture only the public aspects of a task are important, and these are expressed entirely in terms of the messages exchanged.

Tasks represent a useful unit in modeling the semantics of a service and indeed of a service role — a given service may consist of a number of tasks.

2.3.3 The Resource Oriented Model

The Resource Oriented Model focuses on those aspects of the architecture that relate to resources. Resources are a fundamental concept that underpins much of the Web and much of Web services; for example, a Web service is a particular kind of resource that is important to this architecture.

The ROM focuses on the key features of resources that are relevant to the concept of resource, independent of the role the resource has in the context of Web services. Thus we focus on issues such as the ownership of resources, policies associated with resources and so on. Then, by virtue of the fact that Web services are resources, these properties are inherited by Web services.

We illustrate the basic concepts and relationships in the ROM in Figure 2-9:

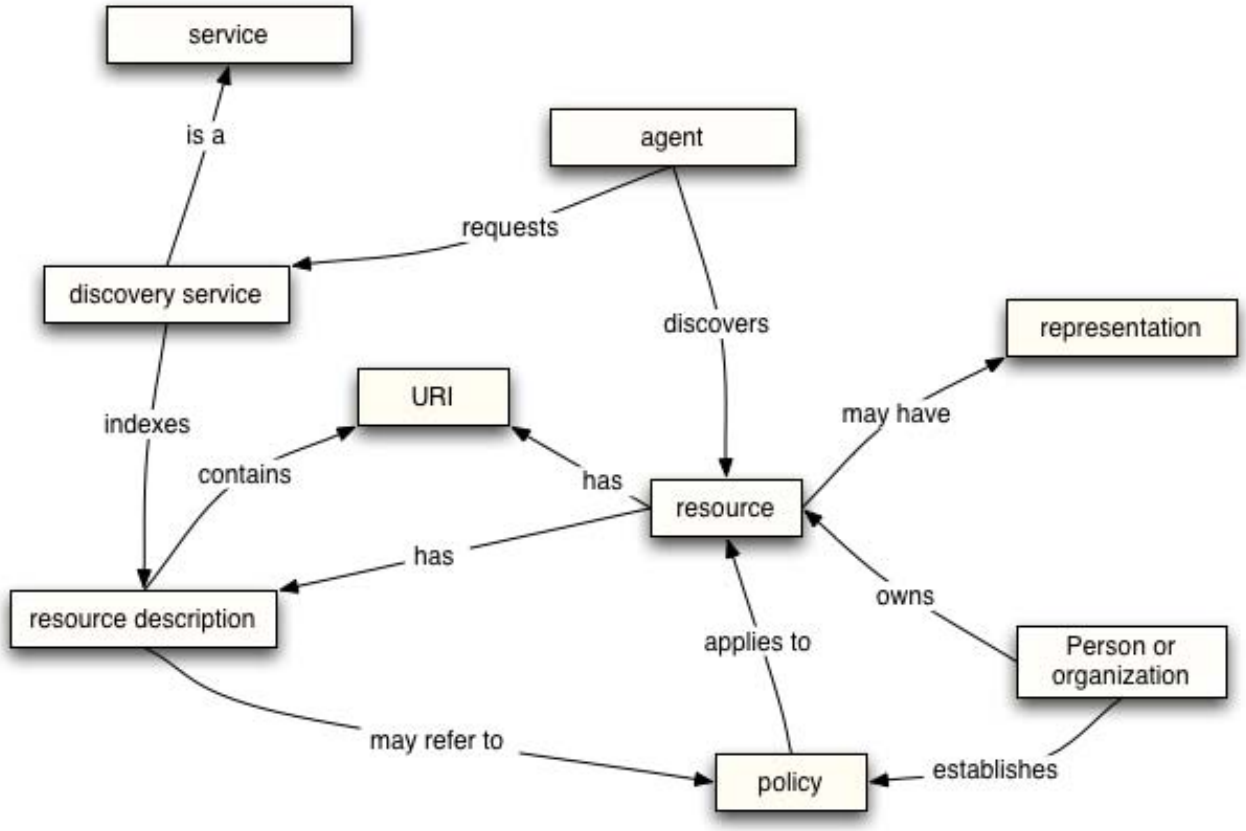


Figure 2-9. Resource Oriented Model

2.3.3.1 Discovery

2.3.3.1.1 Definition

Discovery is the act of locating a machine-processable description of a Web service-related resource that may have been previously unknown and that meets certain functional criteria. It involves matching a set of functional and other criteria with a set of resource descriptions. The goal is to find an appropriate Web service-related resource.[WS Glossary]

2.3.3.1.2 Relationships to other elements

Discovery is

the act of locating a resource description

Discovery involves

matching a set of functional and other criteria with a set of resource descriptions.

Discovery may be performed

by an agent, or by an end-user

Discovery may be realized

using a discovery service

2.3.3.1.3 Explanation

In the context of Web services, the resources being discovered are usually service descriptions. If a requester entity does not already know what service it wishes to engage, the requester entity must

discover one. There are various means by which discovery can be performed. Various things — human end users or agents — may initiate discovery. Requester entities may find service descriptions during development for static binding, or during execution for dynamic binding. For statically bound requester agents, using discovery is optional, as the service description might be obtained in other ways, such as being sent directly from the provider entity to the requester entity, developed collaboratively, or provided by a third party, such as a standards body.

2.3.3.2 Discovery Service

2.3.3.2.1 Definition

A discovery service is a service that enables agents to retrieve Web service-related resource descriptions.

2.3.3.2.2 Relationships to other elements

A discovery service is

a service

A discovery service is used to

publish descriptions

A discovery service is used to

search for resource descriptions

A discovery service may be used

by an agent

2.3.3.2.3 Explanation

A discovery service is used to publish and search for descriptions meeting certain functional or semantic criteria. It is primarily intended for use by requester entities, to facilitate the process of finding an appropiate provider agent for a particular task. However, depending on the implementation and policy of the discovery service (**3.4.2 Discovery: Registry, Index or Peer-to-Peer?**), it may also be used by provider entities to actively publish their service descriptions.

Although the resource model is general purpose, the most important resource for our purposes is the Web service. Furthermore, the primary role of a discovery service is to facilitate the discovery of Web services.

For dynamic discovery, the requester agent may interact directly with the discovery service to find an appropriate provider agent to engage. For static discovery, a human may interact with the discovery service through an appropriate software agent, such as a browser.

The use of an automated discovery service is optional, since other means can be used to enable a requester entity and provider entity to agree on the service description that will govern the interaction. For example, the requester entity might obtain the service description directly from the provider entity, the two parties might develop the service description collaboratively, or, in some circumstances, the service description may be created by the *requester* entity and dictated to the provider entity. (For example, a large company may require its suppliers to provide Web services that conform to a particular service description.) Likewise, a requester entity can obtain a service description from other sources besides a discovery service, such as a local file system, FTP site, URL, or WSIL document.

2.3.3.3 Identifier

2.3.3.3.1 Definition

An identifier is an unambiguous name for a resource.

2.3.3.3.2 Relationships to other elements

an identifier should be realized

a URI

an identifier identifies

a resource that is relevant to the architecture

2.3.3.3.3 Explanation

Identifiers are used to identify resources. In the architecture we use Uniform Resource Identifiers [RFC 2396] to identify resources.

Issue (urivsqname):

Should URIs be used to identify Web services components, rather than QNames?

Some specifications use QNames to identify things. However, QNames may be ambiguous, because the same QName may be used to identify things of different types. (In effect, specifications having this practice have different symbol spaces to distinguish the different uses of a QName.) Should URIs be preferred instead of QNames for Web services? A significant majority of this Working Group believes the answer is yes.

Resolution:

None recorded.

2.3.3.4 Representation

2.3.3.4.1 Definition

A representation is a piece of data that describes a resource state.

2.3.3.4.2 Relationships to other elements

a resource may have a

representation

2.3.3.4.3 Explanation

Representations are data objects that reflect the state of a resource. A resource has a unique identifier (a URI). Note that a representation of a resource need not be the same as the resource itself; for example the resource asociated with the booking state of a restaurant will have different representations depending on when the representation is retrieved. A representation is usually retrieved by performing an HTTP "GET" on a URI.

2.3.3.5 Resource

2.3.3.5.1 Definition

A resource is defined by [RFC 2396] to be anything that can have an identifier. Although resources in general can be anything, this architecture is only concerned with those resources that are relevant to Web services and therefore have some additional characteristics. In particular, they incorporate the concepts of ownership and control: a resource that appears in this architecture is a *thing* that has a name, may have reasonable representations and which can be said to be owned. The ownership of a resource is critically connected with the right to set policy on the resource.

2.3.3.5.2 Relationships to other elements

a resource has

an identifier

a resource may have

zero or more representations

a resource may have

zero or more resource descriptions

a resource is owned by

a person or organization

a resource may be governed by

zero or more policies

2.3.3.5.3 Explanation

Resources form the heart of the Web architecture itself. The Web is a universe of resources that have URIs as identifiers, as defined in [RFC 2396].

From a real-world perspective, a most interesting aspect of a resource is its ownership: a resource is something that can be owned, and therefore have policies applied to it. Policies applying to resources are relevant to the management of Web services, security of access to Web services and many other aspects of the role that a resource has in the world.

2.3.3.6 Resource description

2.3.3.6.1 Definition

A resource description is any machine readable data that may permit resources to be discovered. Resource descriptions may be of many different forms, tailored for specific purposes, but all resource descriptions must contain the resource's identifier.

2.3.3.6.2 Relationships to other elements

A resource description contains

the resource's identifier

A resource description may reference

the policies applicable to the resource

A resource description may reference

the semantics applicable to the resource

2.3.3.6.3 Explanation

A resource description is a machine-processable description of a resource. Resource descriptions are used by and within discovery services to permit agents to discover the resource.

The precise contents of a resource description will vary, depending on the resource, on the purpose of the description and on the accessibility of the resource. However, for our purposes it is important to note that the description must contain the resource's identifier. I.e., a description of the form: "the new resource that is owned by XYZ co." is not regarded as a valid resource description because it does not mention the resource's identifier.

A primary purpose of resource descriptions is to facilitate the discovery of the resource. To aid that purpose, the description is likely to contain information about the location of the resource, how to access it and potentially any policies that govern the policy. Where the resource is a Web service, the description may also contain machine-processable information about how to invoke the Web service and the expected effect of using the Web service.

Note that a resource description is fundamentally distinct from the resource representation. The latter is a snapshot reflecting the state of resource, the description is meta-level information about the resource.

2.3.4 The Policy Model

The Policy Model focuses on those aspects of the architecture that relate to policies and, by extension, security and quality of service.

Security is fundamentally about constraints; about constraints on the behavior on action and on accessing resources. Similarly, quality of service is also about constraints on service. In the PM, these constraints are modeled around the core concept of policy; and the relationships with other elements of the architecture. Thus the PM is a framework in which security can be realized.

However, there are many other kinds of constraints, and policies that are relevant to Web services, including various application-level constraints.

The concepts and relationships in the PM are illustrated in Figure 2-10:

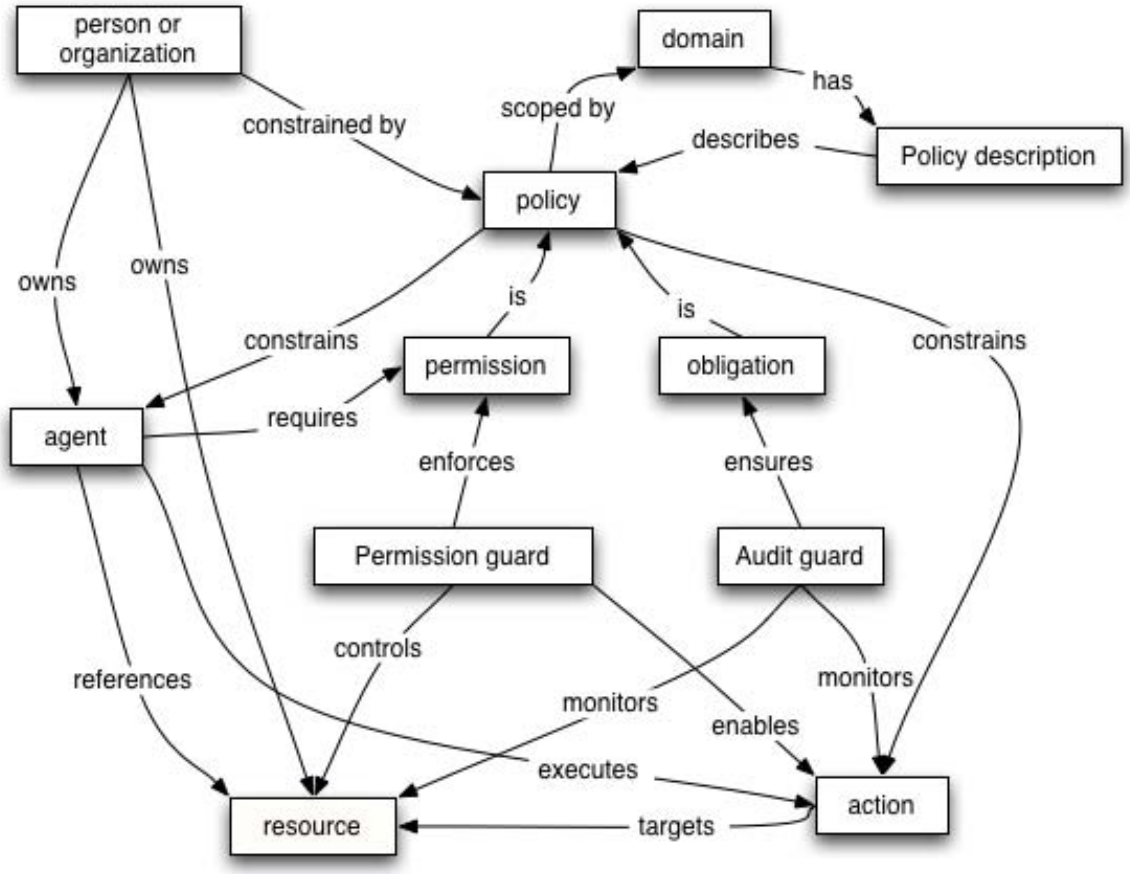


Figure 2-10. Policy Model

2.3.4.1 Audit Guard

2.3.4.1.1 Definition

An audit guard is a mechanism used on behalf of an owner that monitors actions and agents to verify the satisfaction of obligations.

2.3.4.1.2 Relationships to other elements

a audit guard is a

a policy guard

an audit guard may monitor

one or more resources.

an audit guard may monitor

actions relative to one or more services.

an audit guard may determine

if an agent's obligations have been discharged.

2.3.4.1.3 Explanation

An audit guard is an enforcement mechanism. It is used to monitor the discharge of obligations. The role of the audit guard is to monitor that agents, resources and services are consistent with any associated obligations established by the service's owner or manager.

Typically, an audit guard monitors the state of a resource or a service, ensuring that the obligation is satisfied. It determines whether the associated obligations are satisfied.

By their nature, it is not possible to proactively enforce obligations; hence, an obligation violation may result in some kind of retribution after the fact of the violation.

2.3.4.2 Domain

2.3.4.2.1 Definition

A domain is an identified set of agents and/or resources that is subject to the constraints of one of more policies.

2.3.4.2.2 Relationships to other elements

A domain is

a collection of agents and/or resources.

A domain defines

the scope of application of one or more policies

2.3.4.2.3 Explanation

A domain defines the scope of applicability of policies. A domain may be defined explicitly or implicitly. Members of an explicitly defined domain are enumerated by a central authority; members of an implicitly defined domain are not. For example, membership in an implicitly defined domain may depend on the state of the agent or something it possesses, and thus may be dynamic.

2.3.4.3 Obligation

2.3.4.3.1 Definition

An obligation is a kind of policy that prescribes actions and/or states of an agent and/or resource.

2.3.4.3.2 Relationships to other elements

an obligation is a

kind of policy

an obligation may require

an agent to perform one or more actions

an obligation may require

an agent or service to be in one or more allowable states

an obligation may be discharged

by the performance of an action or other event.

2.3.4.3.3 Explanation

An obligation is one of two fundamental types of policies. When an agent has an obligation to perform some action, then it is required to do so. When the action is performed, then the agent can be said to

have satisfied its obligations.

Not all obligations relate to actions. For example, an agent providing a service may have an obligation to maintain a certain state of readiness. (Quality of service policies are often expressed in terms of obligations.) Such an obligation is typically not discharged by any of the obligee's actions; although an event (such as a certain time period expiring) may discharge the obligation.

Obligations, by their nature, cannot be proactively enforced. However, obligations are associated with enforcement mechanisms: audit guards. These monitor controlled resources and agents and may result in some kind of *retribution*; retributions are not modeled by this architecture.

An obligation may continue to exist after its requirements have been met (for example, an obligation to maintain a particular credit card balance), or it may be discharged by some action or event.

2.3.4.4 Permission

2.3.4.4.1 Definition

A permission is a kind of policy that prescribes the allowed actions and states of an agent and/or resource.

2.3.4.4.2 Relationships to other elements

a permission is a

type of policy

a permission may enable

one or more actions

a permission may enable

one or more allowable states

2.3.4.4.3 Explanation

A permission is one of two fundamental types of policies. When an agent has permission to perform some action, to access some resource, or to achieve a certain state, then it is expected that any attempt to perform the action etc., will be successful. Conversely, if an agent does *not* have the required permission, then the action should fail even if it would otherwise have succeeded.

Permissions are enforced by guards, in particular permission guards, whose function is to ensure that permission policies are honored.

2.3.4.5 Permission Guard

2.3.4.5.1 Definition

A permission guard is a mechanism deployed on behalf of an owner to enforce permission policies.

2.3.4.5.2 Relationships to other elements

a permission guard is a

a policy guard

a permission guard is a

a mechanism that enforces permission policies

a permission guard may control

one or more resources.

a permission guard enables

actions relative to one or more services.

2.3.4.5.3 Explanation

A permission guard is an enforcement mechanism that is used to enforce permission policies. The role of the permission guard is to ensure that any uses of a service or resource are consistent with the policies established by the service's owner or manager.

Typically, a permission guard sits between a resource or service and the requester of that resource or service. In many situations, it is not necessary for a service to be aware of the permission guard. For example, one possible role of a message intermediary is to act as a permission guard for the final intended recipient of messages.

A permission guard acts by either enabling a requested action or access, or by denying it. Thus, it is normally possible for permission policies to be proactively enforced.

2.3.4.6 Person or Organization

2.3.4.6.1 Definition

A person or organization may be the owner of agents that provide or request Web services.

2.3.4.6.2 Relationships to other elements

a person or organization may own

an agent

a person or organization may belong to

a domain

a person or organization may establish

policies governing resources that they own

2.3.4.6.3 Explanation

The WSA concept of person or organization is intended to refer to the real-world people that are represented by agents that perform actions on their behalf. All actions considered in this architecture are ultimately rooted in the actions of humans.

2.3.4.7 Policy

2.3.4.7.1 Definition

A policy is a constraint on the behavior of agents or people or organizations.

2.3.4.7.2 Relationships to other elements

a policy is a

constraint on the allowable actions or states of an agent or person or organization

a policy may have

an identifier

a policy may be described

in a policy description

a policy may define

a capability

2.3.4.7.3 Explanation

A policy is a constraint on the behavior of agents as they perform actions or access resources.

There are many kinds of policies, some relate to accessing resources in particular ways, others relate more generally to the allowable actions an agent may perform: both as provider agents and as requester agents.

Logically, we identify two types of policy: permissions and obligations.

Although most policies relate to actions of various kinds, it is not exclusively so. For example, there may be a policy that an agent must be in a certain state (or conversely may not be in a particular state) in relation to the services it is requesting or providing.

Closely associated with policies are the mechanisms for establishing policies and for enforcing them. This architecture does not model the former.

Policies have applications for defining security properties, quality of service properties, management properties and even application properties.

2.3.4.8 Policy Description

2.3.4.8.1 Definition

A policy description is a machine-processable description of a policy or set of policies.

2.3.4.8.2 Relationships to other elements

a policy description describes

a policy

2.3.4.8.3 Explanation

A policy description is a machine processable description of some constraint on the behavior of agents as they perform actions, access resources.

The policy description itself is not the policy, but it may define the policy and it may be used to determine if the policy *applies* in a given situation.

Policy descriptions may include specific conditions, such as "agents of XXX Co. may access files in directory FFF". They may also include more general rules, such as "if an entity has the right to access files in the directory FFF, it also has the obligation to close them after 20 seconds.".

2.3.4.9 Policy Guard

2.3.4.9.1 Definition

A policy guard is a mechanism that enforces one or more policies. It is deployed on behalf of an owner.

2.3.4.9.2 Relationships to other elements

a policy guard has

an owner responsible for establishing the guard

2.3.4.9.3 Explanation

A policy guard is an abstraction that denotes a mechanism that is used by owners of resources to enforce policies.

The architecture identifies two kinds of policy guards: audit guards and permission guards. These relate to the core kinds of policies (obligation and permission policies respectively).

2.4 Relationships

This section defines terms that appear in our architectural models but are not specific to Web services or Web services architecture. However, they are defined here to help clarify our use of these terms in this document.

2.4.1 The *is a* relationship

2.4.1.1 Definition

The *X* is a *Y* relationship denotes the relationship between concepts *X* and *Y*, such that every *X* is also a *Y*.

2.4.1.2 Relationships to other elements

Assuming that *X* is a *Y*, then:

true of

if *P* is true of *Y* then *P* is true of *X*

contains

if *Y* has a *P* then *X* has a *Q* such that *Q* is a *P*.

transitive

if *P* is true of *Y* then *P* is true of *X*

2.4.1.3 Explanation

Essentially, when we say that concept *X* is a *Y* we mean that every feature of *Y* is also a feature of *X*. Note, however, that since *X* is presumably a more specific concept than *Y*, the features of *X* may also be more specific variants of the features of *Y*.

For example, in the service concept, we state that every service has an identifier. In the more specific Web service concept, we note that a Web service has an identifier in the form of a URI identifier.

2.4.2 The *describes* relationship

2.4.2.1 Definition

The concept *Y* describes *X* if and only if *Y* is an expression of some language *L* and that the values of *Y* are instances of *X*.

2.4.2.2 Relationships to other elements

Assuming that *Y* describes *X*, then: if *Y* is a valid expression of *L*, then the values of *Y* are instances of concept *X*

2.4.2.3 Explanation

Essentially, when we say that *Y* describes concept *X* we are saying that the expression *Y* denotes instances of *X*.

For example, in the service description concept, we state that service descriptions are expressed in a service description language. That means that we can expect legal expressions of the service description language to be instances of the service description concept.

2.4.3 The *has a* relationship

2.4.3.1 Definition

Saying that "the concept *X* has a *Y* relationship" denotes that every instance of *X* is associated with an instance of *Y*.

2.4.3.2 Relationships to other elements

Assuming that *X* has a *Y*, then: if *E* is an instance of *X* then *Y* is valid for *E*.

2.4.3.3 Explanation

When we say that "concept *X* has a *Y*" we mean that whenever we see an *X* we should also see a *Y*

For example, in the Web service concept, we state that Web services have URI identifiers. So, whenever the Web service concept is found, we can assume that the Web service referenced has an identifier. This, in turn, allows implementations to use identifiers to reliably refer to Web services. If a given Web service does not have an identifier associated with it, then the architecture has been violated.

2.4.4 The *owns* relationship

2.4.4.1 Definition

The relationship "*X* owns *Y*" denotes the relationship between concepts *X* and *Y*, such that every *X* has the right and authority to control, utilize and dispose of *Y*.

2.4.4.2 Relationships to other elements

Assuming that *X* owns *Y*, then:

policy

X has the right to establish policies that constrain agents and other entities in their use of *Y*

disposal

X has the right to transfer some or all of his rights with respect to *Y* to another entity.

transitive

if *P* is true of *Y* then *P* is true of *X*

2.4.4.3 Explanation

Essentially, when we say that *X* owns *Y* we mean that *X* has a significant set of rights with respect to *Y*, and that those rights are transferrable.

In general, ownership is partial, and there may be many entities that have rights with respect to some service or resource.

2.4.5 The *realized* relationship

2.4.5.1 Definition

The statement "concept *X* is realized as *Y*" denotes that the concept *X* is an abstraction of the concept *Y*. An equivalent view is that the concept *X* is implemented using *Y*.

2.4.5.2 Relationships to other elements

Assuming that *X* is realized as *Y*, then:

implemented

if *Y* is present, or true of a system, then the concept *X* applies to the system

reified

Y is a reification of the concept *X*.

2.4.5.3 Explanation

When we say that the concept or feature *X* is realized as *Y*, we mean that *Y* is an implementation or reification of the concept *X*. I.e., if *Y* is a valid concept of a system then we have also ensured that the concept *X* is valid of the same system.

For example, in the correlation feature, we state that message correlation requires that we associate identifiers with messages. This can be realized in a number of ways — including the identifier in the message header, message body, in a service binding and so on. The message identifier is a key to the realization of message correlation.

3 Stakeholder's Perspectives

This section examines the architecture from various perspectives, each perspective representing one coherent view of the architecture. For example, security represents one major stakeholder's perspective of the architecture itself.

3.1 Service Oriented Architecture

3.1.1 Distributed Systems

A *distributed system* consists of diverse, discrete software agents that must work together to perform some tasks. Furthermore, the agents in a distributed system do not operate in the same processing environment, so they must communicate by hardware/software protocol stacks over a network. This

means that communications with a distributed system are intrinsically less fast and reliable than those using direct code invocation and shared memory. This has important architectural implications because distributed systems require that developers (of infrastructure and applications) consider the unpredictable latency of remote access, and take into account issues of concurrency and the possibility of partial failure [Dist Comp].

Distributed *object* systems are distributed systems in which the semantics of object initialization and method invocation are exposed to remote systems by means of a proprietary or standardized mechanism to broker requests across system boundaries, marshall and unmarshall method argument data, etc. Distributed objects systems typically (albeit not necessarily) are characterized by objects maintaining a fairly complex internal state required to support their methods, a fine grained or "chatty" interaction between an object and a program using it, and a focus on a shared implementation type system and interface hierarchy between the object and the program that uses it.

A Service Oriented Architecture (SOA) is a form of distributed systems architecture that is typically characterized by the following properties:

- Logical view: The service is an abstracted, *logical* view of actual programs, databases, business processes, etc., defined in terms of what it *does*, typically carrying out a business-level operation.
- Message orientation: The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. The internal structure of an agent, including features such as its implementation language, process structure and even database structure, are deliberately abstracted away in the SOA: using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns so-called legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application that can be "wrapped" in message handling code that allows it to adhere to the formal service definition.
- Description orientation: A service is described by machine-processable meta data. The description supports the public nature of the SOA: only those details that are exposed to the public and important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.
- Granularity: Services tend to use a small number of operations with relatively large and complex messages.
- Network orientation: Services tend to be oriented toward use over a network, though this is not an absolute requirement.
- Platform neutral: Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

3.1.2 Web Services and Architectural Styles

Distributed object systems have a number of architectural challenges. [Dist Comp] and others point out:

- Problems introduced by latency and unreliability of the underlying transport.
- The lack of shared memory between the caller and object.
- The numerous problems introduced by partial failure scenarios.
- The challenges of concurrent access to remote resources.
- The fragility of distributed systems if incompatible updates are introduced to any participant.

These challenges apply irrespective of whether the distributed object system is implemented using COM/CORBA or Web services technologies. Web services are no less appropriate than the alternatives if the fundamental criteria for successful distributed object architectures are met. If these criteria are met, Web services technologies may be appropriate if the benefits they offer in terms of platform/vendor neutrality offset the performance and implementation immaturity issues they may introduce.

Conversely, using Web services technologies to implement a distributed system doesn't magically turn a distributed object architecture into an SOA. Nor are Web services technologies *necessarily* the best choice for implementing SOAs -- if the necessary infrastructure and expertise are in place to use COM

or CORBA as the implementation technology and there is no requirement for platform neutrality, using SOAP/WSDL may not add enough benefits to justify their costs in performance, etc.

In general SOA and Web services are most appropriate for applications:

- That must operate over the Internet where reliability and speed cannot be guaranteed;
- Where there is no ability to manage deployment so that all requesters and providers are upgraded at once;
- Where components of the distributed system run on different platforms and vendor products;
- Where an existing application needs to be exposed for use over a network, and can be wrapped as a Web service.

3.1.3 Relationship to the World Wide Web and REST Architectures

The World Wide Web operates as a networked information system that imposes several constraints: Agents identify objects in the system, called *resources*, with Uniform Resource Identifiers (URIs). Agents represent, describe, and communicate resource state via *representations* of the resource in a variety of widely-understood data formats (e.g. XML, HTML, CSS, JPEG, PNG). Agents exchange representations via protocols that use URIs to identify and directly or indirectly address the agents and resources. [Web Arch]

An even more constrained architectural style for reliable Web applications known as *Representation State Transfer* (REST) has been proposed by Roy Fielding and has inspired both the W3C Technical Architecture Group's architecture document [Web Arch] and many who see it as a model for how to build Web services [Fielding]. The REST Web is the subset of the WWW (based on HTTP) in which agents provide *uniform interface semantics* -- essentially create, retrieve, update and delete -- rather than arbitrary or application-specific interfaces, and manipulate resources only by the exchange of *representations*. Furthermore, the REST interactions are "stateless" in the sense that the meaning of a message does not depend on the state of the conversation.

We can identify two major classes of Web services:

- *REST-compliant Web services,* in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and
- *arbitrary Web services,* in which the service may expose an arbitrary set of operations.

Both classes of Web services use URIs to identify resources and use Web protocols (such as HTTP and SOAP 1.2) and XML data formats for messaging. (It should be noted that SOAP 1.2 *can* be used in a manner consistent with REST. However, SOAP 1.2 can also be used in a manner that is *not* consistent with REST.)

3.2 Web Services Technologies

Web service architecture involves many layered and interrelated technologies. There are many ways to visualize these technologies, just as there are many ways to build and use Web services. Figure 3-1 below provides one illustration of some of these technology families.

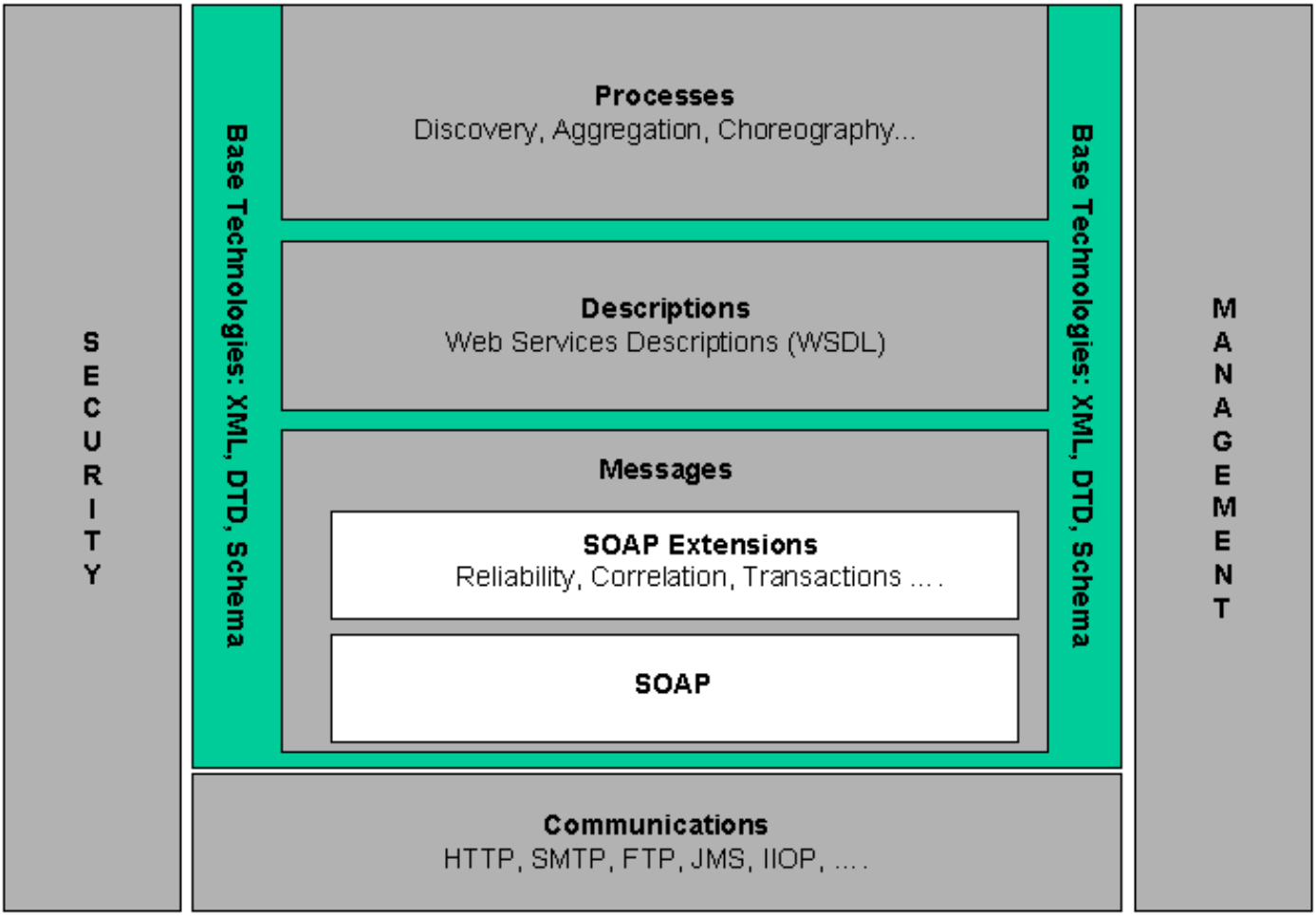


Figure 3-1. Web Services Architecture Stack

In this section we describe some of those technologies that seem critical and the role they fill in relation to this architecture. This is a necessarily bottom-up perspective, since, in this section, we are looking at Web services from the perspective of tools which can be used to design, build and deploy Web serivces.

The technologies that we consider here, in relation to the Architecture, are XML, SOAP, WSDL. However, there are many other technologies that may be useful. (For example, see the list of Web services specifications compiled by Roger Cutler and Paul Denning.) See also **B An Overview of Web Services Security Technologies**

3.2.1 XML

XML solves a key technology requirement that appears in many places. By offering a standard, flexible and inherently extensible data format, XML significantly reduces the burden of deploying the many technologies needed to ensure the success of Web services.

The important aspects of XML, for the purposes of this Architecture, are the core syntax itself, the concepts of the XML Infoset [XML Infoset], XML Schema and XML Namespaces.

XML Infoset is not a data format per se, but a formal set of information items and their associated properties that comprise an abstract description of an XML document [XML 1.0]. The XML Infoset specification provides for a consistent and rigorous set of definitions for use in other specifications that need to refer to the information in a well-formed XML document.

Serialization of the XML Infoset definitions of information may be expressed using XML 1.0 [XML 1.0]. However, this is not an inherent requirement of the architecture. The flexibility in choice of serialization format(s) allows for broader interoperability between agents in the system. In the future, a binary

encoding of the XML infoset may be a suitable replacement for the textual serialization. Such a binary encoding may be more efficient and more suitable for machine-to-machine interactions.

3.2.2 SOAP

SOAP 1.2 provides a standard, extensible, composable framework for packaging and exchanging XML messages. In the context of this architecture, SOAP 1.2 also provides a convenient mechanism for referencing capabilities (typically by use of headers).

[SOAP 1.2 Part 1] defines an XML-based messaging framework: a processing model and an exensibility model. SOAP messages can be carried by a variety of network protocols; such as HTTP, SMTP, FTP, RMI/IIOP, or a proprietary messaging protocol.

[SOAP 1.2 Part 2] defines three optional components: a set of encoding rules for expressing instances of application-defined data types, a convention for representing remote procedure calls (RPC) and responses, and a set of rules for using SOAP with HTTP/1.1.

While SOAP Version 1.2 [SOAP 1.2 Part 1] doesn't define "SOAP" as an acronym anymore, there are two expansions of the term that reflect these different ways in which the technology can be interpreted:

1. Service Oriented Architecture Protocol: In the general case, a SOAP message represents the information needed to invoke a service or reflect the results of a service invocation, and contains the information specified in the service interface definition.
2. Simple Object Access Protocol: When using the optional SOAP RPC Representation, a SOAP message represents a method invocation on a remote object, and the serialization of in the argument list of that method that must be moved from the local environment to the remote environment.

3.2.3 WSDL

WSDL 2.0[WSDL 2.0 Part 1] is a language for describing Web services.

WSDL describes Web services starting with the messages that are exchanged between the requester and provider agents. The messages themselves are described abstractly and then bound to a concrete network protocol and message format.

Web service definitions can be mapped to any implementation language, platform, object model, or messaging system. Simple extensions to existing Internet infrastructure can implement Web services for interaction via browsers or directly within an application. The application could be implemented using COM, JMS, CORBA, COBOL, or any number of proprietary integration solutions. As long as both the sender and receiver agree on the service description, (e.g. WSDL file), the implementations behind the Web services can be anything.

3.3 Using Web Services

The introduction outlined and illustrated (in Figure 1-1) the four broad steps involved in the process of engaging a Web service (see **1.4.5 Overview of Engaging a Web Service**). This section expands on these steps. Although these steps are *necessary*, they may not be *sufficient*: many scenarios will require additional steps, or significant refinements of these fundamental steps. Furthermore, the order in which the steps are performed may vary from situation to situation.

1. The requester and provider entities "become known to each other", in the sense that whichever party initiates the interaction must become aware of the other party. There are two cases.
    a. In a typical case, the *requester* agent will be the initiator. In this case, we would say that the requester entity must become aware of the provider entity, i.e., the requester agent must somehow obtain the address of the provider agent. There are two ways this may typically

occur: (1) the requester entity may obtain the provider agent's address directly from the provider entity; or (2) the requester entity may use a discovery service to locate a suitable service description (which contains the provider agent's invocation address) via an associated functional description, either through manual discovery or autonomous selection. These cases are described more fully in **3.4 Web Service Discovery**.

b. In other cases, the *provider* agent may initiate the exchange of messages between the requester and provider agents. In this case, saying that the requester and provider entities become known to each other actually means that the *provider* entity becomes aware of the *requester* entity, i.e., the provider agent somehow obtains the address of the requester agent. How this occurs is application dependent and irrelevant to this architecture. Although this case is expected to be less common than when the requester agent is the initiator, it is important in some "push" or subscription scenarios.

2. The requester entity and provider entity agree on the service description (a WSDL document) and semantics that will govern the interaction between the requester agent and the provider agent. (See the note below on "Agreeing on the Same Semantics and Service Description for further explanation of what is meant here by "agree".)

   This does not necessarily mean that the requester and provider entities must communicate or negotiate with each other. It simply means that both parties must have the same (or compatible) understandings of the service description and semantics, and intend to uphold them. There are many ways this can be achieved, such as:

   - The requester and provider entities may communicate directly with each other, to explicitly agree on the service description and semantics.
   - The provider entity may publish and offer both the service description and semantics as take-it-or-leave-it "contracts" that the requester entity must accept unmodified as conditions of use.
   - The service description and semantics (excepting the network address of the particular service) may be defined as a standard by an industry organization, and used by many requester and provider entities. In this case, the act of the requester and provider entities reaching agreement is accomplished by both parties independently conforming to the same standard.
   - The service description and semantics (perhaps excepting the network address of the service) may be defined and published by the *requester* entity (even if they are written from provider entity's perspective), and offered to provider entities on a take-it-or-leave-it basis. This may occur, for example, if a large company requires its suppliers to provide Web services that conform to a particular service description and semantics. In this case, agreement is achieved by the provider entity adopting the service description and semantics that the requester entity has published.

   Depending on the scenario, Step 2 (or portions of Step 2) may be performed prior to Step 1.

3. The service description and semantics are input to, or embodied in, both the requester agent and the provider agent as appropriate. In other words, the information in them must either be input to, or implemented in, the requester and provider agents. There are many ways this can be achieved, and this architecture does not specify or care what means are used. For example:

   - An agent could be hard coded to implement a particular, fixed service description and semantics.
   - An agent could be coded in a more general way, and the desired service description and/or semantics could be input dynamically.
   - An agent could be created first, and the service description and/or semantics could be generated or deduced from the agent code. For example, a tool could examine a set of existing class files to generate a service description.

   Regardless of the approach used, from an information perspective both the semantics and the service description must somehow be input to, or implemented in, both the requester agent and the provider agent before the two agents can interact. (This is a slight simplification; see the note below on "Agreeing" on the Same Semantics and Service Description for further explanation.)

4. The requester agent and provider agent exchange SOAP messages on behalf of their owners.

Note:

"Agreeing" on the Same Semantics and Service Description. Although it is convenient to say that the requester and provider entities must "agree" on the semantics and the service description, it is a slight simplification (and perhaps slightly misleading) to say that the parties *must* agree on the *same* semantics and service description:

- The word "agree" often connotes an active communication between the parties and an explicit act (such as signing a contract) to cause the agreement to become binding on the two parties, yet neither of these is required in the case of step 2 above.
- It is a slight simplification to say that the requester and provider agents must implement the *same* semantics and WSD, for two reasons: (1) the requester agent implements them from the perspective of the requester entity, while the provider agent implements them from the perspective of the provider entity (for example, one party's input is the other party's output); and (2) the requester and provider agents only need to implement those aspects of the service description and semantics that are *relevant* to their respective roles.

In summary, it is convenient (and evocative) to say that the requester and provider entities must *agree* on the semantics and the service description that will govern the interaction between the requester and provider agents, but it would be more accurate to say that they simply need to have a *congruent* or *non-conflicting view* of the semantics and service description of the interaction.

3.4 Web Service Discovery

If the requester entity wishes to initiate an interaction with a provider entity and it does not already know what provider agent it wishes to engage, then the requester entity may need to "discover" a suitable candidate. Discovery is "the act of locating a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria. " [WS Glossary] The goal is to find an appropriate Web service.

A discovery service is a service that facilitates the process of performing discovery. It is a logical role, and could be performed by either the requester agent, the provider agent or some other agent.

Figure 3-2 ("Discovery Process") expands on Figure 1-1 to describe the process of engaging a Web service when a discovery service is used.

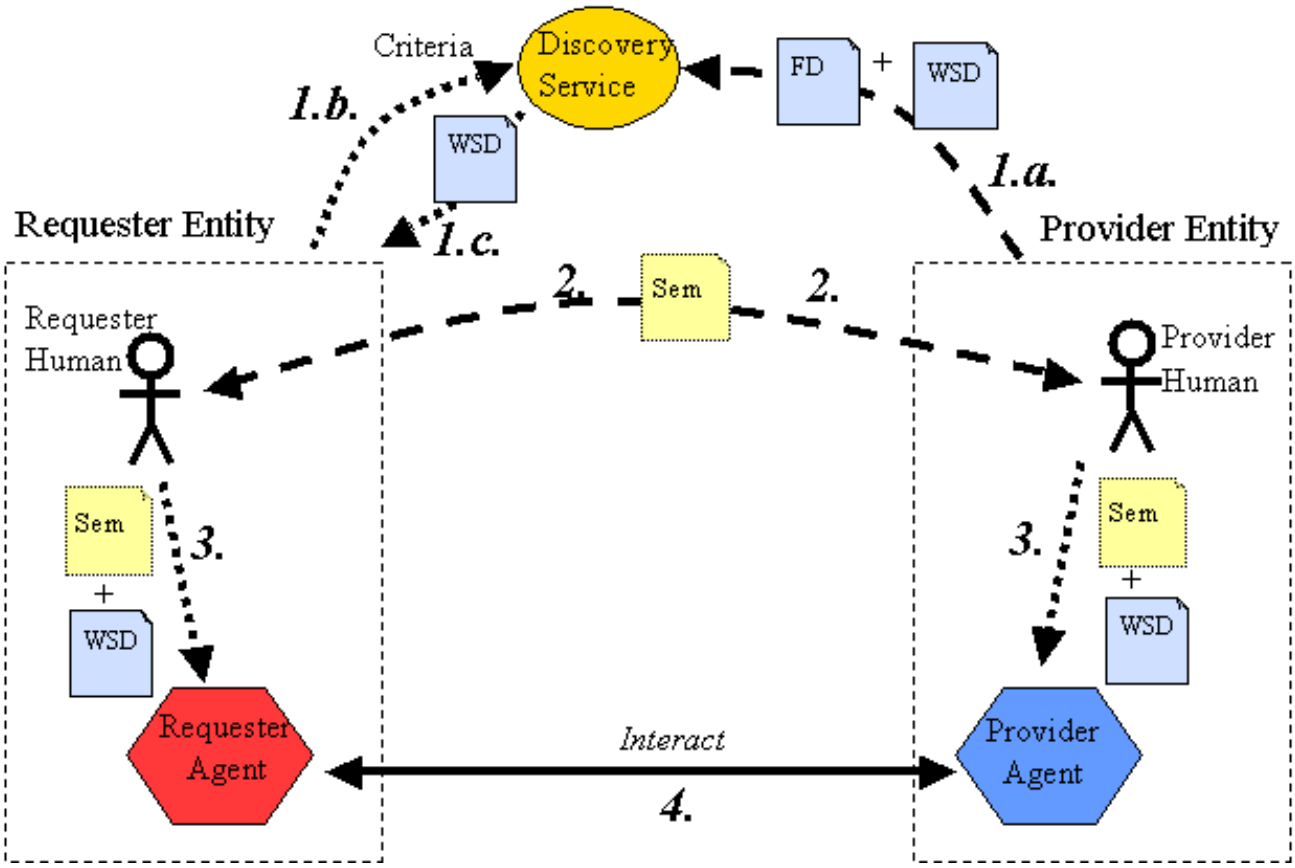


Figure 3-2. Discovery Process

Service engagement using a discovery service proceeds in roughly the following steps.

1. The requester and provider entities "become known to each other":
   a. The discovery service somehow obtains both the Web service description ("WSD" in Figure 3-2) and an associated functional description ("FD") of the service.

      The functional description ("FD" in Figure 3-2) is a machine-processable description of the functionality (or partial semantics) of the service that the provider entity is offering. It could be as simple as a few words of meta data or a URI, or it may be more complex, such as a TModel (in UDDI) or a collection of RDF, DAML-S or OWL-S statements.

      This architecture does not specify or care how the discovery service obtains the service description or functional description. For example, if the discovery service is implemented as a search engine, then it might crawl the Web, collecting service descriptions wherever it finds them, with the provider entity having no knowledge of it. Or, if the discovery service is implemented as a registry (such as UDDI), then the provider entity may need to actively publish the service description and functional description directly to the discovery service.
   b. The requester entity supplies criteria to the discovery service to select a Web service description based on its associated functional description, capabilities and potentially other characteristics. One might locate a service having certain desired functionality or semantics; however, it may be possible to specify "non-functional" criteria related to the provider agent, such as the name of the provider entity, performance or reliability criteria, or criteria related to the provider entity, such as the provider entity's vendor rating.
   c. The discovery service returns one or more Web service descriptions (or references to them) that meet the specified criteria. If multiple service descriptions are returned, the requester entity selects one, perhaps using additional criteria.
2. The requester and provider entities agree on the semantics ("Sem" in Figure 3-2) of the desired interaction. Although this may commonly be achieved by the provider entity defining the semantics and offering them on a take-it-or-leave-it basis to the requester entity, it could be

achieved in other ways. For example, both parties may adopt certain standard service semantics that are defined by some industry standards body. Or in some circumstances the requester could define the semantics. The important point is that the parties must *agree* (in the sense described in **3.3 Using Web Services**) on the semantics, regardless of how that is achieved.

Step 2 also requires that the parties agree (in the sense described in **3.3 Using Web Services**) on the service description that is to be used. However, since the requester entity obtained the Web service description in Step 1.c, in effect the requester and provider entities have already done so.

3. The service description and semantics are input to, or embodied in, both the requester agent and the provider agent, as appropriate.
4. The requester agent and provider agent exchange SOAP messages on behalf of their owners.

3.4.1 Manual Versus Autonomous Discovery

The discovery process described above is not specific about who or what within the requester entity actually performs the discovery. Under *manual discovery*, a requester *human* uses a discovery service (typically at design time) to locate and select a service description that meets the desired functional and other criteria. Under *autonomous discovery*, the requester *agent* performs this task, either at design time or run time. Although the steps are similar in either case, the constraints and needs are significantly different, such as:

- *Interface requirements.* The requirements for something that is intended for human interaction are very different from the requirements for something that is intended for machine interaction.
- *Need for standardization.* There is far less need to standardize an interface or protocol that humans use than those that machines are intended to use.
- *Trust.* People do not necessarily trust machines to make decisions that may have significant consequences. This is explained more fully in **3.6.4.5 Trust and Discovery**.

In the case of autonomous discovery, the need for machine-processable semantics is greatly increased.

One situation in which autonomous discovery is often needed is when the requester agent has been interacting with a particular provider agent, but for some reason needs to refresh its choice of provider agent, either because the previous provider agent is no longer available, or other reasons.

3.4.2 Discovery: Registry, Index or Peer-to-Peer?

At present, there are three leading viewpoints on how a discovery service should be conceived: as a *registry*, as an *index*, or as a *peer-to-peer* system. What are the differences? For what purpose is one better than the other?

3.4.2.1 The Registry Approach

A *registry* is an authoritative, centrally controlled store of information.

- Publishing a service description requires an active step by the provider entity: it must explicitly place the information into the registry before that information is available to others. In the case of a registry:
- The registry owner decides *who* has authority to place information into, or update, the registry. Although the owner of registry R may delegate permission to approved provider entities that wish to publish their own service descriptions, an arbitrary third party could not publish a description of someone else's service in registry R. This means, for example, that company X would not be able to register a functional description of company Y's service, even if that description would be valuable to others and may be superior in some ways to Y's own description.
- The registry owner decides *what* information is placed in the registry. Others cannot independently augment that information.

UDDI is often seen as an example of the registry approach, but it can also be used as an index.

3.4.2.2 The Index Approach

In contrast with a registry, an *index* is a compilation or guide to information that exists elsewhere. It is not authorative and does not centrally control the information that it references. In the case of an index:

- Publishing is passive: the provider entity exposes the service and functional descriptions on the Web, and those who are interested (the index owners) collect them without the provider entity's specific knowledge.
- Anyone can create their own index. When descriptions are exposed, they can be harvested using spiders and arranged into an index. Multiple organizations may have such indexes.
- The information contained in an index could be out of date. However, since the index contains pointers to the authoritative information, the information can be verified before use.
- An index could include third-party information.
- Different indexes could provide different kinds of information — some richer, some sparser.
- Free-market forces determine which index people will use to discover the information that they seek.

Google is often cited as an example of the index approach.

It is important to note that the key difference between the registry approach and the index approach is not merely the difference between a registry itself and an index *in isolation*. Indeed, UDDI could be used as a means to implement an individual index: just spider the Web, and put the results into a UDDI registry. Rather, the key difference is one of *control*: Who controls *what* and *how* service descriptions get discovered? In the registry model, it is the owner of the registry who controls this. In the index model, since anyone can create an index, market forces determine which indexes become popular. Hence, it is effectively the market that controls what and how service descriptions get discovered.

3.4.2.3 Peer-to-Peer (P2P) Discovery

Peer-to-Peer (P2P) computing provides an alternative that does not rely on centralized registries; rather it allows Web services to discover each other dynamically. Under this view, a Web service is a node in a network of peers, which may or may not be Web services. At discovery time, a requester agent queries its neighbors in search of a suitable Web service. If any one of them matches the request, then it replies. Otherwise each queries its own neighboring peers and the query propagates through the network until a particular hop count or other termination criterion is reached.

Peer-to-peer architectures do not need a centralized registry, since any node will respond to the queries it receives. P2P architectures do not have a single point of failure, such as a centralized registry. Furthermore, each node may contain its own indexing of the existing Web services. Finally, nodes contact each other directly, so the information they they receive is known to be current. (In contrast, in the registry or index approach there may be significant latency between the time a Web service is updated and the updated description is reflected in the registry or index.)

The reliability provided by the high connectivity of P2P systems comes with performance costs and lack of guarantees of predicting the path of propagation. Any node in the P2P network has to provide the resources needed to guarantee query propagations and response routing, which in turn means that most of the time the node acts as a relayer of information that may be of no interest to the node itself. This results in inefficiencies and large overhead especially as the nodes become more numerous and connectivity increases. Furthermore, there may be no guarantee that a request will spread across the entire network, therefore there is no guarantee to find the providers of a service.

3.4.2.4 Discovery Service Trade-Offs

Because of their respective advantages and disadvantages, P2P systems, indexes and centralized registries strike different trade-offs that make them appropriate in different situations. P2P systems are more appropriate in dynamic environments in which proximity naturally limits the need to propagate requests, such as ubiquitous computing. Centralized registries may be more appropriate in more static

or controlled environments where information does not change frequently. Indexes may be more appropriate in situations that must scale well and accommodate competition and diversity in indexing strategies.

3.4.3 Federated Discovery Services

Although the registry viewpoint is a more centralized approach to discovery than the index approach, there will arise situations where multiple registries exist on the Web. It is expected that multiple indexes will also exist. In such an environment, web service requesters that need to use a discovery service may need to obtain information from more than one registry or index. Federation refers to the ability to consolidate the results of queries that span more than a single registry or index, and make them appear more like a single service.

A registry or index may contain information about other registries or indexes to help support federation. For example, a registry dedicated to air travel services may know about another registry dedicated to rail travel services. A third registry for general travel services may contain information about some travel services, but may look to other registries for certain categories of services. A search of the general travel registry may return a referral to the requester pointing them to the rail travel registry. Federation of results in this scenario, as contrasted to the referral, would require the general travel registry to submit a query to the rail travel registry on behalf of the requester. The general travel registry would then merge the results of the query to the rail travel registry with the results of a query to its own registry. The general, rail, and air travel registries may need to share a common taxonomy or ontology to avoid forwarding inappropriate queries to other registries. In this scenario, we assume the general travel registry examined the query from the requester and therefore did not forward the query to the air travel registry.

The general travel registry could have discovered the rail travel registry using a spider or index approach. An indexing engine could have come across a registry, and based on the information it harvested from the registry classified it as a rail travel registry. An alternative approach would be for the rail travel registry to publish information to the general travel registry and using the shared taxonomy could classify itself as a registry for rail travel services.

Note that each registry or index may provide a web service for discovery, so it may be appropriate to use a choreography or orchestration description language to describe the exchanges among these services needed for federation.

3.4.4 Functional Descriptions and Discovery

As mentioned at the beginning of **2.3.3.1 Discovery**, Web services discovery requires the ability to search for appropriate Web services based on functional descriptions ("FD" in Figure 3-2) or other criteria. Because these functional descriptions need to be machine processable, written by many provider entities and read by many requester entities, an appropriate language for representing functional descriptions should at least be:

- Web friendly (based on URIs and globally scalable)
- Unambiguous
- Capable of expressing any existing or future functionality
- Capable of expressing existing and new vocabularies and relationships between functionalities

This is an area that needs further standardization work. One such effort is OWL-S.

3.5 Web Service Semantics

For computer programs to successfully interact with each other a number of conditions must be established:

1. There must be a physical connection between them, such that data from one process may reach

another
2. There must be agreement (in the sense discussed in **3.3 Using Web Services**) on the *form* of the data such as whether the data is lines of text, XML structures, etc.
3. The two (or more) programs must share agreement (in the sense discussed in **3.3 Using Web Services**) as to the intended meaning of the data. For example, whether the data is intended to represent an HTML page to be rendered, or whether the data represents the current status of a bank account; the expectations and the processing involved in processing the data is different — even if the form of the data is identical.
4. There must be agreement (in the sense discussed in **3.3 Using Web Services**) as to the implied processing of messages exchanged between the programs. For example, purchase ordering Web service is expected — by the agent that places the order — to process the document containing the purchase order *as a purchase order*, as opposed to simply recording it for auditing purposes.

As we shall see below, more may be required, but for now this list is sufficient.

3.5.1 Message semantics and visibility

The extent to which the shared agreement about the form, structure and meaning of a message is shared *beyond* just the agents involved with the message governs the overall *visibility* of the message semantics. The emphasis on messages, rather than on the actions that are caused by messages, means that SOAs have good visibility: third parties may inspect the flow of messages and have a some assurance as to the services being invoked and the roles of the various parties. This, in turn, means that intermediaries, such as firewalls, are in a better situation for performing their functions. A firewall can look at the message traffic, and at the structure of the message, and make predictable and reasonable decisions about security.

In REST-compliant SOAs, additional visibility comes from the uniform interface semantics, essentially those of the HTTP protocol: an intermediary can inspect the URI of the resource being manipulated, the TCP/IP address of the requester, and the interface operation requested (e.g. GET, PUT, DELETE) and determine whether the requested operation should be performed. The TCP/IP and HTTP protocols have a widely supported set of conventions (e.g. known ports) to support intermediaries, and firewalls, proxies, caches, etc. are almost universal today.

Visibility, however, goes beyond firewalls. In this architecture, instead of emphasising a REST-style uniform interface, we emphasize messages' structure in terms of envelopes, headers and bodies. We enhance visibility architecturally by fostering agreements on particular forms of headers. For example, by having well-known standards that describe the form and interpretation of authentication tokens in headers, we can simultaneously reduce the cost of performing authentication and increase the overall visiblity of the message's semantics: if the authentication aspect of a message can be specified in a standard way then it is easier for a larger number of interested parties to process the message. Furthermore, increased visibility can reduce the cost of entry into a marketplace.

Other potential examples of standardized headers include support for message reliability, support for message correlation, support for process flow and service composition and support for choreography.

This argument can be extended from obvious infrastructure-related processing of messages to more application-related processing of the message. For example, by capturing customer identification in a well-understood header, then all applications capable of processing that header will be able to extract the customer information of a message *independently* of the intended final disposition of the message.

This, in turn, suggests an extremely powerful architectural approach to message processing: different stakeholders in an organization, represented by different applications processing different aspects of messages, can collaborate with a minimal pre-ordained design.

3.5.2 Semantics of the Architectural Models

The different models in the architecture focus on different aspects of the interoperability issues between

Web service agents. The Message Oriented Model focuses on how Web service agents (requester and provider agents) may interact with each other using a message oriented communication model. The format of messages as XML infosets and the structuring of messages in terms of envelopes, headers and bodies, as described in that model, acts to lay a foundation for the standard comprehension of messages exchanged between Web service agents.

The Service Oriented Model builds on the basics of message communication by adding the concept of action and service. Essentially, the service model allows us to interpret messages as requests for actions and as responses to those requests. Furthermore, it allows an interpretation of the different aspects of messages to be expressed in terms of different expectations, in well understood ways, of the different parts of the message: in effect, an incremental and layered approach to service is possible using well understood headers.

The Resource Oriented Model extends this further by adding the concept of resource. Resources are important internally to the architecture (a Web service is best understood as a resource in the context of Web service management and in terms of policy management) and externally: resources are an important metaphor for interpreting the interaction between a requester entity and a provider entity.

3.5.3 The Role of Metadata

An important part of the Service Oriented Architecture approach is the extensive use of metadata. This is important for several reasons: it fosters interoperability by requiring increased precision in the documentation of Web services and it permits tools to give a higher degree of automation to the development of Web services (and hence lowers the cost of deploying same).

The metadata associated with a Web service can be regarded as a partial machine-readable description of the semantics of the Web service. In particular using technologies such as WSDL, a Web service can be described in a machine readable document as to the forms of expected messages, the datatypes of elements of messages and using a choreography description language the expected flows of messages between Web service agents.

However, current technologies used for describing Web services are probably not yet sufficient to meet interoperability requirements on a global scale. We see the following areas where increased and richer meta-data would further enhance interoperability:

- It should be possible to identify the real-world entities referenced by elements of messages. For example, when using a credit card to arrange for the purchase of goods or services, the element of the message that contains the credit card information is fundamentally a reference to a real-world entity: the account of the card holder.

  The appropriate technology for this is standardized ontology languages, such as OWL.
- It should be possible to identify the expected effects of any actions undertaken by Web service requester and provider agents. That this cannot be captured by datatyping can be illustrated with the example of a Web service for withdrawing money from an account as compared to depositing money (more accurately, transferring from an account to another account, or vice versa). The datatypes of messages associated with two such services may be identical, but with dramatically different effects: instead of being paid for goods and services, the risk is that one's account is drained instead.

  We expect that a richer model of services, together with technologies for identifying the effects of actions, is required. Such a model is likely to incorporate concepts such as contracts (both legally binding and technical contracts) as well as ontologies of action.
- Finally, a Web service program may "understand" what a particular message means in terms of the expected results of the message, but, unless there is also an understanding of the relationship between the requester entity and the provider entity, the provider agent may not be able to accurately determine whether the requested actions are *warranted*.

  For example, a provider agent may receive a request to transfer money from one account to another. The request may be valid in the sense that the datatypes of the message are correct, and that the semantic markers associated with the message lead the provider agent to correctly

interpret the message as a transfer request. However, the transaction still may not be valid, or fully comprehensible, unless the provider agent can properly identfy the relationship of the requester agent's owner (i.e., the requester entity) to the requested action. Currently, such concerns are often treated simply as security considerations, which they are, in an ad hoc fashion. However, when one considers issues such as delegated authority, proxy requests, and so on, it becomes clear that a simple authentication model cannot accurately capture the requirements.

We expect that a model that formalizes concepts such as institutions, roles (in business terms), "regulations" and regulation formation will be required. With such a model we should be able to capture not only simple notions of authority, but more subtle distinctions such as the authority to delegate an action, authority by virtue of such delegation, authority to authorize and so on.

3.6 Web Services Security

Threats to Web services involve threats to the host system, the application and the entire network infrastructure. To secure Web services, a range of XML-based security mechanisms are needed to solve problems related to authentication, role-based access control, distributed security policy enforcement, message layer security that accommodate the presence of intermediaries.

At this time, there are no broadly-adopted specifications for Web services security. As a result developers can either build up services that do not use these capabilities or can develop ad-hoc solutions that may lead to interoperability problems.

Web services implementations may require point-to-point and/or end-to-end security mechanisms, depending upon the degree of threat or risk. Traditional, connection-oriented, point-to-point security mechanisms may not meet the end-to-end security requirements of Web services. However, security is a balance of assessed risk and cost of countermeasures. Depending on implementers risk tolerance, point-to-point transport level security can provide enough security countermeasures.

3.6.1 Security policies

From the perspective of this architecture, there are three fundamental concepts related to security: the resources that must be secured; the mechanisms by which these resources are secured (i.e., policy guards); and policies, which are machine-processable documents describing constraints on these resources.

Policies can be logically broken down into two main types: permission policies and obligatory policies. A permission policy concerns those actions and accesses that entities are permitted to perform and an obligation policy concerns those actions and states that entities are required to perform. These are closely related, and dependent: it is not consistent to be obliged to perform some action that one does not have permission to perform. A given policy document is likely to contain a mix of obligation and permission policy statements.

The two kinds of policies have different enforcement mechanisms: a permission guard is a mechanism that can be used to verify that a requested action or access is permitted; an audit guard can only verify after the fact that an obligation has not been met. The precise form of these guards is likely to vary, both with the resources being controlled and with the implementation technologies deployed.

The architecture is principally concerned with the existence of guards and their role in the architecture. In a well engineered system it may be possible to construct guards that are not directly visible to either the requester or provider agents. For example, the unauthorized access threat may be countered by a mechanism that validates the identity of potential agents who wish access the controlled resource. That mechanism is, in turn, controlled by the policy document which expresses what evidence must be offered by which agents before the access is permitted.

A permission guard acts as a guard enabling or disabling access to a resource or action. In the context of SOAP, for example, one important role of SOAP intermediaries is that of permission guards: the intermediary may not, in fact, forward a message if some security policy is violated.

Not all guards are active processes. For example, confidentiality of information is encouraged by encryption of messages. As noted above, it is potentially necessary to encrypt not only the content of SOAP messages but also the identities of the sender and receiver agents. The guard here is the encryption itself; although this may be further backed up by other active guards that apply policy.

3.6.2 Message Level Security Threats

Traditional network level security mechanisms such as Transport Layer Security (SSL/TLS), Virtual Private Networks (VPNs), IPSec (Internet Protocol Security) and Secure Multipurpose Internet Mail Exchange (S/MIME) are point-to-point technologies. Although traditional security technologies are used in Web services security, however, they are not sufficient for providing end-to-end security context, as Web services require more granularities. In general, Web services use a message-based approach that enables complex interactions that can include the routing of messages between and across various trust domains.

Web services face traditional security challenges. A message might travel between various intermediaries before it reaches its destination. Therefore, message-level security is important as opposed to point-to-point, transport-level, security. In Figure 3-3 below, the requester agent is communicating with the ultimate receiver through the use of one or more intermediaries. The security context of the SOAP message is end-to-end. However, there may be a need for the intermediary to have access to some of the information in the message. This is illustrated as a security context between the intermediary and the original requester agent, and the intermediary and the ultimate receiver.

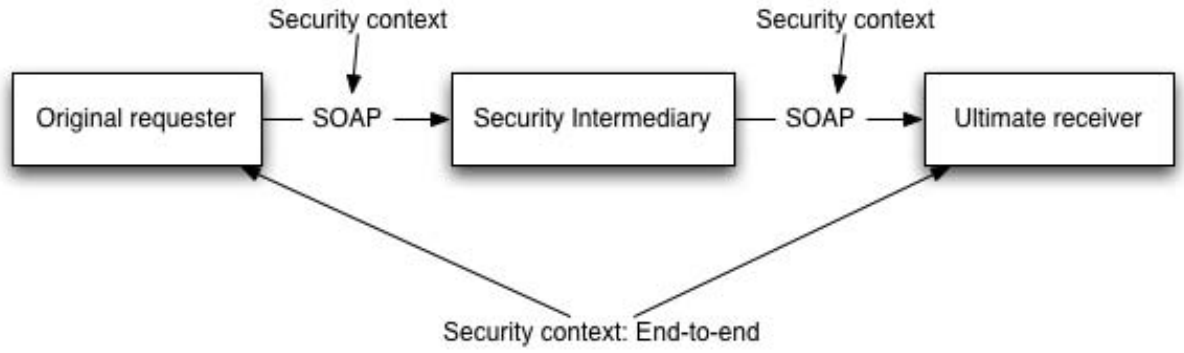


Figure 3-3. End-to-End Security

The threats listed below addresses message security.

3.6.2.1 Message Alteration

These threats affect message integrity, whereby, an attacker may modify parts (or the whole) message. For example, an attacker may delete part of a message, or modify part of a message, or insert extra information into a message. The attacks may affect message header and/or body parts.

An attacker may also affect message integrity by manipulating its attachments. For example, an attacker may delete an attachment, or modify an attachment, or insert an attachment into a message.

3.6.2.2 Confidentiality

In this threat, unauthorized entities obtain access to information with in a message or message parts. For example, an intermediary obtains access to credit card information that was intended for the ultimate recipient.

3.6.2.3 Man-in-the-middle

Man-in-the-middle attacks are also known as bucket-brigade attacks. In this kind of assault it is possible for an attacker to compromise a SOAP intermediary and then intercepts messages between the web service requester and the ultimate receiver. The original parties will think that they are communicating with each other. The attacker may just have access to the messages or may modify them. Mutual authentication techniques can be used to alleviate the threats of this attack.

3.6.2.4 Spoofing

Spoofing is a complex attack that exploits trust relationships. The attacker assumes the identity of a trusted entity in order to sabotage the security of the target entity. As far as the target entity knows, it is carrying on a conversation with a trusted entity. Usually, spoofing is used as a technique to launch other form of attacks such as forged messages. Strong authentication techniques are needed to defend against such attacks.

3.6.2.5 Denial of Service

Denial of service (DoS) attacks focus on preventing legitimate users of a service from the ability to use the service. DoS attacks are easy to implement and can cause significant damage. DoS attacks can disrupt the operation of the agent that is under attack and effectively disconnect it from the rest of the world. DoS attacks can take various forms and target variety of services. DoS attacks exploit weaknesses in the architecture of the system that is under attack. Ironically, security mechanisms themselves add overhead that can be exploited in DoS attacks.

Distributed denial of service (DDoS) attacks uses the resources of more than one machine to launch synchronized DoS attacks on a resource.

3.6.2.6 Replay Attacks

In this attack an intruder intercepts a message and then replays it back to a targeted agent. Appropriate authentication techniques coupled with techniques such as time stamp and sequence numbering the messages can defend against replay attacks.

3.6.3 Web Services Security Requirements

There are many security challenges for adopting Web services. At the highest level, the objective is to create an environment, where message level transactions and business processes can be conducted securely in an end-to-end fashion. There is a need to ensure that messages are secured during transit, with or without the presence of intermediaries. There may also be a need to ensure the security of the data in storage.

The requirements for providing end-to-end security for Web services are summarized in the next sub-sections.

3.6.3.1 Authentication Mechanisms

Authentication is needed in order to verify the identities of the requester and provider agents. In some cases, the use of mutual authentication may be needed since the participants may not necessarily be directly connected by a single hop. For example the participants might be the initial requester and an intermediary. Depending on the security policy it may be possible to authenticate the requester, the receiver or to mandate the use of mutual authentication.

Several methods can be used to authenticate services. Techniques include: passwords, one time pass and certificates. Password-based authentication must use strong passwords. Password authentication alone may be insufficient. Based on vulnerability assessment it may be necessary to combine password authentication with other authentication and authorization process such as certificates, Lightweight Directory Access Protocol (LDAP), Remote Authentication Dial-in User Service (RADIUS), Kerberos, and Public Key Infrastructure (PKI).

3.6.3.2 Authorization

Authorization is needed in order to control access to resources. Once authenticated, authorization mechanisms control the requester access to appropriate system resources. There should be controlled access to systems and their components. Policy determines the access rights of a requester. The principle of least privilege access should be used when access rights are given to a requester.

3.6.3.3 Data Integrity and Data Confidentiality

Data integrity techniques ensure that information has not been altered, or modified during transmission without detection. Data confidentiality ensures that the data is only accessible by the intended parties. Data encryption and digital signature techniques can be used for this purpose.

3.6.3.4 Integrity of Transactions and Communications

This is needed to ensure that the business process was done properly and the flow of operations was executed in a correct manner.

3.6.3.5 Non-Repudiation

Non-repudiation is a security service that protects a party to a transaction against false denial of the occurrence of that transaction by another party. Non-repudiation technologies provide evidence about the occurrence of transactions that that may be used by a third party to resolve disagreement.

3.6.3.6 End-to-End Integrity and Confidentiality of Messages

The integrity and confidentiality of messages must be ensured even in the presence of intermediaries.

3.6.3.7 Audit Trails

Audit trails are needed in order to trace user access and behavior. They are also needed in order to ensure system integrity through verification. Audit trails can be performed by agents. Such agents can play the role of an audit guard that can monitor; watch resources and other agents, validating those obligations that have been established are respected and/or discharged. It is often not possible to prevent the violation of obligations. Instead, if an audit guard detects a policy violation, some form of retribution or remediation must be enacted. The precise forms of this are, of course, beyond the scope of this architecture.

3.6.3.8 Distributed Enforcement of Security Policies

Implementers must be able to define a security policy and enforce it across various platforms with varying privileges.

3.6.4 Security Consideration of This Architecture

Organizations that implement Web services must be able to conduct business in a secure fashion. This implies that all aspects of Web services including routing, management, publication, and discovery should be performed in a secure manner. Web services implementers must be able to utilize security services such as authentication, authorization, encryption and auditing.

Web services messages can flow through firewalls, and can be tunneled through existing ports and protocols. Web services security requires the use of appropriate corporate wide policies that may need to be integrated with external cross-enterprise policy and trust resolution. Organizations may need to implement the capabilities that are listed next.

3.6.4.1 Cross-Domain Identities

Requester and provider agents may communicate with each other using various identity verification schemes from different security domains. Many systems define role based access privileges based on identity. It is important for Web services to be able to support the mapping of identities across multiple domains and even within a single domain.

A provider entity and a requester entity may use their identities to encrypt and sign messages that they exchange. They may exchange identity credentials within a context of initial messages (handshake). That allows further trusted interactions. Service's identity is optional, and it is perfectly possible to implement a business service without an identity if it always acts on behalf of a requester entity (that is, impersonating the requester entity). Not having a requester entity's identity translates into anonymous access, which is rarely allowed for business services.

3.6.4.2 Distributed Policies

Security Policies that are associated with requester entity, service and discovery mechanism can be used to define the access privileges of request and responses between parties. These polices can be validated at run time in the context of interaction. Each party in an interaction validates its own policies.

3.6.4.3 Trust Policies

Trust Policies are distributed policies that apply to the environment of the other side's party in an interaction. A requester entity needs to *trust* the environment of a service and the provider entity needs to trust the environment of the requester entity. Trust policies may be recursive — they may be defined against trust policies of involved parties and even whole domains. An example of this is: "I will trust you if you trust my friend and my friend trusts you."

Distributed Identities, Policies and Trust can be described and processed by a machine. For example, an X.509 certificate can be embedded in an message, thus asserting the sender’s Identity. A Policy can be described in XML and attached to the service contract. Machines could process, resolve and adjust security based on the given descriptions.

Trust mechanisms can be used to form Delegation and Federation relationships. These mechanisms can be used to facilitate secure interactions between web services across trust boundaries in a distributed fashion.

3.6.4.4 Secure Discovery Mechanism

Secure Discovery Mechanism enforces policies that govern publication and discovery of a service. For example, developers of SOA applications for the procurement department may not be allowed to discover services available in the human resources department, if those developers are not entitled to use human resources services. When publishing a service, an identity is usually necessary to assert service publication policies, except for some cases of peer-to-peer discovery. When a requester entity discovers a service, it may or may not provide an Identity; discovery may well be anonymous.

3.6.4.5 Trust and Discovery

Suppose a requester entity discovers a Web service being offered by a provider entity that was previously unknown to that requester entity. Should the requester entity *trust* that service? If the use of that service requires the requester to divulge sensitive information (such as credit card numbers) to the service then there may be significant risk involved.

This decision — whether or not to trust a particular service — inherently arises when a requester entity chooses a Web service from a previously unknown provider entity. This has ramification in the discovery process, and leads to an important difference between manual discovery and autonomous discovery.

When manual discovery is used, a human makes the judgement (perhaps using other, independently obtained information) of whether to trust and engage a previously unknown service that is discovered.

Whereas with autonomous discovery, a machine makes this decision. Since people may not trust a machine to make significant judgement decisions that could put themselves or their organizations at risk, agents performing autonomous discovery are often limited to using private discovery services that list only those services that have been pre-screened and deemed trustworthy by the requester entity. This limited form of autonomous discovery would be more precisely called autonomous *selection*, since the available candidates are already known in advance. Two other ways to mitigate the trust issue in automated discovery include: (1) a agent could autonomously discover candidate Web services and then show them to the human user to choose; or (2) an agent could autonomously discover candidate services and then check a trusted registery for independent information about them, such as a Dunn and Bradstreet quality rating.

3.6.4.6 Secure Messaging

Secure Messaging ensures privacy, confidentiality and integrity of interactions. Digital signatures techniques can be used to help ensure non-repudiation.

Techniques that ensure channel security can be used for securing messages. However, such techniques are applicable in a few limited cases. Examples include a static direct connection between a requester agent and a provider agent. For some applications, such mechanisms can be appropriate. However, in the general case, message security techniques such as encryption and signing of the message payload can be used in routing and reliable messaging.

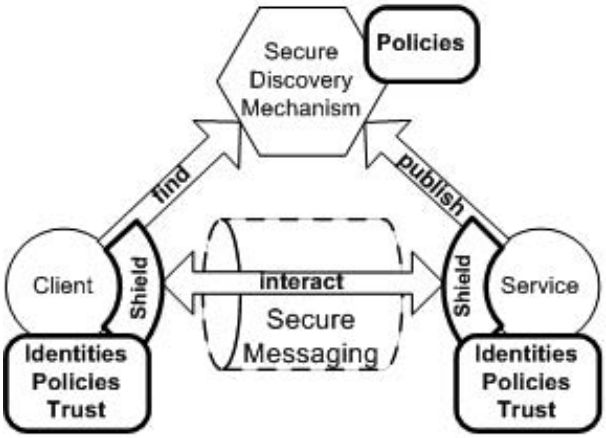


Figure 3-4. Secure Discovery

3.6.5 Privacy Considerations

Issue (privacy_needs_more_work):

The relationship between privacy and Web services technology needs clarification.

There is considerably more complexity to privacy than treated in this section.

Resolution:

None recorded.

Privacy as related to behavior, habits and actions are expressed in terms of policies that the owners of data — typically the users of Web services — have, together with mechanisms necessary to ensure that the owners' rights are respected.

Privacy policies are typically much more of the obligatory form than access control policies. A policy that requires a provider agent to properly propagate P3P tags, for example, represents an obligation on the

provider entity. However, it is not possible to prevent a rogue provider agent from leaking private information. Thus, it should be possible to monitor the public actions of the Web service to verify that the P3P tags are propagated appropriately.

Many privacy-related constraints are concerned with maintaining certain kinds of state. For example, a provider entity may have a constraint that any P3P tags associated with a use of one of its Web services are appropriately propagated to third parties. Such a constraint cannot easily expressed in terms of the allowed actions that the provider agent may perform. It is an obligation to ensure that the publicly observable condition (the proper use of P3P tags) is always maintained (presumably maintained in private also). Similarly, a provider agent may link the possible actions that a requester agent may perform to the requester agent maintaining a particular level of secure access (e.g., administrative tasks may only be performed if the request is using secure communications).

3.7 Peer-to-Peer Interaction

To support Web services interacting in a peer to peer style, the architecture must support peer to peer message exchange patterns, must permit Web services to have persistent identity, must permit descriptions of the capabilities of peers and must support flexibility in the discovery of peers by each other.

In the message exchange pattern concept we allow for Web services to communicate with each other using a very general concept of message exchange. Furthermore, we allow for the fact that a message exchange pattern can itself be identified — this permits interacting Web service agents to explicitly reference a particular message pattern in their interactions.

A Web service wishing to use a peer-to-peer style interaction may use, for example, a publish-subscribe form of message exchange pattern. This kind of message exchange is just one of the possible message exchange patterns possible when the pattern is explicitly identifiable.

In the agent concept we note that agents have identifiers. The primary role of an agent identifier is to permit long running interactions spanning multiple messages. Much like correlation, an agent's identifier can be used to link messages together. For example, in a publish and subscribe scenario, a publishing Web service may include references to the Web service that requested the subscription, separately from and additionaly to, the actual recipient of the service.

The agent concept also clarifies that a given agent may adopt the role of a provider agent and/or a requester agent. I.e., these are roles of an agent, not necessarily intrinsic to the agent itself. Such flexibility is a key part of the peer to peer mode of interaction between Web services.

In the service concept we state that services have a semantics that may be identified in aservice description and that may be expressed in a service description language. This identification of the semantics of a service, and for more advanced agents the description of the service contract itself, permits agents implementing Web services to determine the capabilities of other peer agents. This is turn, is a critical success factor in the architecture supporting peer-to-peer interaction of Web services.

Finally, the fact that services have descriptions means that these descriptions may be published in discovery agencies and also retrieved from such agencies. In effect, the availability of explicit descriptions enables Web services and agents to discover each other automatically as well as having these hard-coded.

3.8 Web Services Reliability

Dealing with errors and glitches is an inescapable fact of life, especially in the context of a global network linking services belonging to many different people. While we cannot eliminate errors and glitches, our goal is to both reduce the the error frequency for interactions and, where errors occur, to provide a greater amount of information about either successful or unsuccessful attempts at service.

Note that our focus on reliability is not really on issues such as syntax errors, or even badly written applications. There is sufficient scope for things to go wrong at the level of network connections being broken, servers being switched off and on in the middle of transactions and even people entering incorrect information in some description file.

In the context of Web services, we can address the issues of reliability at several distinct levels: the reliable and predictable delivery of infrastructure services, such as message transport and service discovery, of reliable and predictable interactions between services, and of the reliable and predictable behavior of individual requester and provider agents. This analysis is generally separate from concerns of fault tolerance, availability or security, but there may of course be overlapping issues.

In the context of security, deliberate acts can cause things to go wrong -- for example, denial of service attacks. This is a sufficiently important case that we deal with it in a separate section.

3.8.1 Message reliability

Reliability at the level of messages is often referred to as reliable messaging. In any distributed system there are fundamental limits to the reliability of communication between agents on a public network. However, in practice there are techniques that we can use to greatly increase the reliability of messages, and in those cases where communication fails then we can gain some feedback as to what went wrong.

In more detail, we identify two properties of message sending that are important: the sender of the message would like to be able to determine whether a given message has been received by its intended receiver and that the message has been received exactly once.

Knowing if a message has been received correctly allows the sender to take compensating action in the event the message has not been received. At the very least, the sender may attempt to resend a message that has not been received.

The general goal of reliable messaging is to define mechanisms that make it possible to achieve these objectives with a high probability of success in the face of inevitable but unpredictable network, system and software failures.

This goal may also be examined with respect to whether one wishes to confirm only the receipt of a message, or perhaps also to confirm the validity of that message. Three questions may be asked about message validity:

1. Was the message received the same as the one sent? This may be determined by such techniques as byte counts, check sums, digital signatures.
2. Does the message conform to the formats specified by the agreed upon protocol for the message? Typically determined by automatic systems using syntax constraints (e.g. XML well formed) and structural constraints (validate against one or more XML schemas or WSDL message definitions).
3. Does the message conform to the business rules expected by the receiver? For this purpose additional constraints and validity checks related to the business process are typically checked by application logic and/or human process managers.

Of these, the first is considered to be part of reliable messaging, the last is partly addressed by Web service choreography, but is more closely related to the business expectations of the parties.

The Web services architecture does not itself give specific support for reliable messaging, or for reporting in the event of failure. However, it does give guidance as to how this may be accomplished. The headers and body structure of messages can be utilized: by providing standardized headers to support message auditing then message reliability infrastructures can be deployed in ways that do not need to impact applications and services.

In effect, we can augment message traffic as necessary with specific headers and intermediaries that

implement specific semantics for message reliability and reporting in the case that message communication fails. Recall that the architecture does not itself mandate a specific means of message delivery. In fact, we envisage many potential modes of communication, including HTTP, SMTP, JMS based message transports. A given message may even involve multiple kinds of message transport. However, since all messages are structured according to SOAP, we can incorporate overall message reliability within the SOAP message structure.

Message reliability is most often achieved via an acknoweldgement infrastructure, which is a set of rules defining how the parties to a message should communicate with each other concerning the receipt of that message and its validity. WS-Reliability and WS-ReliableMessaging are examples of specifications for an acknowledgement infrastructure that leverage the SOAP Extensibility Model. In cases where the underlying transport layer already provides reliable messaging support (e.g. a queue-based infrastructure), the same level of reliability can be achieved in SOAP by defining a binding that relies on the underlying properties of the transport.

3.8.2 Service reliability

As with message reliability, we are not in a position to be able to offer guarantees that service provider agents and/service requester agents will always perform flawlessly; again, especially in the context of a distributed system over a public network where the different agents may be owned by different people and subject to different policies and management it is not possible to engineer complete service reliability. However, as with message communication we can deploy techniques that greatly enhance reliability and reduce the cost of failure. The principal technique here is one of transactional context management.

Transaction management allows conversations between agents to be managed so that all the parties involved have a greater degree of confidence that the transactions between them progress satisfactorily, and in the event of failure the failure may be identified and transactions either cancelled, rolled back or compensated for.

The architecture does not give specific advice on how to implement transactional reliability. However, again as with message reliability, the combination of the flexible and extensible message structures and the concept of multiple processing of messages (via intermediaries implementing service roles) gives us guidance.

One way to incorporate transactional support would be to use standardized headers containing information such as transactional bracket markers and context information that are added to messages exchanged between service requester agents and service provider agents in such a way that *intermediaries* can process messages and monitor transactions in a way that only minimally impacts existing applications. Specialized transactional intermediaries could process messages' transaction-specific headers (such as beginning of transaction, commitment, roll-back and so on) and mark messages that they process with the results; so that applications can respond appropriately.

Related to transactional monitoring is the monitoring of service choreographies. A significant aspect of the specification of the interface of a service is the pattern of message traffic that one might see. For simple cases, this pattern is often very straightforward; however, for most realistic cases, the choreography of services can be very complex. Monitoring that messages are arriving in the order expected is potentially a significant tool in the deployment of reliable services.

Again, as with transactional monitoring, one approach would be to deploy specialized intermediary processes whose specific function is to ensure that the choreographic as well as the static (i.e., message structure) requirements of service usage are being met. This is especially important when the provider agent of a service is not in the same ownership domain as the requester agent.

The key architectural property being used here is the potential deployment of third party services that monitor and process messages in specific role-oriented ways that neither the requesters of services nor the providers of serives needs to be unduly concerned with. This is possible because the architecture does not require messages to be consumed by single agents — nor conversely to be produced by single

agents — but allows multiple agents to *collaborate* in the processing of a given message. Each service role establishes a specific functionality, often encoded in specific headers of the messages.

3.8.3 Reliability and management

The reliability of the individual requester and provider agents is out of scope of this architecture as we do not comment on the realization of Web services. In some cases reliability at this level can be enhanced by provider entities adopting deployment platforms that have strong management capabilities. Note that platform manageability represents a *different* perspective than the notion of management identified in Service Management (below), which focuses on the manageability of services from a peer or business-partner perspective.

3.9 Web Service Management

Web service management is the management of Web services through a set of management capabilities that enable monitoring, controlling, and reporting of, service qualities and service usage. Such service qualities include health qualities such as availability (presence and number of service instances) and performance (e.g. access latency and failure rates), and also accessibility (of endpoints). Facets of service usage information that may be managed include frequency, duration, scope, functional extent, and access authorization.

A Web service becomes manageable when it exposes a set of management operations that support management capabilities. These management capabilities realize their monitoring, controlling and reporting functions with the assistance of a management information model that models various types of service usage and service quality information associated with management of the Web service. Typical information types include request and response counts, begin and end timers, lifecycle states, entity identifiers (e.g. of senders, receivers, contexts, messages, etc.).

Although the provision of management capabilities enables a Web service to become manageable, the extent and degree of permissible management are defined in management policies that are associated with the Web service. Management policies therefore are used to define the obligations for, and permissions to, managing the Web service.

Just as the Web service being managed needs to have common service semantics that are understood by both the requester and provider entities, Web service management also requires common management semantics, in relation to management policies and management capabilities, to be understood by the requester and provider entities.

Figure 3-5 illustrates how the concepts of service, policy and capability defined in this architecture can be applied to management.

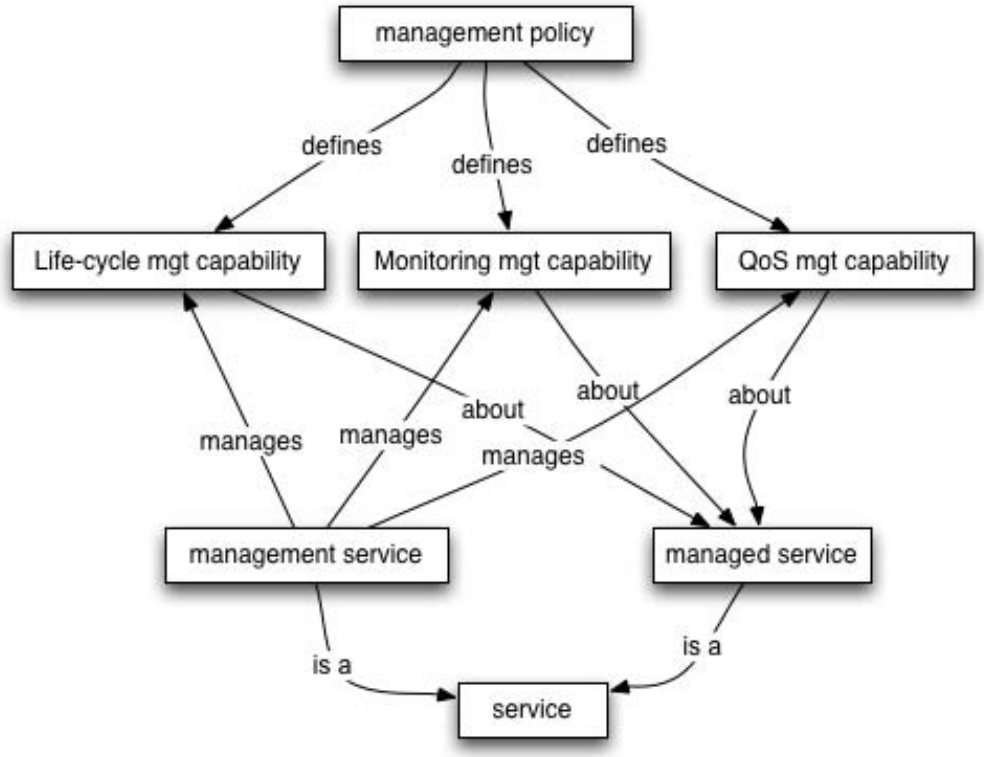


Figure 3-5. Management Concepts and Relationships

More detailed information about Web services management is available in the management documents that were produced by the Management Task Force of this Working Group.

3.10 Web Services and EDI: Transaction Tracking

One of the basic assumptions that many people make about the role of Web services is that they will be used for functions similar to those presently provided by Electronic Data Interchange (EDI). Since EDI is a well established technology, it is useful to examine the expectations that current EDI users may have for a technology that is to be used as a replacement. That is, what do they do now that they will also expect from a new technology? The most basic of these expectations concern security, message reliability and a function that we will call "tracking". Since security and message reliability are covered elsewhere in this document, this section will focus on tracking.

3.10.1 When Something Goes Wrong

What happens when a transaction goes awry for reasons other than the loss of a message or security violations? Although it is possible, and useful, to automate safeguards both at the protocol and application level, experience indicates that there is a virtually limitless variety of ways that business transactions can fail. Informal interviews with current EDI practitioners have indicated that in practice the 80-20 of what EDI people actually do is involved with the issue of finding out what has actually happened in transactions when something has gone wrong. For example, such an interaction may start with a phone call that goes something like, "Why haven't you paid us?" and continues, "We think we have paid you". In these cases there is often a good faith desire on both sides to figure out what has happened and comply with the requirements of the transaction, but the information that people are working with may differ and coming to a common understanding can take some work.

3.10.2 The Need for Tracking

In current EDI operations, many of the questions that must be answered in these cases can be handled in an automated fashion by the vendor of the proprietary network used in the transactions. For example, if company A asks if the invoice to company B was delivered, the vendor can access its records,

probably from a central repository, and respond, "The message was delivered to company B's mailbox on Dec 24 but they have not as yet downloaded the message". Queries of this sort are relatively easy to satisfy in this environment because the vendor is in control of all aspects of the communication. In a Web services scenario, where the transactions take place in a distributed environment, with no central authority, some other means must replace the current automated queries to the EDI vendor or this important tracking capability will be lost.

One possibility would be to provide some kind of uniform tracking interface. The basic requirement here is for companies that are cooperating in a business transaction to find out at any time what is the status and history of the transaction. Significant complexity is added by the fact that multiple companies may be involved. That is, company A may initiate a transaction by sending a message to B, but the process may then involve messages between B and C. In some cases the interactions between B and C may be known to A (as opposed to being part of B's internal process that is opaque to A). It is not immediately clear whether this should be handled by A querying both B and C, or if a responding to a query from A to B should carry with it the obligation to query C and return the results. This is presumably an issue which must be ironed out in the creation of the specification(s) for the uniform interface.

3.10.3 Examples of Tracking

As illustrations, here are some of the typical queries that A might send to B or C. Web Services Usage Scenarios [WSAUS] contains additional examples.

1. (Query to B) Did you receive and process message M from A?
2. (Query to B) Please return copies of all messages associated with Transaction T.
3. (Query to B) Please return copies of all messages between A and B in a given time range.
4. (Query to C) Please return all messages associated with transactions involving A during a given time frame (including messages between B and C related to transactions in which A is involved).
5. (Query to B) Please return copies of messages between B and other companies involved with a transaction (or all transactions in a date range).

Of course, in all cases, the party performing the query must be authorized to receive the information.

Current EDI practices may automate some of these queries; others may involve manual processing. In general, however, there are significant cost savings to be realized by automating as much of the process as possible.

3.10.4 Requirements for Effective Tracking

In order to help automate the tracking process, there are various requirements, some of which are probably achievable using current or planned specifications and others of which may require new ones:

1. A uniform, interoperable interface for tracking queries, so that company A can send a standard query to all of its business partners. This interface should be associated with the functional Web services interfaces. For example, such an interface might be implemented as part of a management interface.
2. Standard identifiers for transactions and individual messages that are necessary to define the queries. Note that some of these queries involve identifiers of participants in a transaction other than sender and receiver of a particular message. There are clearly aspects of this requirement that are related to the choreography domain.
3. Policies controlling whether party A is authorized to make tracking queries to B. There may be several variants of such policies: e.g. a can query B about messages directly between A and B but not messages between B and C associated with transactions involving A and so on. It may be possible to establish these policies using mechanisms currently available or under development in the security or policy domain, or there may be transactional aspects to these policies that are not currently being considered.
4. A method to establish the trust relationships necessary to implement the policies in 3.

3.10.5 Tracking and URIs

One of the important connections between Web services architecture and Web architecture as a whole, is the common use of URIs. Although URIs are important to many aspects of Web services, it is particularly worth noting their potential role and benefit in indentifying and tracking transactions in Web services.

As a simple example to illustrate this benefit, suppose URIs are used as transaction identifiers. Each time a new transaction is initiated, a new URI is generated to unambiguously identify that transaction, much like a primary key in a database. However, while a database key may only be unambiguous within a particular database, a URI is *globally unambiguous*, which means that it can be conveniently transmitted to others without loss or confusion of meaning.

Furthermore, a URI may be *dereferenceable*: If the URI also represents the location of a document (or a dynamic query into a database), it could act as a convenient link for determining the status or history of that transaction, provided the user is authorized to access such information. (Security mechanisms will need to ensure that a tracking URI cannot be dereferenced without proper authority and privacy controls, but the use of URIs is largely orthogonal to this requirement.)

The potential value of this dual use of URIs — both as globally unambiguous identifiers and as universally dereferenceable links — is one of the most fundamental and important insights in the architecture of the Web. Because the Web services architecture builds on the Web architecture, Web services can leverage the benefits of clarity, simplicity, universality and convenience that this use of URI offers.

This is not to say that Web services tracking *must* be done using URIs in this way. Indeed, there are other ways tracking can be performed, and any engineering design must take many factors into consideration. Rather, the point is to illuminate the fact that, because Web services architecture is based on Web architecture, Web services have the *possibility* of taking advantage of this use of URIs.

4 Conclusions

4.1 Requirements Analysis

We believe this architecture substantially meets the requirements defined in [WSA Reqs], with the exception of security and privacy. Although this architecture contains substantial material that lays the foundation for addressing these, more work is needed. The Working Group wanted to do more to address these but was not able to do so with the available resources.

4.2 Value of This Work

This architecture lays the conceptual foundation for establishing interopable Web services. The architecture identifies a number of important abstractions and their interdependencies.

Contributions of this work include the following:

- Provides a coherent framework that allows specific technologies to be considered in a logical context and facilitates the work of specification writers and architects.
- Defines a consistent vocabulary, including an authoritative definition of "Web service" that has received widespread acceptance in industry [WS Glossary].
- Defines an OWL ontology of Web services architecture concepts [OWLO].
- Distinguishes SOA from distributed object architecture.
- Clarifies the architectural relationship between the Web and Web services
- Clarifies the relationship between Web services and REST.
- Identifies gaps and inconsistencies in existing Web services specifications.
- Identifies the role of semantics and the need for machine-processable semantics and ontologies in Web services

4.3 Significant Unresolved Issues

(See also the issues list previously maintained by the Working Group.)

1. What is the difference between an MEP and a Choreography? [See **2.3.1.7 Message Exchange Pattern (MEP)**]
2. What should be the representation returned by an HTTP "GET" on a Web service URI? [See **2.3.2.10 Service**]
3. Should URIs be used to identify Web services components, rather than QNames? [See **2.3.3.3 Identifier**]
4. The relationship between privacy and Web services technology needs clarification. [See **3.6.5 Privacy Considerations**]
5. SOAP 1.2 and this architecture introduce the concept of "intermediaries", but this concept is not represented in WSDL 2.0.
6. What happens if two logical WSDL documents define the same service differently? [See email thread available at http://lists.w3.org/Archives/Public/www-ws-desc/2003Dec/0045.html]
7. The relationship between conversations, correlations and transactions and choreography is unclear and needs more work.
8. There is a need for consistent tracking mechanisms in Web services. [See **3.10 Web Services and EDI: Transaction Tracking**]

A Overview of Web Services Specifications (Non-Normative)

An annotated list of Web services specifications (available at http://lists.w3.org/Archives/Public/www-ws-arch/2004Feb/0022.html) was produced independently by two members of this Working Group, Roger Cutler and Paul Denning. *Although this Working Group feels that this is a useful list, the opinions expressed therein are the personal opinions of those authors and do not represent the consensus of the Working Group.*

B An Overview of Web Services Security Technologies (Non-Normative)

This section attempts to provide a non-exhaustive description of current available work around Web services security relevant to the requirements and solutions presented in **3.6 Web Services Security**.

Note that although these technologies build on existing security technologies, they are relatively new and need to be fully tested in actual deployment scenarios.

B.1 XML-Signature and XML-Encryption

XML signatures are designed for use in XML transactions. It is a standard that was jointly developed by W3C and the IETF (RFC 2807, RFC 3275). The standard defines a schema for capturing the result of a digital signature operation applied to arbitrary data and its processing. XML signatures add authentication, data integrity, and support for non-repudiation to the signed data.

XML Signature has the ability to sign only specific portions of the XML tree rather than the complete document. This is important when a single XML document may need to be signed by multiple times by a single or multiple parties. This flexibility can ensure the integrity of certain portions of an XML document, while leaving open the possibility for other portions of the document to change. Signature validation mandates that the data object that was signed be accessible to the party that interested in the transaction. The XML signature will generally indicate the location of the original signed object.

XML Encryption specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption element which contains or references the cipher data.

B.2 Web Services Security

Developed at OASIS, Web Services Security (WSS) defines a SOAP extension providing quality of protection through message integrity, message confidentiality, and message authentication. WSS mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

The work provides a general mechanism for associating security tokens with messages. The specification does not require a specific type of security token. It is designed to support multiple security token formats. WSS describes how to encode binary security tokens. The specification describes how to encode X.509 certificates and Kerberos tickets. Additionally, it also describes how to include opaque encrypted keys.

The WSS specification defines an end to end security framework that provides support for intermediary security processing. Message integrity is provided by using XML Signature in conjunction with security tokens to ensure that messages are transmitted without modifications. The integrity mechanisms can support multiple signatures, possibly by multiple actors. The techniques are extensible such that they can support additional signature formats. Message confidentiality is granted by using XML Encryption in conjunction with security tokens to keep portions of SOAP messages confidential. The encryption mechanisms can support operations by multiple actors.

B.3 XML Key Management Specification (XKMS) 2.0

XKMS 2.0 is an XML-based way of managing the Public Key Infrastructure (PKI), a system that uses public-key cryptography for encrypting, signing, authorizing and verifying the authenticity of information in the Internet. It specifies protocols for distributing and registering public keys, suitable for use in conjunction with the proposed standard for XML Signature and XML Encryption.

XKMS allow implementers to outsource the task of key registration and validation to a "trust" utility. This simplify implementation since the actual work of managing public and private key pairs and other PKI details is done by third party.

An XKMS trust utility works with any PKI system, passing the information back and forth between it and the Web service. Since the trust utility does the work, the Web service itself can be kept simple. XKMS is a W3C specification.

B.4 Security Assertion Markup Language (SAML)

SAML is an Extensible Markup Language standard (XML) that supports Single Sign On. SAML allows a user to log on once to a Web site and conduct business with affiliated but separate Web sites. SAML can be used in business-to-business and business-to-consumer transactions.

There are threes basic SAML components: assertions, protocol, and binding. Assertions can be one of three types: authentication, attribute, and authorization. Authentication assertion validates the identity of the user. The attribute assertion contains specific information about the user. While, the authorization assertion identifies what the user is authorized to do.

The protocol defines how SAML request and receives assertions. There are several available binding for SAML. There are bindings that define how SAML message exchanges are mapped to SOAP, HTTP, SMTP and FTP among others. The Organization for the Advancement of Structured Information Standards (OASIS) is the body developing SAML.

B.5 XACML: Communicating Policy Information

XACML is an Extensible Markup Language standard (XML) based technology, developed by Organization for the Advancement of Structured Information Standards (OASIS) for writing access

control polices for disparate devices and applications.

XACML includes an access control language and request/response language that let developers write policies that determine what users can access on a network or over the Web. XACML can be used to connect disparate access control policy engines.

B.6 Identity Federation

The Liberty Alliance is defining specifications dealing with various aspects of identity. Their phase 2 work is grouped into three categories: ID-FF, ID-WSF, and ID-SIS.

ID-FF (Identity Federation Framework) discusses how businesses or organizations can be affiliated into circles of trust and trust relationships. ID-FF includes several normative specifications, which in turn make normative references to SAML.

ID-WSF (Identity Web Services Framework) is a set of specifications for creating, discovering, using, and updating various aspects of identities through a particular type of web service known as an Identity Service. ID-WSF builds on ID-FF. A user (Principal) may register with several Identity Services. A prominent part of ID-WSF is a discovery service for locating an Identity Service for a given user (Principal). ID-SWF also defines a Data Services Template. ID-WSF has also defined a draft specification for an approach to negotiating an authentication method using SOAP messages to identify SASL mechanisms (RFC 2222).

Note that WS-Security specifically states that establishing a security context or authentication mechanisms is outside its scope. ID-WSF may fill this void. However, WS-Security also defines a Username Token Profile, which could be used as an authentication mechanism. Potentially, Liberty ID-WSF could be used to negotiate the use of WSS Username Token Profile as the authentication mechanism. Currently, WSS Username Token Profile is not registered in IANA's SASL Mechanisms collection.

ID-SIS (Identity Service Instance Specifications) defines profiles for particular types of Identity Services. These profiles conform to the ID-WSF Data Services Template. Liberty has defined two such profiles. The Employee Profile (ID-SIS-EP) defines how to query and modify information associated with a Principal in the context of their employer. The Personal Profile (ID-SIS-PP) defines how to query and modify identity information for Principals themselves.

C References (Non-Normative)

Dist Comp
A Note on Distributed Computing, S. C. Kendall, J. Waldo, A. Wollrath, G. Wyant, November 1994 (See http://research.sun.com/techrep/1994/abstract-29.html.)

Fielding
Architectural Styles and the Design of Network-based Software Architectures, PhD. Dissertation, R. Fielding, 2000 (See http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.)

OWLO
OWL Ontology of Web service architecture concepts, M. Paolucci, N. Srinivasan, K. Sycara (See http://www.w3.org/2004/02/wsa/.)

RFC 2396
Uniform Resource Identifiers (URI): Generic Syntax, IETF RFC 2396, T. Berners-Lee, R. Fielding, L. Masinter, August 1998 (See http://ietf.org/rfc/rfc2396.txt.)

SOAP 1.2 Part 1
SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation, M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Nielsen, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.)

SOAP 1.2 Part 2
SOAP Version 1.2 Part 2: Adjuncts, W3C Recommendation, M. Gudgin, M. Hadley, N.

Mendelsohn, J-J. Moreau, H. Nielsen, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part2-20030624/.)

Web Arch

Architecture of the World Wide Web, First Edition, W3C Working Draft, I. Jacobs, 9 December 2003 (See http://www.w3.org/TR/2003/WD-webarch-20031209/.)

WS Glossary

Web Services Glossary, W3C Working Group Note, H. Haas, A.Brown, 11 Febuary 2004 (See http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/.)

WSA Reqs

Web Services Architecture Requirements, W3C Working Group Note, D. Austin, A. Barbir, C. Ferris, S. Garg, 11 February 2004 (See http://www.w3.org/TR/2004/NOTE-wsa-reqs-20040211.)

WSAUS

Web Services Architecture Usage Scenarios, W3C Working Group Note, H. He, H. Haas, D. Orchard, 11 February 2004 (See http://www.w3.org/TR/2004/NOTE-ws-arch-scenarios-20040211/.)

WSDL 2.0 Part 1

Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Working Draft, R. Chinnici, M. Gudgin, J-J. Moreau, J. Schlimmer, S. Weerawarana, 10 November 2003 (See http://www.w3.org/TR/2003/WD-wsdl20-20031110/.)

XML 1.0

Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler. 6 October 2000 (See http://www.w3.org/TR/2000/REC-xml-20001006.)

XML Infoset

XML Information Set, W3C Recommendation, J. Cowan, R. Tobin, 24 October 2001 (See http://www.w3.org/TR/2001/REC-xml-infoset-20011024/.)

D Acknowledgments (Non-Normative)

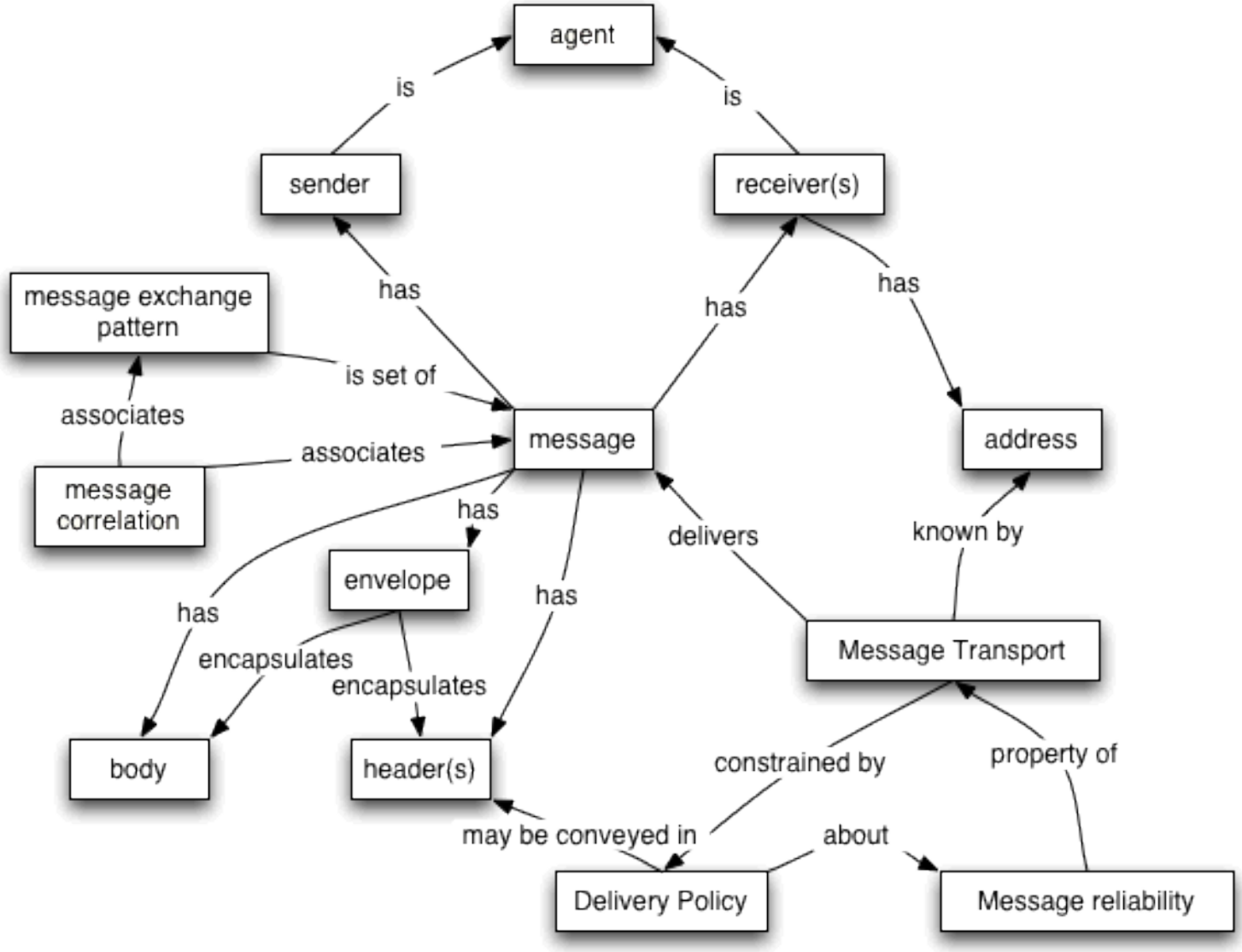
This document has been produced by the Web Services Architecture Working Group . The chairs of this Working Group were Chris Ferris (until July 2002), Michael Champion (starting July 2002) and Dave Hollander (starting July 2002). The chairs also wish to thank the following (listed in alphabetic order) for their substantial contributions to the final documents: Daniel Austin, Mark Baker, Abbie Barbir, David Booth, Martin Chapman, Ugo Corda, Roger Cutler, Paul Denning, Zulah Eckert, Chris Ferris, Hugo Haas, Hao He, Yin-Leng Husband, Mark Jones, Heather Kreger, Michael Mahan, Frank McCabe, Eric Newcomer, David Orchard, Katia Sycara.

Members of the Working Group are (at the time of writing, and in alphabetical order): Geoff Arnold (Sun Microsystems, Inc.), Mukund Balasubramanian (Infravio, Inc.), Mike Ballantyne (EDS), Abbie Barbir (Nortel Networks), David Booth (W3C), Mike Brumbelow (Apple), Doug Bunting (Sun Microsystems, Inc.), Greg Carpenter (Nokia), Tom Carroll (W. W. Grainger, Inc.), Alex Cheng (Ipedo), Michael Champion (Software AG), Martin Chapman (Oracle Corporation), Ugo Corda (SeeBeyond Technology Corporation), Roger Cutler (ChevronTexaco), Jonathan Dale (Fujitsu), Suresh Damodaran (Sterling Commerce(SBC)), James Davenport (MITRE Corporation), Paul Denning (MITRE Corporation), Gerald Edgar (The Boeing Company), Shishir Garg (France Telecom), Hugo Haas (W3C), Hao He (The Thomson Corporation), Dave Hollander (Contivo), Yin-Leng Husband (Hewlett-Packard Company), Mario Jeckle (DaimlerChrysler Research and Technology), Heather Kreger (IBM), Sandeep Kumar (Cisco Systems Inc), Hal Lockhart (OASIS), Michael Mahan (Nokia), Francis McCabe (Fujitsu), Michael Mealling (VeriSign, Inc.), Jeff Mischkinsky (Oracle Corporation), Eric Newcomer (IONA), Mark Nottingham (BEA Systems), David Orchard (BEA Systems), Bijan Parsia (MIND Lab), Adinarayana Sakala (IONA), Waqar Sadiq (EDS), Igor Sedukhin (Computer Associates), Hans-Peter Steiert (DaimlerChrysler Research and Technology), Katia Sycara (Carnegie Mellon University), Bryan Thompson (Hicks & Associates, Inc.), Sinisa Zimek (SAP).

Previous members of the Working Group were: Assaf Arkin (Intalio, Inc.), Daniel Austin (W. W. Grainger, Inc.), Mark Baker (Idokorro Mobile, Inc. / Planetfred, Inc.), Tom Bradford (XQRL, Inc.), Allen Brown (Microsoft Corporation), Dipto Chakravarty (Artesia Technologies), Jun Chen (MartSoft Corp.), Alan

Davies (SeeBeyond Technology Corporation), Glen Daniels (Macromedia), Ayse Dilber (AT&T), Zulah Eckert (Hewlett-Packard Company), Colleen Evans (Sonic Software), Chris Ferris (IBM), Daniela Florescu (XQRL Inc.), Sharad Garg (Intel), Mark Hapner (Sun Microsystems, Inc.), Joseph Hui (Exodus/ Digital Island), Michael Hui (Computer Associates), Nigel Hutchison (Software AG), Marcel Jemio (DISA), Mark Jones (AT&T), Timothy Jones (CrossWeave, Inc.), Tom Jordahl (Macromedia), Jim Knutson (IBM), Steve Lind (AT&T), Mark Little (Arjuna), Bob Lojek (Intalio, Inc.), Anne Thomas Manes (Systinet), Jens Meinkoehn (T-Nova Deutsche Telekom Innovationsgesellschaft), Nilo Mitra (Ericsson), Don Mullen (TIBCO Software, Inc.), Himagiri Mukkamala (Sybase, Inc.), Joel Munter (Intel), Henrik Frystyk Nielsen (Microsoft Corporation), Duane Nickull (XML Global Technologies), David Noor (Rogue Wave Software), Srinivas Pandrangi (Ipedo), Kevin Perkins (Compaq), Mark Potts (Talking Blocks, Inc.), Fabio Riccardi (XQRL, Inc.), Don Robertson (Documentum), Darran Rolls (Waveset Technologies, Inc.), Krishna Sankar (Cisco Systems Inc), Jim Shur (Rogue Wave Software), Patrick Thompson (Rogue Wave Software), Steve Vinoski (IONA), Scott Vorthmann (TIBCO Software, Inc.), Jim Webber (Arjuna), Prasad Yendluri (webMethods, Inc.), Jin Yu (MartSoft Corp.) .

The people who have contributed to discussions on the www-ws-arch public mailing list are also gratefully acknowledged.


agent
is
is
sender
receiver(s)
message exchange pattern
has
has
has
is set of
associates
associates
message
address
message correlation
has
delivers
known by
envelope
has
has
encapsulates
encapsulates
Message Transport
body
header(s)
constrained by
property of
may be conveyed in
about
Delivery Policy
Message reliability

javax.jms

Interface Connection

All Known Subinterfaces:

QueueConnection, TopicConnection, XAConnection, XAQueueConnection, XATopicConnection

public interface **Connection**

A `Connection` object is a client's active connection to its JMS provider. It typically allocates provider resources outside the Java virtual machine (JVM).

Connections support concurrent use.

A connection serves several purposes:

- It encapsulates an open connection with a JMS provider. It typically represents an open TCP/IP socket between a client and the service provider software.
- Its creation is where client authentication takes place.
- It can specify a unique client identifier.
- It provides a `ConnectionMetaData` object.
- It supports an optional `ExceptionListener` object.

Because the creation of a connection involves setting up authentication and communication, a connection is a relatively heavyweight object. Most clients will do all their messaging with a single connection. Other more advanced applications may use several connections. The JMS API does not architect a reason for using multiple connections; however, there may be operational reasons for doing so.

A JMS client typically creates a connection, one or more sessions, and a number of message producers and consumers. When a connection is created, it is in stopped mode. That means that no messages are being delivered.

It is typical to leave the connection in stopped mode until setup is complete (that is, until all message consumers have been created). At that point, the client calls the connection's `start` method, and messages begin arriving at the connection's consumers. This setup convention minimizes any client confusion that may result from asynchronous message delivery while the client is still in the process of setting itself up.

A connection can be started immediately, and the setup can be done afterwards. Clients that do this must be prepared to handle asynchronous message delivery while they are still in the process of setting up.

A message producer can send messages while a connection is stopped.

Version:

1.1 - February 1, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`ConnectionFactory`, `QueueConnection`, `TopicConnection`

Method Summary

void	**close**() Closes the connection.
ConnectionConsumer	**createConnectionConsumer**(Destination destination, String messageSelector, ServerSessionPool sessionPool, int maxMessages) Creates a connection consumer for this connection (optional operation).
ConnectionConsumer	**createDurableConnectionConsumer**(Topic topic, String subscriptionName, String messageSelector, ServerSessionPool sessionPool, int maxMessages) Create a durable connection consumer for this connection (optional operation).
Session	**createSession**(boolean transacted, int acknowledgeMode) Creates a Session object.
String	**getClientID**() Gets the client identifier for this connection.
ExceptionListener	**getExceptionListener**() Gets the ExceptionListener object for this connection.
ConnectionMetaData	**getMetaData**() Gets the metadata for this connection.
void	**setClientID**(String clientID) Sets the client identifier for this connection.
void	**setExceptionListener**(ExceptionListener listener) Sets an exception listener for this connection.
void	**start**() Starts (or restarts) a connection's delivery of incoming messages.
void	**stop**() Temporarily stops a connection's delivery of incoming messages.

Method Detail

createSession

```
public Session createSession(boolean transacted,
                             int acknowledgeMode)
                      throws JMSException
```

Creates a Session object.

Parameters:

transacted - indicates whether the session is transacted

acknowledgeMode - indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, and Session.DUPS_OK_ACKNOWLEDGE.

Returns:

a newly created session

Throws:

JMSException - if the Connection object fails to create a session due to some internal error or lack of support for the specific transaction and acknowledgement mode.

Since:
1.1
See Also:
`Session.AUTO_ACKNOWLEDGE`, `Session.CLIENT_ACKNOWLEDGE`, `Session.DUPS_OK_ACKNOWLEDGE`

---

getClientID

```
public String getClientID()
                  throws JMSException
```

Gets the client identifier for this connection.

This value is specific to the JMS provider. It is either preconfigured by an administrator in a `ConnectionFactory` object or assigned dynamically by the application by calling the `setClientID` method.

Returns:
the unique client identifier
Throws:
`JMSException` - if the JMS provider fails to return the client ID for this connection due to some internal error.

---

setClientID

```
public void setClientID(String clientID)
                 throws JMSException
```

Sets the client identifier for this connection.

The preferred way to assign a JMS client's client identifier is for it to be configured in a client-specific `ConnectionFactory` object and transparently assigned to the `Connection` object it creates.

Alternatively, a client can set a connection's client identifier using a provider-specific value. The facility to set a connection's client identifier explicitly is not a mechanism for overriding the identifier that has been administratively configured. It is provided for the case where no administratively specified identifier exists. If one does exist, an attempt to change it by setting it must throw an `IllegalStateException`. If a client sets the client identifier explicitly, it must do so immediately after it creates the connection and before any other action on the connection is taken. After this point, setting the client identifier is a programming error that should throw an `IllegalStateException`.

The purpose of the client identifier is to associate a connection and its objects with a state maintained on behalf of the client by a provider. The only such state identified by the JMS API is that required to support durable subscriptions.

If another connection with the same `clientID` is already running when this method is called, the JMS provider should detect the duplicate ID and throw an `InvalidClientIDException`.

Parameters:
`clientID` - the unique client identifier
Throws:
`JMSException` - if the JMS provider fails to set the client ID for this connection due to some internal error.
`InvalidClientIDException` - if the JMS client specifies an invalid or duplicate client ID.
`IllegalStateException` - if the JMS client attempts to set a connection's client ID at the wrong time or when it has been administratively configured.

---

getMetaData

```
public ConnectionMetaData getMetaData()
                            throws JMSException
```

Gets the metadata for this connection.

Returns:
the connection metadata
Throws:
`JMSException` - if the JMS provider fails to get the connection metadata for this connection.
See Also:
`ConnectionMetaData`

---

getExceptionListener

```
public ExceptionListener getExceptionListener()
                                      throws JMSException
```

Gets the `ExceptionListener` object for this connection. Not every `Connection` has an `ExceptionListener` associated with it.

Returns:
the `ExceptionListener` for this connection, or null. if no `ExceptionListener` is associated with this connection.
Throws:
`JMSException` - if the JMS provider fails to get the `ExceptionListener` for this connection.
See Also:
`setExceptionListener(javax.jms.ExceptionListener)`

---

setExceptionListener

```
public void setExceptionListener(ExceptionListener listener)
                          throws JMSException
```

Sets an exception listener for this connection.

If a JMS provider detects a serious problem with a connection, it informs the connection's `ExceptionListener`, if one has been registered. It does this by calling the listener's `onException` method, passing it a `JMSException` object describing the problem.

An exception listener allows a client to be notified of a problem asynchronously. Some connections only consume messages, so they would have no other way to learn their connection has failed.

A connection serializes execution of its `ExceptionListener`.

A JMS provider should attempt to resolve connection problems itself before it notifies the client of them.

Parameters:
`listener` - the exception listener

Throws:
JMSException - if the JMS provider fails to set the exception listener for this connection.

---

start

```
public void start()
            throws JMSException
```

Starts (or restarts) a connection's delivery of incoming messages. A call to `start` on a connection that has already been started is ignored.

Throws:
JMSException - if the JMS provider fails to start message delivery due to some internal error.
See Also:
stop()

---

stop

```
public void stop()
            throws JMSException
```

Temporarily stops a connection's delivery of incoming messages. Delivery can be restarted using the connection's `start` method. When the connection is stopped, delivery to all the connection's message consumers is inhibited: synchronous receives block, and messages are not delivered to message listeners.

This call blocks until receives and/or message listeners in progress have completed.

Stopping a connection has no effect on its ability to send messages. A call to `stop` on a connection that has already been stopped is ignored.

A call to `stop` must not return until delivery of messages has paused. This means that a client can rely on the fact that none of its message listeners will be called and that all threads of control waiting for `receive` calls to return will not return with a message until the connection is restarted. The receive timers for a stopped connection continue to advance, so receives may time out while the connection is stopped.

If message listeners are running when `stop` is invoked, the `stop` call must wait until all of them have returned before it may return. While these message listeners are completing, they must have the full services of the connection available to them.

Throws:
JMSException - if the JMS provider fails to stop message delivery due to some internal error.
See Also:
start()

---

close

```
public void close()
            throws JMSException
```

Closes the connection.

Since a provider typically allocates significant resources outside the JVM on behalf of a connection, clients should close these resources when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

There is no need to close the sessions, producers, and consumers of a closed connection.

Closing a connection causes all temporary destinations to be deleted.

When this method is invoked, it should not return until message processing has been shut down in an orderly fashion. This means that all message listeners that may have been running have returned, and that all pending receives have returned. A close terminates all pending message receives on the connection's sessions' consumers. The receives may return with a message or with null, depending on whether there was a message available at the time of the close. If one or more of the connection's sessions' message listeners is processing a message at the time when connection `close` is invoked, all the facilities of the connection and its sessions must remain available to those listeners until they return control to the JMS provider.

Closing a connection causes any of its sessions' transactions in progress to be rolled back. In the case where a session's work is coordinated by an external transaction manager, a session's `commit` and `rollback` methods are not used and the result of a closed session's work is determined later by the transaction manager. Closing a connection does NOT force an acknowledgment of client-acknowledged sessions.

Invoking the `acknowledge` method of a received message from a closed connection's session must throw an `IllegalStateException`. Closing a closed connection must NOT throw an exception.

Throws:

`JMSException` - if the JMS provider fails to close the connection due to some internal error. For example, a failure to release resources or to close a socket connection can cause this exception to be thrown.

---

createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer(Destination destination,
                                                   String messageSelector,
                                                   ServerSessionPool sessionPool,
                                                   int maxMessages)
                                            throws JMSException
```

Creates a connection consumer for this connection (optional operation). This is an expert facility not used by regular JMS clients.

Parameters:

`destination` - the destination to access
`messageSelector` - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
`sessionPool` - the server session pool to associate with this connection consumer
`maxMessages` - the maximum number of messages that can be assigned to a server session at one time

Returns:

the connection consumer

Throws:

`JMSException` - if the `Connection` object fails to create a connection consumer due to some internal error or invalid arguments for `sessionPool` and `messageSelector`.
`InvalidDestinationException` - if an invalid destination is specified.
`InvalidSelectorException` - if the message selector is invalid.

Since:

1.1

See Also:
ConnectionConsumer

---

createDurableConnectionConsumer

```
public ConnectionConsumer createDurableConnectionConsumer(Topic topic,
                                                          String subscriptionName,
                                                          String messageSelector,
                                                          ServerSessionPool sessionPool,
                                                          int maxMessages)
                                                   throws JMSException
```

Create a durable connection consumer for this connection (optional operation). This is an expert facility not used by regular JMS clients.

Parameters:
`topic` - topic to access
`subscriptionName` - durable subscription name
`messageSelector` - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
`sessionPool` - the server session pool to associate with this durable connection consumer
`maxMessages` - the maximum number of messages that can be assigned to a server session at one time

Returns:
the durable connection consumer

Throws:
`JMSException` - if the `Connection` object fails to create a connection consumer due to some internal error or invalid arguments for `sessionPool` and `messageSelector`.
`InvalidDestinationException` - if an invalid destination is specified.
`InvalidSelectorException` - if the message selector is invalid.

Since:
1.1

See Also:
ConnectionConsumer

---

javax.jms

Interface ConnectionFactory

All Known Subinterfaces:

QueueConnectionFactory, TopicConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

---

public interface **ConnectionFactory**

A `ConnectionFactory` object encapsulates a set of connection configuration parameters that has been defined by an administrator. A client uses it to create a connection with a JMS provider.

A `ConnectionFactory` object is a JMS administered object and supports concurrent use.

JMS administered objects are objects containing configuration information that are created by an administrator and later used by JMS clients. They make it practical to administer the JMS API in the enterprise.

Although the interfaces for administered objects do not explicitly depend on the Java Naming and Directory Interface (JNDI) API, the JMS API establishes the convention that JMS clients find administered objects by looking them up in a JNDI namespace.

An administrator can place an administered object anywhere in a namespace. The JMS API does not define a naming policy.

It is expected that JMS providers will provide the tools an administrator needs to create and configure administered objects in a JNDI namespace. JMS provider implementations of administered objects should be both `javax.jndi.Referenceable` and `java.io.Serializable` so that they can be stored in all JNDI naming contexts. In addition, it is recommended that these implementations follow the JavaBeans™ design patterns.

This strategy provides several benefits:

- It hides provider-specific details from JMS clients.
- It abstracts administrative information into objects in the Java programming language ("Java objects") that are easily organized and administered from a common management console.

- Since there will be JNDI providers for all popular naming services, this means that JMS providers can deliver one implementation of administered objects that will run everywhere.

An administered object should not hold on to any remote resources. Its lookup should not use remote resources other than those used by the JNDI API itself.

Clients should think of administered objects as local Java objects. Looking them up should not have any hidden side effects or use surprising amounts of local resources.

Version:

1.1 - February 1, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`Connection`, `QueueConnectionFactory`, `TopicConnectionFactory`

Method Summary

`Connection`	**`createConnection`**`()` Creates a connection with the default user identity.
`Connection`	**`createConnection`**`(String userName, String password)` Creates a connection with the specified user identity.

Method Detail

createConnection

```
public Connection createConnection()
                           throws JMSException
```

Creates a connection with the default user identity. The connection is created in stopped mode. No messages will be delivered until the `Connection.start` method is explicitly called.

Returns:

a newly created connection

Throws:

`JMSException` - if the JMS provider fails to create the connection due to some internal error.

`JMSSecurityException` - if client authentication fails due to an invalid user name

or password.

Since:

1.1

---

createConnection

```
public Connection createConnection(String userName,
                                   String password)
                            throws JMSException
```

Creates a connection with the specified user identity. The connection is created in stopped mode. No messages will be delivered until the `Connection.start` method is explicitly called.

Parameters:

`userName` - the caller's user name
`password` - the caller's password

Returns:

a newly created connection

Throws:

`JMSException` - if the JMS provider fails to create the connection due to some internal error.
`JMSSecurityException` - if client authentication fails due to an invalid user name or password.

Since:

1.1

---

javax.jms

Interface Destination

All Known Subinterfaces:

Queue, TemporaryQueue, TemporaryTopic, Topic

---

public interface **Destination**

A `Destination` object encapsulates a provider-specific address. The JMS API does not define a standard address syntax. Although a standard address syntax was considered, it was decided that the differences in address semantics between existing message-oriented middleware (MOM) products were too wide to bridge with a single syntax.

Since `Destination` is an administered object, it may contain provider-specific configuration information in addition to its address.

The JMS API also supports a client's use of provider-specific address names.

`Destination` objects support concurrent use.

A `Destination` object is a JMS administered object.

JMS administered objects are objects containing configuration information that are created by an administrator and later used by JMS clients. They make it practical to administer the JMS API in the enterprise.

Although the interfaces for administered objects do not explicitly depend on the Java Naming and Directory Interface (JNDI) API, the JMS API establishes the convention that JMS clients find administered objects by looking them up in a JNDI namespace.

An administrator can place an administered object anywhere in a namespace. The JMS API does not define a naming policy.

It is expected that JMS providers will provide the tools an administrator needs to create and configure administered objects in a JNDI namespace. JMS provider implementations of administered objects should implement the `javax.naming.Referenceable` and `java.io.Serializable`

interfaces so that they can be stored in all JNDI naming contexts. In addition, it is recommended that these implementations follow the JavaBeans™ design patterns.

This strategy provides several benefits:

- It hides provider-specific details from JMS clients.
- It abstracts JMS administrative information into objects in the Java programming language ("Java objects") that are easily organized and administered from a common management console.
- Since there will be JNDI providers for all popular naming services, JMS providers can deliver one implementation of administered objects that will run everywhere.

An administered object should not hold on to any remote resources. Its lookup should not use remote resources other than those used by the JNDI API itself.

Clients should think of administered objects as local Java objects. Looking them up should not have any hidden side effects or use surprising amounts of local resources.

Version:

1.0 - 3 August 1998

Author:

Mark Hapner, Rich Burridge

See Also:

`Queue`, `Topic`

---

javax.jms

Interface MessageProducer

All Known Subinterfaces:

QueueSender, TopicPublisher

---

public interface **MessageProducer**

A client uses a `MessageProducer` object to send messages to a destination. A `MessageProducer` object is created by passing a `Destination` object to a message-producer creation method supplied by a session.

`MessageProducer` is the parent interface for all message producers.

A client also has the option of creating a message producer without supplying a destination. In this case, a destination must be provided with every send operation. A typical use for this kind of message producer is to send replies to requests using the request's `JMSReplyTo` destination.

A client can specify a default delivery mode, priority, and time to live for messages sent by a message producer. It can also specify the delivery mode, priority, and time to live for an individual message.

A client can specify a time-to-live value in milliseconds for each message it sends. This value defines a message expiration time that is the sum of the message's time-to-live and the GMT when it is sent (for transacted sends, this is the time the client sends the message, not the time the transaction is committed).

A JMS provider should do its best to expire messages accurately; however, the JMS API does not define the accuracy provided.

Version:

1.1 - February 2, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`TopicPublisher`, `QueueSender`, `Session.createProducer(javax.jms.`

Destination)

Method Summary

void	**close**() Closes the message producer.
int	**getDeliveryMode**() Gets the producer's default delivery mode.
Destination	**getDestination**() Gets the destination associated with this MessageProducer.
boolean	**getDisableMessageID**() Gets an indication of whether message IDs are disabled.
boolean	**getDisableMessageTimestamp**() Gets an indication of whether message timestamps are disabled.
int	**getPriority**() Gets the producer's default priority.
long	**getTimeToLive**() Gets the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system.
void	**send**(Destination destination, Message message) Sends a message to a destination for an unidentified message producer.
void	**send**(Destination destination, Message message, int deliveryMode, int priority, long timeToLive) Sends a message to a destination for an unidentified message producer, specifying delivery mode, priority and time to live.
void	**send**(Message message) Sends a message using the MessageProducer's default delivery mode, priority, and time to live.
void	**send**(Message message, int deliveryMode, int priority, long timeToLive) Sends a message to the destination, specifying delivery mode, priority, and time to live.
void	**setDeliveryMode**(int deliveryMode) Sets the producer's default delivery mode.
void	**setDisableMessageID**(boolean value) Sets whether message IDs are disabled.

void	**setDisableMessageTimestamp**(boolean value) Sets whether message timestamps are disabled.
void	**setPriority**(int defaultPriority) Sets the producer's default priority.
void	**setTimeToLive**(long timeToLive) Sets the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system.

Method Detail

setDisableMessageID

```
public void setDisableMessageID(boolean value)
                         throws JMSException
```

Sets whether message IDs are disabled.

Since message IDs take some effort to create and increase a message's size, some JMS providers may be able to optimize message overhead if they are given a hint that the message ID is not used by an application. By calling the `setDisableMessageID` method on this message producer, a JMS client enables this potential optimization for all messages sent by this message producer. If the JMS provider accepts this hint, these messages must have the message ID set to null; if the provider ignores the hint, the message ID must be set to its normal unique value.

Message IDs are enabled by default.

Parameters:

`value` - indicates if message IDs are disabled

Throws:

`JMSException` - if the JMS provider fails to set message ID to disabled due to some internal error.

---

getDisableMessageID

```
public boolean getDisableMessageID()
                            throws JMSException
```

Gets an indication of whether message IDs are disabled.

Returns:

an indication of whether message IDs are disabled

Throws:

`JMSException` - if the JMS provider fails to determine if message IDs are disabled due to some internal error.

---

setDisableMessageTimestamp

```
public void setDisableMessageTimestamp(boolean value)
                                throws JMSException
```

Sets whether message timestamps are disabled.

Since timestamps take some effort to create and increase a message's size, some JMS providers may be able to optimize message overhead if they are given a hint that the timestamp is not used by an application. By calling the `setDisableMessageTimestamp` method on this message producer, a JMS client enables this potential optimization for all messages sent by this message producer. If the JMS provider accepts this hint, these messages must have the timestamp set to zero; if the provider ignores the hint, the timestamp must be set to its normal value.

Message timestamps are enabled by default.

Parameters:

`value` - indicates if message timestamps are disabled

Throws:

`JMSException` - if the JMS provider fails to set timestamps to disabled due to some internal error.

---

getDisableMessageTimestamp

```
public boolean getDisableMessageTimestamp()
                                 throws JMSException
```

Gets an indication of whether message timestamps are disabled.

Returns:

an indication of whether message timestamps are disabled

Throws:

JMSException - if the JMS provider fails to determine if timestamps are disabled due to some internal error.

---

setDeliveryMode

```
public void setDeliveryMode(int deliveryMode)
                     throws JMSException
```

Sets the producer's default delivery mode.

Delivery mode is set to PERSISTENT by default.

Parameters:

deliveryMode - the message delivery mode for this message producer; legal values are DeliveryMode.NON_PERSISTENT and DeliveryMode.PERSISTENT

Throws:

JMSException - if the JMS provider fails to set the delivery mode due to some internal error.

See Also:

getDeliveryMode(), DeliveryMode.NON_PERSISTENT, DeliveryMode.PERSISTENT, Message.DEFAULT_DELIVERY_MODE

---

getDeliveryMode

```
public int getDeliveryMode()
                   throws JMSException
```

Gets the producer's default delivery mode.

Returns:

the message delivery mode for this message producer

Throws:

JMSException - if the JMS provider fails to get the delivery mode due to some internal error.

See Also:

setDeliveryMode(int)

---

setPriority

```
public void setPriority(int defaultPriority)
                 throws JMSException
```

Sets the producer's default priority.

The JMS API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. Clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority. Priority is set to 4 by default.

Parameters:

`defaultPriority` - the message priority for this message producer; must be a value between 0 and 9

Throws:

`JMSException` - if the JMS provider fails to set the priority due to some internal error.

See Also:

`getPriority()`, `Message.DEFAULT_PRIORITY`

---

getPriority

```
public int getPriority()
                throws JMSException
```

Gets the producer's default priority.

Returns:

the message priority for this message producer

Throws:

`JMSException` - if the JMS provider fails to get the priority due to some internal error.

See Also:

`setPriority(int)`

---

setTimeToLive

```
public void setTimeToLive(long timeToLive)
                  throws JMSException
```

Sets the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system.

Time to live is set to zero by default.

Parameters:

`timeToLive` - the message time to live in milliseconds; zero is unlimited

Throws:

`JMSException` - if the JMS provider fails to set the time to live due to some internal error.

See Also:

`getTimeToLive()`, `Message.DEFAULT_TIME_TO_LIVE`

---

getTimeToLive

```
public long getTimeToLive()
                    throws JMSException
```

Gets the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system.

Returns:

the message time to live in milliseconds; zero is unlimited

Throws:

`JMSException` - if the JMS provider fails to get the time to live due to some internal error.

See Also:

`setTimeToLive(long)`

---

getDestination

```
public Destination getDestination()
                             throws JMSException
```

Gets the destination associated with this `MessageProducer`.

Returns:

this producer's `Destination/`

Throws:

JMSException - if the JMS provider fails to get the destination for this MessageProducer due to some internal error.

Since:

1.1

---

close

```
public void close()
           throws JMSException
```

Closes the message producer.

Since a provider may allocate some resources on behalf of a MessageProducer outside the Java virtual machine, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

Throws:

JMSException - if the JMS provider fails to close the producer due to some internal error.

---

send

```
public void send(Message message)
           throws JMSException
```

Sends a message using the MessageProducer's default delivery mode, priority, and time to live.

Parameters:

message - the message to send

Throws:

JMSException - if the JMS provider fails to send the message due to some internal error.

MessageFormatException - if an invalid message is specified.

InvalidDestinationException - if a client uses this

method with a MessageProducer with an invalid destination.
UnsupportedOperationException - if a client uses this method with a MessageProducer that did not specify a destination at creation time.

Since:

1.1

See Also:

Session.createProducer(javax.jms.Destination), MessageProducer

---

send

```
public void send(Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive)
          throws JMSException
```

Sends a message to the destination, specifying delivery mode, priority, and time to live.

Parameters:

message - the message to send
deliveryMode - the delivery mode to use
priority - the priority for this message
timeToLive - the message's lifetime (in milliseconds)

Throws:

JMSException - if the JMS provider fails to send the message due to some internal error.
MessageFormatException - if an invalid message is specified.
InvalidDestinationException - if a client uses this method with a MessageProducer with an invalid destination.
UnsupportedOperationException - if a client uses this method with a MessageProducer that did not specify a destination at creation time.

Since:

1.1

See Also:

Session.createProducer(javax.jms.Destination)

---

send

```
public void send(Destination destination,
                 Message message)
          throws JMSException
```

Sends a message to a destination for an unidentified message producer. Uses the MessageProducer's default delivery mode, priority, and time to live.

Typically, a message producer is assigned a destination at creation time; however, the JMS API also supports unidentified message producers, which require that the destination be supplied every time a message is sent.

Parameters:

destination - the destination to send this message to
message - the message to send

Throws:

JMSException - if the JMS provider fails to send the message due to some internal error.
MessageFormatException - if an invalid message is specified.
InvalidDestinationException - if a client uses this method with an invalid destination.
UnsupportedOperationException - if a client uses this method with a MessageProducer that specified a destination at creation time.

Since:

1.1

See Also:

Session.createProducer(javax.jms.Destination), MessageProducer

---

send

```
public void send(Destination destination,
                 Message message,
                 int deliveryMode,
```

```
                int priority,
                long timeToLive)
        throws JMSException
```

Sends a message to a destination for an unidentified message producer, specifying delivery mode, priority and time to live.

Typically, a message producer is assigned a destination at creation time; however, the JMS API also supports unidentified message producers, which require that the destination be supplied every time a message is sent.

Parameters:

destination - the destination to send this message to
message - the message to send
deliveryMode - the delivery mode to use
priority - the priority for this message
timeToLive - the message's lifetime (in milliseconds)

Throws:

JMSException - if the JMS provider fails to send the message due to some internal error.
MessageFormatException - if an invalid message is specified.
InvalidDestinationException - if a client uses this method with an invalid destination.

Since:

1.1

See Also:

Session.createProducer(javax.jms.Destination)

---

javax.jms

Interface MessageConsumer

All Known Subinterfaces:

QueueReceiver, TopicSubscriber

---

public interface **MessageConsumer**

A client uses a `MessageConsumer` object to receive messages from a destination. A `MessageConsumer` object is created by passing a `Destination` object to a message-consumer creation method supplied by a session.

`MessageConsumer` is the parent interface for all message consumers.

A message consumer can be created with a message selector. A message selector allows the client to restrict the messages delivered to the message consumer to those that match the selector.

A client may either synchronously receive a message consumer's messages or have the consumer asynchronously deliver them as they arrive.

For synchronous receipt, a client can request the next message from a message consumer using one of its `receive` methods. There are several variations of `receive` that allow a client to poll or wait for the next message.

For asynchronous delivery, a client can register a `MessageListener` object with a message consumer. As messages arrive at the message consumer, it delivers them by calling the `MessageListener`'s `onMessage` method.

It is a client programming error for a `MessageListener` to throw an exception.

Version:

1.0 - 13 March 1998

Author:

Mark Hapner, Rich Burridge

See Also:

`QueueReceiver`, `TopicSubscriber`, `Session`

Method Summary

void	**close**() Closes the message consumer.
MessageListener	**getMessageListener**() Gets the message consumer's MessageListener.
String	**getMessageSelector**() Gets this message consumer's message selector expression.
Message	**receive**() Receives the next message produced for this message consumer.
Message	**receive**(long timeout) Receives the next message that arrives within the specified timeout interval.
Message	**receiveNoWait**() Receives the next message if one is immediately available.
void	**setMessageListener**(MessageListener listener) Sets the message consumer's MessageListener.

Method Detail

getMessageSelector

```
public String getMessageSelector()
                            throws JMSException
```

Gets this message consumer's message selector expression.

Returns:

this message consumer's message selector, or null if no message selector exists for the message consumer (that is, if the message selector was not set or was set to null or the empty string)

Throws:

JMSException - if the JMS provider fails to get the message selector due to some internal error.

getMessageListener

```
public MessageListener getMessageListener()
                                    throws JMSException
```

Gets the message consumer's `MessageListener`.

Returns:
the listener for the message consumer, or null if no listener is set

Throws:
`JMSException` - if the JMS provider fails to get the message listener due to some internal error.

See Also:
`setMessageListener(javax.jms.MessageListener)`

---

setMessageListener

```
public void setMessageListener(MessageListener listener)
                        throws JMSException
```

Sets the message consumer's `MessageListener`.

Setting the message listener to null is the equivalent of unsetting the message listener for the message consumer.

The effect of calling `MessageConsumer.setMessageListener` while messages are being consumed by an existing listener or the consumer is being used to consume messages synchronously is undefined.

Parameters:
`listener` - the listener to which the messages are to be delivered

Throws:
`JMSException` - if the JMS provider fails to set the message listener due to some internal error.

See Also:
`getMessageListener()`

---

receive

```
public Message receive()
                throws JMSException
```

Receives the next message produced for this message consumer.

This call blocks indefinitely until a message is produced or until this message consumer is closed.

If this `receive` is done within a transaction, the consumer retains the message until the transaction commits.

Returns:

the next message produced for this message consumer, or null if this message consumer is concurrently closed

Throws:

`JMSException` - if the JMS provider fails to receive the next message due to some internal error.

---

receive

```
public Message receive(long timeout)
                throws JMSException
```

Receives the next message that arrives within the specified timeout interval.

This call blocks until a message arrives, the timeout expires, or this message consumer is closed. A `timeout` of zero never expires, and the call blocks indefinitely.

Parameters:

`timeout` - the timeout value (in milliseconds)

Returns:

the next message produced for this message consumer, or null if the timeout expires or this message consumer is concurrently closed

Throws:

`JMSException` - if the JMS provider fails to receive the next message due to some internal error.

---

receiveNoWait

```
public Message receiveNoWait()
                    throws JMSException
```

Receives the next message if one is immediately available.

Returns:

the next message produced for this message consumer, or null if one is not available

Throws:

JMSException - if the JMS provider fails to receive the next message due to some internal error.

---

close

```
public void close()
           throws JMSException
```

Closes the message consumer.

Since a provider may allocate some resources on behalf of a `MessageConsumer` outside the Java virtual machine, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

This call blocks until a `receive` or message listener in progress has completed. A blocked message consumer `receive` call returns null when this message consumer is closed.

Throws:

JMSException - if the JMS provider fails to close the consumer due to some internal error.

---

javax.jms

Interface Queue

All Superinterfaces:

Destination

All Known Subinterfaces:

TemporaryQueue

---

public interface **Queue**
extends Destination

A `Queue` object encapsulates a provider-specific queue name. It is the way a client specifies the identity of a queue to JMS API methods. For those methods that use a `Destination` as a parameter, a `Queue` object used as an argument. For example, a queue can be used to create a `MessageConsumer` and a `MessageProducer` by calling:

- `Session.CreateConsumer(Destination destination)`
- `Session.CreateProducer(Destination destination)`

The actual length of time messages are held by a queue and the consequences of resource overflow are not defined by the JMS API.

Version:

1.1 February 2 - 2000

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`Session.createConsumer(Destination)`, `Session.createProducer(Destination)`, `Session.createQueue(String)`, `QueueSession.createQueue(String)`

Method Summary

String	**getQueueName**() Gets the name of this queue.
String	**toString**() Returns a string representation of this object.

Method Detail

getQueueName

```
public String getQueueName()
                    throws JMSException
```

Gets the name of this queue.

Clients that depend upon the name are not portable.

Returns:
the queue name

Throws:
JMSException - if the JMS provider implementation of Queue fails to return the queue name due to some internal error.

---

toString

```
public String toString()
```

Returns a string representation of this object.

Returns:
the provider-specific identity values for this queue

---

javax.jms

Interface Session

All Superinterfaces:

Runnable

All Known Subinterfaces:

QueueSession, TopicSession, XAQueueSession, XASession, XATopicSession

---

public interface **Session**
extends Runnable

A `Session` object is a single-threaded context for producing and consuming messages. Although it may allocate provider resources outside the Java virtual machine (JVM), it is considered a lightweight JMS object.

A session serves several purposes:

- It is a factory for its message producers and consumers.
- It supplies provider-optimized message factories.
- It is a factory for `TemporaryTopics` and `TemporaryQueues`.
- It provides a way to create `Queue` or `Topic` objects for those clients that need to dynamically manipulate provider-specific destination names.
- It supports a single series of transactions that combine work spanning its producers and consumers into atomic units.
- It defines a serial order for the messages it consumes and the messages it produces.
- It retains messages it consumes until they have been acknowledged.
- It serializes execution of message listeners registered with its message consumers.
- It is a factory for `QueueBrowsers`.

A session can create and service multiple message producers and consumers.

One typical use is to have a thread block on a synchronous `MessageConsumer` until a message arrives. The thread may then use one or more of the `Session`'s `MessageProducers`.

If a client desires to have one thread produce messages while others consume them, the client should use a separate session for its producing thread.

Once a connection has been started, any session with one or more registered message listeners is dedicated to the thread of control that delivers messages to it. It is erroneous for client code to use this session or any of its constituent objects from another thread of control. The only exception to this rule is the use of the session or connection `close` method.

It should be easy for most clients to partition their work naturally into sessions. This model allows clients to start simply and incrementally add message processing complexity as their need for concurrency grows.

The `close` method is the only session method that can be called while some other session method is being executed in another thread.

A session may be specified as transacted. Each transacted session supports a single series of transactions. Each transaction groups a set of message sends and a set of message receives into an atomic unit of work. In effect, transactions organize a session's input message stream and output message stream into series of atomic units. When a transaction commits, its atomic unit of input is acknowledged and its associated atomic unit of output is sent. If a transaction rollback is done, the transaction's sent messages are destroyed and the session's input is automatically recovered.

The content of a transaction's input and output units is simply those messages that have been produced and consumed within the session's current transaction.

A transaction is completed using either its session's `commit` method or its session's `rollback` method. The completion of a session's current transaction automatically begins the next. The result is that a transacted session always has a current transaction within which its work is done.

The Java Transaction Service (JTS) or some other transaction monitor may be used to combine a session's transaction with transactions on other resources (databases, other JMS sessions, etc.). Since Java distributed transactions are controlled via the Java Transaction API (JTA), use of the session's `commit` and `rollback` methods in this context is prohibited.

The JMS API does not require support for JTA; however, it does define how a provider supplies this support.

Although it is also possible for a JMS client to handle distributed transactions directly, it is unlikely that many JMS clients will do this. Support for JTA in the JMS API is targeted at systems vendors who will be integrating the JMS API into their application server products.

Version:

1.1 February 2, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`QueueSession`, `TopicSession`, `XASession`

Field Summary

static int	**AUTO_ACKNOWLEDGE** With this acknowledgment mode, the session automatically acknowledges a client's receipt of a message either when the session has successfully returned from a call to `receive` or when the message listener the session has called to process the message successfully returns.
static int	**CLIENT_ACKNOWLEDGE** With this acknowledgment mode, the client acknowledges a consumed message by calling the message's `acknowledge` method.
static int	**DUPS_OK_ACKNOWLEDGE** This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages.
static int	**SESSION_TRANSACTED** This value is returned from the method `getAcknowledgeMode` if the session is transacted.

Method Summary

void	**close**() Closes the session.
void	**commit**() Commits all messages done in this transaction and releases any locks currently held.
QueueBrowser	**createBrowser**(Queue queue) Creates a `QueueBrowser` object to peek at the messages on the specified queue.
QueueBrowser	**createBrowser**(Queue queue, String messageSelector) Creates a `QueueBrowser` object to peek at the messages on the specified queue using a message selector.
BytesMessage	**createBytesMessage**() Creates a `BytesMessage` object.
MessageConsumer	**createConsumer**(Destination destination) Creates a `MessageConsumer` for the specified destination.
MessageConsumer	**createConsumer**(Destination destination, String messageSelector) Creates a `MessageConsumer` for the specified destination, using a message selector.

MessageConsumer	**createConsumer**(Destination destination, String messageSelector, boolean NoLocal) Creates MessageConsumer for the specified destination, using a message selector.
TopicSubscriber	**createDurableSubscriber**(Topic topic, String name) Creates a durable subscriber to the specified topic.
TopicSubscriber	**createDurableSubscriber**(Topic topic, String name, String messageSelector, boolean noLocal) Creates a durable subscriber to the specified topic, using a message selector and specifying whether messages published by its own connection should be delivered to it.
MapMessage	**createMapMessage**() Creates a MapMessage object.
Message	**createMessage**() Creates a Message object.
ObjectMessage	**createObjectMessage**() Creates an ObjectMessage object.
ObjectMessage	**createObjectMessage**(Serializable object) Creates an initialized ObjectMessage object.
MessageProducer	**createProducer**(Destination destination) Creates a MessageProducer to send messages to the specified destination.
Queue	**createQueue**(String queueName) Creates a queue identity given a Queue name.
StreamMessage	**createStreamMessage**() Creates a StreamMessage object.
TemporaryQueue	**createTemporaryQueue**() Creates a TemporaryQueue object.
TemporaryTopic	**createTemporaryTopic**() Creates a TemporaryTopic object.
TextMessage	**createTextMessage**() Creates a TextMessage object.
TextMessage	**createTextMessage**(String text) Creates an initialized TextMessage object.
Topic	**createTopic**(String topicName) Creates a topic identity given a Topic name.

int	**getAcknowledgeMode**() Returns the acknowledgement mode of the session.
MessageListener	**getMessageListener**() Returns the session's distinguished message listener (optional).
boolean	**getTransacted**() Indicates whether the session is in transacted mode.
void	**recover**() Stops message delivery in this session, and restarts message delivery with the oldest unacknowledged message.
void	**rollback**() Rolls back any messages done in this transaction and releases any locks currently held.
void	**run**() Optional operation, intended to be used only by Application Servers, not by ordinary JMS clients.
void	**setMessageListener**(MessageListener listener) Sets the session's distinguished message listener (optional).
void	**unsubscribe**(String name) Unsubscribes a durable subscription that has been created by a client.

Field Detail

AUTO_ACKNOWLEDGE

```
public static final int AUTO_ACKNOWLEDGE
```

With this acknowledgment mode, the session automatically acknowledges a client's receipt of a message either when the session has successfully returned from a call to `receive` or when the message listener the session has called to process the message successfully returns.

See Also:

Constant Field Values

CLIENT_ACKNOWLEDGE

```
public static final int CLIENT_ACKNOWLEDGE
```

With this acknowledgment mode, the client acknowledges a consumed message by calling the message's `acknowledge` method. Acknowledging a consumed message acknowledges all messages that the session has consumed.

When client acknowledgment mode is used, a client may build up a large number of unacknowledged messages while attempting to process them. A JMS provider should provide administrators with a way to limit client overrun so that clients are not driven to resource exhaustion and ensuing failure when some resource they are using is temporarily blocked.

See Also:

`Message.acknowledge()`, Constant Field Values

DUPS_OK_ACKNOWLEDGE

```
public static final int DUPS_OK_ACKNOWLEDGE
```

This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages. This is likely to result in the delivery of some duplicate messages if the JMS provider fails, so it should only be used by consumers that can tolerate duplicate messages. Use of this mode can reduce session overhead by minimizing the work the session does to prevent duplicates.

See Also:

Constant Field Values

SESSION_TRANSACTED

```
public static final int SESSION_TRANSACTED
```

This value is returned from the method `getAcknowledgeMode` if the session is transacted. If a `Session` is transacted, the acknowledgement mode is ignored.

See Also:

Constant Field Values

Method Detail

createBytesMessage

```
public BytesMessage createBytesMessage()
                                  throws JMSException
```

Creates a `BytesMessage` object. A `BytesMessage` object is used to send a message containing a stream of uninterpreted bytes.

Throws:

JMSException - if the JMS provider fails to create this message due to some internal error.

---

createMapMessage

```
public MapMessage createMapMessage()
                             throws JMSException
```

Creates a `MapMessage` object. A `MapMessage` object is used to send a self-defining set of name-value pairs, where names are `String` objects and values are primitive values in the Java programming language.

Throws:

JMSException - if the JMS provider fails to create this message due to some internal error.

---

createMessage

```
public Message createMessage()
                       throws JMSException
```

Creates a `Message` object. The `Message` interface is the root interface of all JMS messages. A `Message` object holds all the standard message header information. It can be sent when a message containing only header information is sufficient.

Throws:

JMSException - if the JMS provider fails to create this message due to some internal error.

---

createObjectMessage

```
public ObjectMessage createObjectMessage()
                                    throws JMSException
```

Creates an `ObjectMessage` object. An `ObjectMessage` object is used to send a message that contains a serializable Java object.

Throws:

`JMSException` - if the JMS provider fails to create this message due to some internal error.

---

createObjectMessage

```
public ObjectMessage createObjectMessage(Serializable object)
                                    throws JMSException
```

Creates an initialized `ObjectMessage` object. An `ObjectMessage` object is used to send a message that contains a serializable Java object.

Parameters:

`object` - the object to use to initialize this message

Throws:

`JMSException` - if the JMS provider fails to create this message due to some internal error.

---

createStreamMessage

```
public StreamMessage createStreamMessage()
                                    throws JMSException
```

Creates a `StreamMessage` object. A `StreamMessage` object is used to send a self-defining stream of primitive values in the Java programming language.

Throws:

`JMSException` - if the JMS provider fails to create this message due to some internal error.

---

createTextMessage

```
public TextMessage createTextMessage()
                            throws JMSException
```

Creates a `TextMessage` object. A `TextMessage` object is used to send a message containing a `String` object.

Throws:

`JMSException` - if the JMS provider fails to create this message due to some internal error.

---

createTextMessage

```
public TextMessage createTextMessage(String text)
                            throws JMSException
```

Creates an initialized `TextMessage` object. A `TextMessage` object is used to send a message containing a `String`.

Parameters:

`text` - the string used to initialize this message

Throws:

`JMSException` - if the JMS provider fails to create this message due to some internal error.

---

getTransacted

```
public boolean getTransacted()
                     throws JMSException
```

Indicates whether the session is in transacted mode.

Returns:

true if the session is in transacted mode

Throws:

`JMSException` - if the JMS provider fails to return the transaction mode due to some internal error.

---

getAcknowledgeMode

```
public int getAcknowledgeMode()
                       throws JMSException
```

Returns the acknowledgement mode of the session. The acknowledgement mode is set at the time that the session is created. If the session is transacted, the acknowledgement mode is ignored.

Returns:

If the session is not transacted, returns the current acknowledgement mode for the session. If the session is transacted, returns SESSION_TRANSACTED.

Throws:

JMSException - if the JMS provider fails to return the acknowledgment mode due to some internal error.

Since:

1.1

See Also:

Connection.createSession(boolean, int)

---

commit

```
public void commit()
            throws JMSException
```

Commits all messages done in this transaction and releases any locks currently held.

Throws:

JMSException - if the JMS provider fails to commit the transaction due to some internal error.
TransactionRolledBackException - if the transaction is rolled back due to some internal error during commit.
IllegalStateException - if the method is not called by a transacted session.

---

rollback

```
public void rollback()
              throws JMSException
```

Rolls back any messages done in this transaction and releases any locks currently held.

Throws:

JMSException - if the JMS provider fails to roll back the transaction due to some internal error.
IllegalStateException - if the method is not called by a transacted session.

---

close

```
public void close()
            throws JMSException
```

Closes the session.

Since a provider may allocate some resources on behalf of a session outside the JVM, clients should close the resources when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

There is no need to close the producers and consumers of a closed session.

This call will block until a `receive` call or message listener in progress has completed. A blocked message consumer `receive` call returns `null` when this session is closed.

Closing a transacted session must roll back the transaction in progress.

This method is the only `Session` method that can be called concurrently.

Invoking any other `Session` method on a closed session must throw a `JMSException.IllegalStateException`. Closing a closed session must *not* throw an exception.

Throws:

JMSException - if the JMS provider fails to close the session due to some internal error.

---

recover

```
public void recover()
             throws JMSException
```

Stops message delivery in this session, and restarts message delivery with the oldest unacknowledged message.

All consumers deliver messages in a serial order. Acknowledging a received message

automatically acknowledges all messages that have been delivered to the client.

Restarting a session causes it to take the following actions:

- Stop message delivery
- Mark all messages that might have been delivered but not acknowledged as "redelivered"
- Restart the delivery sequence including all unacknowledged messages that had been previously delivered. Redelivered messages do not have to be delivered in exactly their original delivery order.

Throws:

`JMSException` - if the JMS provider fails to stop and restart message delivery due to some internal error.
`IllegalStateException` - if the method is called by a transacted session.

---

getMessageListener

```
public MessageListener getMessageListener()
                                    throws JMSException
```

Returns the session's distinguished message listener (optional).

Returns:

the message listener associated with this session

Throws:

`JMSException` - if the JMS provider fails to get the message listener due to an internal error.

See Also:

`setMessageListener(javax.jms.MessageListener)`, `ServerSessionPool`, `ServerSession`

---

setMessageListener

```
public void setMessageListener(MessageListener listener)
                        throws JMSException
```

Sets the session's distinguished message listener (optional).

When the distinguished message listener is set, no other form of message receipt in the session can be used; however, all forms of sending messages are still supported.

This is an expert facility not used by regular JMS clients.

Parameters:

`listener` - the message listener to associate with this session

Throws:

`JMSException` - if the JMS provider fails to set the message listener due to an internal error.

See Also:

`getMessageListener()`, `ServerSessionPool`, `ServerSession`

---

run

```
public void run()
```

Optional operation, intended to be used only by Application Servers, not by ordinary JMS clients.

Specified by:

`run` in interface `Runnable`

See Also:

`ServerSession`

---

createProducer

```
public MessageProducer createProducer(Destination destination)
                                throws JMSException
```

Creates a `MessageProducer` to send messages to the specified destination.

A client uses a `MessageProducer` object to send messages to a destination. Since `Queue` and `Topic` both inherit from `Destination`, they can be used in the destination parameter to create a `MessageProducer` object.

Parameters:

`destination` - the `Destination` to send to, or null if this is a producer which does not have a specified destination.

Throws:

`JMSException` - if the session fails to create a MessageProducer due to some internal error.

`InvalidDestinationException` - if an invalid destination is specified.

Since:

1.1

---

createConsumer

```
public MessageConsumer createConsumer(Destination destination)
                               throws JMSException
```

Creates a `MessageConsumer` for the specified destination. Since `Queue` and `Topic` both inherit from `Destination`, they can be used in the destination parameter to create a `MessageConsumer`.

Parameters:

`destination` - the `Destination` to access.

Throws:

`JMSException` - if the session fails to create a consumer due to some internal error.

`InvalidDestinationException` - if an invalid destination is specified.

Since:

1.1

---

createConsumer

```
public MessageConsumer createConsumer(Destination destination,
                                      String messageSelector)
                               throws JMSException
```

Creates a `MessageConsumer` for the specified destination, using a message selector. Since `Queue` and `Topic` both inherit from `Destination`, they can be used in the destination parameter to create a `MessageConsumer`.

A client uses a `MessageConsumer` object to receive messages that have been sent to a destination.

Parameters:

`destination` - the `Destination` to access

`messageSelector` - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.

Throws:

`JMSException` - if the session fails to create a MessageConsumer due to some internal

error.
InvalidDestinationException - if an invalid destination is specified.
InvalidSelectorException - if the message selector is invalid.

Since:

1.1

---

createConsumer

```
public MessageConsumer createConsumer(Destination destination,
                                       String messageSelector,
                                       boolean NoLocal)
                                throws JMSException
```

Creates `MessageConsumer` for the specified destination, using a message selector. This method can specify whether messages published by its own connection should be delivered to it, if the destination is a topic.

Since `Queue` and `Topic` both inherit from `Destination`, they can be used in the destination parameter to create a `MessageConsumer`.

A client uses a `MessageConsumer` object to receive messages that have been published to a destination.

In some cases, a connection may both publish and subscribe to a topic. The consumer `NoLocal` attribute allows a consumer to inhibit the delivery of messages published by its own connection. The default value for this attribute is False. The `noLocal` value must be supported by destinations that are topics.

Parameters:

`destination` - the `Destination` to access
`messageSelector` - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
`NoLocal` - - if true, and the destination is a topic, inhibits the delivery of messages published by its own connection. The behavior for `NoLocal` is not specified if the destination is a queue.

Throws:

JMSException - if the session fails to create a MessageConsumer due to some internal error.
InvalidDestinationException - if an invalid destination is specified.
InvalidSelectorException - if the message selector is invalid.

Since:

1.1

---

createQueue

```
public Queue createQueue(String queueName)
                  throws JMSException
```

Creates a queue identity given a `Queue` name.

This facility is provided for the rare cases where clients need to dynamically manipulate queue identity. It allows the creation of a queue identity with a provider-specific name. Clients that depend on this ability are not portable.

Note that this method is not for creating the physical queue. The physical creation of queues is an administrative task and is not to be initiated by the JMS API. The one exception is the creation of temporary queues, which is accomplished with the `createTemporaryQueue` method.

Parameters:

`queueName` - the name of this `Queue`

Returns:

a `Queue` with the given name

Throws:

`JMSException` - if the session fails to create a queue due to some internal error.

Since:

1.1

---

createTopic

```
public Topic createTopic(String topicName)
                  throws JMSException
```

Creates a topic identity given a `Topic` name.

This facility is provided for the rare cases where clients need to dynamically manipulate topic identity. This allows the creation of a topic identity with a provider-specific name. Clients that depend on this ability are not portable.

Note that this method is not for creating the physical topic. The physical creation of topics is an administrative task and is not to be initiated by the JMS API. The one exception is the creation of temporary topics, which is accomplished with the `createTemporaryTopic` method.

Parameters:

`topicName` - the name of this `Topic`

Returns:

a `Topic` with the given name

Throws:

`JMSException` - if the session fails to create a topic due to some internal error.

Since:

1.1

---

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber(Topic topic,
                                               String name)
                                        throws JMSException
```

Creates a durable subscriber to the specified topic.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it uses a durable `TopicSubscriber`. The JMS provider retains a record of this durable subscription and insures that all messages from the topic's publishers are retained until they are acknowledged by this durable subscriber or they have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies (within client identifier) each durable subscription it creates. Only one session at a time can have a `TopicSubscriber` for a particular durable subscription.

A client can change an existing durable subscription by creating a durable `TopicSubscriber` with the same name and a new topic and/or message selector. Changing a durable subscriber is equivalent to unsubscribing (deleting) the old one and creating a new one.

In some cases, a connection may both publish and subscribe to a topic. The subscriber `NoLocal` attribute allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is false.

Parameters:

`topic` - the non-temporary `Topic` to subscribe to

`name` - the name used to identify this subscription

Throws:

`JMSException` - if the session fails to create a subscriber due to some internal error.

`InvalidDestinationException` - if an invalid topic is specified.

Since:

1.1

---

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber(Topic topic,
                                                String name,
                                                String messageSelector,
                                                boolean noLocal)
                                         throws JMSException
```

Creates a durable subscriber to the specified topic, using a message selector and specifying whether messages published by its own connection should be delivered to it.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it uses a durable `TopicSubscriber`. The JMS provider retains a record of this durable subscription and insures that all messages from the topic's publishers are retained until they are acknowledged by this durable subscriber or they have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name which uniquely identifies (within client identifier) each durable subscription it creates. Only one session at a time can have a `TopicSubscriber` for a particular durable subscription. An inactive durable subscriber is one that exists but does not currently have a message consumer associated with it.

A client can change an existing durable subscription by creating a durable `TopicSubscriber` with the same name and a new topic and/or message selector. Changing a durable subscriber is equivalent to unsubscribing (deleting) the old one and creating a new one.

Parameters:

`topic` - the non-temporary `Topic` to subscribe to
`name` - the name used to identify this subscription
`messageSelector` - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
`noLocal` - if set, inhibits the delivery of messages published by its own connection

Throws:

`JMSException` - if the session fails to create a subscriber due to some internal error.
`InvalidDestinationException` - if an invalid topic is specified.
`InvalidSelectorException` - if the message selector is invalid.

Since:

1.1

createBrowser

```
public QueueBrowser createBrowser(Queue queue)
                           throws JMSException
```

Creates a `QueueBrowser` object to peek at the messages on the specified queue.

Parameters:

`queue` - the `queue` to access

Throws:

`JMSException` - if the session fails to create a browser due to some internal error.

`InvalidDestinationException` - if an invalid destination is specified

Since:

1.1

---

createBrowser

```
public QueueBrowser createBrowser(Queue queue,
                                  String messageSelector)
                           throws JMSException
```

Creates a `QueueBrowser` object to peek at the messages on the specified queue using a message selector.

Parameters:

`queue` - the `queue` to access

`messageSelector` - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.

Throws:

`JMSException` - if the session fails to create a browser due to some internal error.

`InvalidDestinationException` - if an invalid destination is specified

`InvalidSelectorException` - if the message selector is invalid.

Since:

1.1

---

createTemporaryQueue

```
public TemporaryQueue createTemporaryQueue()
                                    throws JMSException
```

Creates a `TemporaryQueue` object. Its lifetime will be that of the `Connection` unless it is deleted earlier.

Returns:
a temporary queue identity
Throws:
`JMSException` - if the session fails to create a temporary queue due to some internal error.
Since:
1.1

---

createTemporaryTopic

```
public TemporaryTopic createTemporaryTopic()
                                    throws JMSException
```

Creates a `TemporaryTopic` object. Its lifetime will be that of the `Connection` unless it is deleted earlier.

Returns:
a temporary topic identity
Throws:
`JMSException` - if the session fails to create a temporary topic due to some internal error.
Since:
1.1

---

unsubscribe

```
public void unsubscribe(String name)
                 throws JMSException
```

Unsubscribes a durable subscription that has been created by a client.

This method deletes the state being maintained on behalf of the subscriber by its provider.

It is erroneous for a client to delete a durable subscription while there is an active

`MessageConsumer` or `TopicSubscriber` for the subscription, or while a consumed message is part of a pending transaction or has not been acknowledged in the session.

Parameters:

`name` - the name used to identify this subscription

Throws:

`JMSException` - if the session fails to unsubscribe to the durable subscription due to some internal error.

`InvalidDestinationException` - if an invalid subscription name is specified.

Since:

1.1

Overview Package Class Tree Deprecated Index Help *Java™ 2 Platform Ent. Ed. v1.4*

PREV CLASS NEXT CLASS **FRAMES NO FRAMES All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Submit a bug or feature

javax.jms

Interface TextMessage

All Superinterfaces:

Message

public interface **TextMessage**
extends Message

A `TextMessage` object is used to send a message containing a `java.lang.String`. It inherits from the `Message` interface and adds a text message body.

This message type can be used to transport text-based messages, including those with XML content.

When a client receives a `TextMessage`, it is in read-only mode. If a client attempts to write to the message at this point, a `MessageNotWriteableException` is thrown. If `clearBody` is called, the message can now be both read from and written to.

Version:

1.1 - February 2, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`Session.createTextMessage()`, `Session.createTextMessage(String)`, `BytesMessage`, `MapMessage`, `Message`, `ObjectMessage`, `StreamMessage`, `String`

Field Summary

Fields inherited from interface javax.jms.Message
DEFAULT_DELIVERY_MODE, DEFAULT_PRIORITY, DEFAULT_TIME_TO_LIVE

Method Summary

String	**getText**() Gets the string containing this message's data.
void	**setText**(String string) Sets the string containing this message's data.

Methods inherited from interface javax.jms.Message

acknowledge, clearBody, clearProperties, getBooleanProperty, getByteProperty, getDoubleProperty, getFloatProperty, getIntProperty, getJMSCorrelationID, getJMSCorrelationIDAsBytes, getJMSDeliveryMode, getJMSDestination, getJMSExpiration, getJMSMessageID, getJMSPriority, getJMSRedelivered, getJMSReplyTo, getJMSTimestamp, getJMSType, getLongProperty, getObjectProperty, getPropertyNames, getShortProperty, getStringProperty, propertyExists, setBooleanProperty, setByteProperty, setDoubleProperty, setFloatProperty, setIntProperty, setJMSCorrelationID, setJMSCorrelationIDAsBytes, setJMSDeliveryMode, setJMSDestination, setJMSExpiration, setJMSMessageID, setJMSPriority, setJMSRedelivered, setJMSReplyTo, setJMSTimestamp, setJMSType, setLongProperty, setObjectProperty, setShortProperty, setStringProperty

Method Detail

setText

```
public void setText(String string)
             throws JMSException
```

Sets the string containing this message's data.

Parameters:

`string` - the `String` containing the message's data

Throws:

`JMSException` - if the JMS provider fails to set the text due to some internal error.

`MessageNotWriteableException` - if the message is in read-only mode.

getText

```
public String getText()
                throws JMSException
```

Gets the string containing this message's data. The default value is null.

Returns:

the `String` containing the message's data

Throws:

`JMSException` - if the JMS provider fails to get the text due to some internal error.

Overview Package Class Tree Deprecated Index Help *Java™ 2 Platform Ent. Ed. v1.4*

PREV CLASS NEXT CLASS **FRAMES NO FRAMES All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Submit a bug or feature

javax.jms

Interface Topic

All Superinterfaces:

Destination

All Known Subinterfaces:

TemporaryTopic

---

public interface **Topic**
extends Destination

A `Topic` object encapsulates a provider-specific topic name. It is the way a client specifies the identity of a topic to JMS API methods. For those methods that use a `Destination` as a parameter, a `Topic` object may used as an argument . For example, a Topic can be used to create a `MessageConsumer` and a `MessageProducer` by calling:

- `Session.CreateConsumer(Destination destination)`
- `Session.CreateProducer(Destination destination)`

Many publish/subscribe (pub/sub) providers group topics into hierarchies and provide various options for subscribing to parts of the hierarchy. The JMS API places no restriction on what a `Topic` object represents. It may be a leaf in a topic hierarchy, or it may be a larger part of the hierarchy.

The organization of topics and the granularity of subscriptions to them is an important part of a pub/sub application's architecture. The JMS API does not specify a policy for how this should be done. If an application takes advantage of a provider-specific topic-grouping mechanism, it should document this. If the application is installed using a different provider, it is the job of the administrator to construct an equivalent topic architecture and create equivalent `Topic` objects.

Version:

1.1 - February 2, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`Session.createConsumer(Destination)`, `Session.createProducer`

(Destination), TopicSession.createTopic(String)

Method Summary	
String	**getTopicName**() Gets the name of this topic.
String	**toString**() Returns a string representation of this object.

Method Detail

getTopicName

```
public String getTopicName()
                    throws JMSException
```

Gets the name of this topic.

Clients that depend upon the name are not portable.

Returns:
the topic name

Throws:
JMSException - if the JMS provider implementation of `Topic` fails to return the topic name due to some internal error.

---

toString

```
public String toString()
```

Returns a string representation of this object.

Returns:
the provider-specific identity values for this topic

Submit a bug or feature

javax.jms

Interface TopicConnection

All Superinterfaces:

Connection

All Known Subinterfaces:

XATopicConnection

---

public interface **TopicConnection**
extends Connection

A `TopicConnection` object is an active connection to a publish/subscribe JMS provider. A client uses a `TopicConnection` object to create one or more `TopicSession` objects for producing and consuming messages.

A `TopicConnection` can be used to create a `TopicSession`, from which specialized topic-related objects can be created. A more general, and recommended approach is to use the `Connection` object.

The `TopicConnection` object should be used to support existing code.

Version:

1.1 - February 2, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`Connection`, `ConnectionFactory`, `TopicConnectionFactory`

Method Summary

ConnectionConsumer	**createConnectionConsumer**(Topic topic, String messageSelector, ServerSessionPool sessionPool, int maxMessages) Creates a connection consumer for this connection (optional operation).
ConnectionConsumer	**createDurableConnectionConsumer**(Topic topic, String subscriptionName, String messageSelector, ServerSessionPool sessionPool, int maxMessages) Create a durable connection consumer for this connection (optional operation).
TopicSession	**createTopicSession**(boolean transacted, int acknowledgeMode) Creates a `TopicSession` object.

Methods inherited from interface javax.jms.Connection

close, createConnectionConsumer, createSession, getClientID, getExceptionListener, getMetaData, setClientID, setExceptionListener, start, stop

Method Detail

createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                throws JMSException
```

Creates a `TopicSession` object.

Parameters:

`transacted` - indicates whether the session is transacted

`acknowledgeMode` - indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are `Session.AUTO_ACKNOWLEDGE`, `Session.CLIENT_ACKNOWLEDGE`, and `Session.DUPS_OK_ACKNOWLEDGE`.

Returns:

a newly created topic session

Throws:

`JMSException` - if the `TopicConnection` object fails to create a session due to some internal error or lack of support for the specific transaction and acknowledgement mode.

See Also:

`Session.AUTO_ACKNOWLEDGE`, `Session.CLIENT_ACKNOWLEDGE`, `Session.DUPS_OK_ACKNOWLEDGE`

---

createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer(Topic topic,
                                                   String messageSelector,
                                                   ServerSessionPool sessionPool,
                                                   int maxMessages)
                                            throws JMSException
```

Creates a connection consumer for this connection (optional operation). This is an expert facility not used by regular JMS clients.

Parameters:

`topic` - the topic to access

`messageSelector` - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.

`sessionPool` - the server session pool to associate with this connection consumer

`maxMessages` - the maximum number of messages that can be assigned to a server session at one time

Returns:

the connection consumer

Throws:

`JMSException` - if the `TopicConnection` object fails to create a connection consumer due to some internal error or invalid arguments for `sessionPool` and `messageSelector`.

`InvalidDestinationException` - if an invalid topic is specified.

`InvalidSelectorException` - if the message selector is invalid.

See Also:

`ConnectionConsumer`

---

createDurableConnectionConsumer

```
public ConnectionConsumer createDurableConnectionConsumer(Topic topic,
                                                          String subscriptionName,
                                                          String messageSelector,
                                                          ServerSessionPool sessionPool,
                                                          int maxMessages)
                                                   throws JMSException
```

Create a durable connection consumer for this connection (optional operation). This is an expert facility not used by regular JMS clients.

Specified by:

`createDurableConnectionConsumer` in interface `Connection`

Parameters:

`topic` - the topic to access

`subscriptionName` - durable subscription name

`messageSelector` - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.

`sessionPool` - the server session pool to associate with this durable connection consumer

`maxMessages` - the maximum number of messages that can be assigned to a server session at one time

Returns:

the durable connection consumer

Throws:

`JMSException` - if the `TopicConnection` object fails to create a connection consumer due to some internal error or invalid arguments for `sessionPool` and `messageSelector`.

`InvalidDestinationException` - if an invalid topic is specified.

`InvalidSelectorException` - if the message selector is invalid.

See Also:

`ConnectionConsumer`

---

javax.jms

Interface TopicConnectionFactory

All Superinterfaces:

ConnectionFactory

All Known Subinterfaces:

XATopicConnectionFactory

---

public interface **TopicConnectionFactory**
extends ConnectionFactory

A client uses a `TopicConnectionFactory` object to create `TopicConnection` objects with a publish/subscribe JMS provider.

A `TopicConnectionFactory` can be used to create a `TopicConnection`, from which specialized topic-related objects can be created. A more general, and recommended approach is to use the `ConnectionFactory` object.

The `TopicConnectionFactory` object should be used to support existing code.

Version:

1.1 - February 2, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`ConnectionFactory`

Method Summary

TopicConnection	**createTopicConnection**() Creates a topic connection with the default user identity.

TopicConnection	**createTopicConnection**(String userName, String password) Creates a topic connection with the specified user identity.

Methods inherited from interface javax.jms.ConnectionFactory
createConnection, createConnection

Method Detail

createTopicConnection

```
public TopicConnection createTopicConnection()
                                        throws JMSException
```

Creates a topic connection with the default user identity. The connection is created in stopped mode. No messages will be delivered until the `Connection.start` method is explicitly called.

Returns:
a newly created topic connection

Throws:
`JMSException` - if the JMS provider fails to create a topic connection due to some internal error.
`JMSSecurityException` - if client authentication fails due to an invalid user name or password.

---

createTopicConnection

```
public TopicConnection createTopicConnection(String userName,
                                             String password)
                                      throws JMSException
```

Creates a topic connection with the specified user identity. The connection is created in stopped mode. No messages will be delivered until the `Connection.start` method is explicitly called.

Parameters:
`userName` - the caller's user name

`password` - the caller's password

Returns:

a newly created topic connection

Throws:

`JMSException` - if the JMS provider fails to create a topic connection due to some internal error.

`JMSSecurityException` - if client authentication fails due to an invalid user name or password.

---

javax.jms

Interface TopicPublisher

All Superinterfaces:

MessageProducer

public interface **TopicPublisher**
extends MessageProducer

A client uses a `TopicPublisher` object to publish messages on a topic. A `TopicPublisher` object is the publish-subscribe form of a message producer.

Normally, the `Topic` is specified when a `TopicPublisher` is created. In this case, an attempt to use the `publish` methods for an unidentified `TopicPublisher` will throw a `java.lang.UnsupportedOperationException`.

If the `TopicPublisher` is created with an unidentified `Topic`, an attempt to use the `publish` methods that assume that the `Topic` has been identified will throw a `java.lang.UnsupportedOperationException`.

During the execution of its `publish` method, a message must not be changed by other threads within the client. If the message is modified, the result of the `publish` is undefined.

After publishing a message, a client may retain and modify it without affecting the message that has been published. The same message object may be published multiple times.

The following message headers are set as part of publishing a message: `JMSDestination`, `JMSDeliveryMode`, `JMSExpiration`, `JMSPriority`, `JMSMessageID` and `JMSTimeStamp`. When the message is published, the values of these headers are ignored. After completion of the `publish`, the headers hold the values specified by the method publishing the message. It is possible for the `publish` method not to set `JMSMessageID` and `JMSTimeStamp` if the setting of these headers is explicitly disabled by the `MessageProducer.setDisableMessageID` or `MessageProducer.setDisableMessageTimestamp` method.

Creating a `MessageProducer` provides the same features as creating a `TopicPublisher`. A

`MessageProducer` object is recommended when creating new code. The `TopicPublisher` is provided to support existing code.

Because `TopicPublisher` inherits from `MessageProducer`, it inherits the `send` methods that are a part of the `MessageProducer` interface. Using the `send` methods will have the same effect as using the `publish` methods: they are functionally the same.

Version:

1.1 February 2, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`Session.createProducer(Destination)`, `TopicSession.createPublisher(Topic)`

Method Summary

`Topic`	**`getTopic`**`()` Gets the topic associated with this `TopicPublisher`.
`void`	**`publish`**`(Message message)` Publishes a message to the topic.
`void`	**`publish`**`(Message message, int deliveryMode, int priority, long timeToLive)` Publishes a message to the topic, specifying delivery mode, priority, and time to live.
`void`	**`publish`**`(Topic topic, Message message)` Publishes a message to a topic for an unidentified message producer.
`void`	**`publish`**`(Topic topic, Message message, int deliveryMode, int priority, long timeToLive)` Publishes a message to a topic for an unidentified message producer, specifying delivery mode, priority and time to live.

Methods inherited from interface javax.jms.MessageProducer

`close`, `getDeliveryMode`, `getDestination`, `getDisableMessageID`, `getDisableMessageTimestamp`, `getPriority`, `getTimeToLive`, `send`, `send`, `send`, `send`, `setDeliveryMode`, `setDisableMessageID`, `setDisableMessageTimestamp`, `setPriority`, `setTimeToLive`

Method Detail

getTopic

```
public Topic getTopic()
                throws JMSException
```

Gets the topic associated with this `TopicPublisher`.

Returns:
this publisher's topic

Throws:
`JMSException` - if the JMS provider fails to get the topic for this `TopicPublisher` due to some internal error.

---

publish

```
public void publish(Message message)
             throws JMSException
```

Publishes a message to the topic. Uses the `TopicPublisher`'s default delivery mode, priority, and time to live.

Parameters:
`message` - the message to publish

Throws:
`JMSException` - if the JMS provider fails to publish the message due to some internal error.
`MessageFormatException` - if an invalid message is specified.
`InvalidDestinationException` - if a client uses this method with a `TopicPublisher` with an invalid topic.
`UnsupportedOperationException` - if a client uses this method with a `TopicPublisher` that did not specify a topic at creation time.

See Also:
`MessageProducer.getDeliveryMode()`, `MessageProducer.getTimeToLive()`, `MessageProducer.getPriority()`

---

publish

```
public void publish(Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive)
             throws JMSException
```

Publishes a message to the topic, specifying delivery mode, priority, and time to live.

Parameters:

`message` - the message to publish
`deliveryMode` - the delivery mode to use
`priority` - the priority for this message
`timeToLive` - the message's lifetime (in milliseconds)

Throws:

`JMSException` - if the JMS provider fails to publish the message due to some internal error.
`MessageFormatException` - if an invalid message is specified.
`InvalidDestinationException` - if a client uses this method with a `TopicPublisher` with an invalid topic.
`UnsupportedOperationException` - if a client uses this method with a `TopicPublisher` that did not specify a topic at creation time.

---

publish

```
public void publish(Topic topic,
                    Message message)
             throws JMSException
```

Publishes a message to a topic for an unidentified message producer. Uses the `TopicPublisher`'s default delivery mode, priority, and time to live.

Typically, a message producer is assigned a topic at creation time; however, the JMS API also supports unidentified message producers, which require that the topic be supplied every time a message is published.

Parameters:

`topic` - the topic to publish this message to
`message` - the message to publish

Throws:

`JMSException` - if the JMS provider fails to publish the message due to some internal error.

MessageFormatException - if an invalid message is specified.
InvalidDestinationException - if a client uses this method with an invalid topic.

See Also:

MessageProducer.getDeliveryMode(), MessageProducer.getTimeToLive(), MessageProducer.getPriority()

---

publish

```
public void publish(Topic topic,
                    Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive)
             throws JMSException
```

Publishes a message to a topic for an unidentified message producer, specifying delivery mode, priority and time to live.

Typically, a message producer is assigned a topic at creation time; however, the JMS API also supports unidentified message producers, which require that the topic be supplied every time a message is published.

Parameters:

`topic` - the topic to publish this message to
`message` - the message to publish
`deliveryMode` - the delivery mode to use
`priority` - the priority for this message
`timeToLive` - the message's lifetime (in milliseconds)

Throws:

JMSException - if the JMS provider fails to publish the message due to some internal error.
MessageFormatException - if an invalid message is specified.
InvalidDestinationException - if a client uses this method with an invalid topic.

Submit a bug or feature

javax.jms

Interface TopicSubscriber

All Superinterfaces:

MessageConsumer

public interface **TopicSubscriber**
extends MessageConsumer

A client uses a `TopicSubscriber` object to receive messages that have been published to a topic. A `TopicSubscriber` object is the publish/subscribe form of a message consumer. A `MessageConsumer` can be created by using `Session.createConsumer`.

A `TopicSession` allows the creation of multiple `TopicSubscriber` objects per topic. It will deliver each message for a topic to each subscriber eligible to receive it. Each copy of the message is treated as a completely separate message. Work done on one copy has no effect on the others; acknowledging one does not acknowledge the others; one message may be delivered immediately, while another waits for its subscriber to process messages ahead of it.

Regular `TopicSubscriber` objects are not durable. They receive only messages that are published while they are active.

Messages filtered out by a subscriber's message selector will never be delivered to the subscriber. From the subscriber's perspective, they do not exist.

In some cases, a connection may both publish and subscribe to a topic. The subscriber `NoLocal` attribute allows a subscriber to inhibit the delivery of messages published by its own connection.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it uses a durable `TopicSubscriber`. The JMS provider retains a record of this durable subscription and insures that all messages from the topic's publishers are retained until they are acknowledged by this durable subscriber or they have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies (within client identifier) each durable subscription it creates. Only one session at a time can have a `TopicSubscriber` for a particular durable

subscription.

A client can change an existing durable subscription by creating a durable `TopicSubscriber` with the same name and a new topic and/or message selector. Changing a durable subscription is equivalent to unsubscribing (deleting) the old one and creating a new one.

The `unsubscribe` method is used to delete a durable subscription. The `unsubscribe` method can be used at the `Session` or `TopicSession` level. This method deletes the state being maintained on behalf of the subscriber by its provider.

Creating a `MessageConsumer` provides the same features as creating a `TopicSubscriber`. To create a durable subscriber, use of `Session.CreateDurableSubscriber` is recommended. The `TopicSubscriber` is provided to support existing code.

Version:

1.1 - February 2, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`Session.createConsumer(javax.jms.Destination)`, `Session.createDurableSubscriber(javax.jms.Topic, java.lang.String)`, `TopicSession`, `TopicSession.createSubscriber(javax.jms.Topic)`, `MessageConsumer`

Method Summary

`boolean`	**`getNoLocal`**`()` Gets the `NoLocal` attribute for this subscriber.
`Topic`	**`getTopic`**`()` Gets the `Topic` associated with this subscriber.

Methods inherited from interface javax.jms.MessageConsumer

`close`, `getMessageListener`, `getMessageSelector`, `receive`, `receive`, `receiveNoWait`, `setMessageListener`

Method Detail

getTopic

```
public Topic getTopic()
                throws JMSException
```

Gets the `Topic` associated with this subscriber.

Returns:

this subscriber's `Topic`

Throws:

`JMSException` - if the JMS provider fails to get the topic for this topic subscriber due to some internal error.

---

getNoLocal

```
public boolean getNoLocal()
                  throws JMSException
```

Gets the `NoLocal` attribute for this subscriber. The default value for this attribute is false.

Returns:

true if locally published messages are being inhibited

Throws:

`JMSException` - if the JMS provider fails to get the `NoLocal` attribute for this topic subscriber due to some internal error.

---

javax.jms

Interface TopicSession

All Superinterfaces:

Runnable, Session

---

public interface **TopicSession**
extends Session

A `TopicSession` object provides methods for creating `TopicPublisher`, `TopicSubscriber`, and `TemporaryTopic` objects. It also provides a method for deleting its client's durable subscribers.

A `TopicSession` is used for creating Pub/Sub specific objects. In general, use the `Session` object, and use `TopicSession` only to support existing code. Using the `Session` object simplifies the programming model, and allows transactions to be used across the two messaging domains.

A `TopicSession` cannot be used to create objects specific to the point-to-point domain. The following methods inherit from `Session`, but must throw an `IllegalStateException` if used from `TopicSession`:

- `createBrowser`
- `createQueue`
- `createTemporaryQueue`

Version:

1.1 - April 9, 2002

Author:

Mark Hapner, Rich Burridge, Kate Stout

See Also:

`Session`, `Connection.createSession(boolean, int)`, `TopicConnection.createTopicSession(boolean, int)`, `XATopicSession.getTopicSession()`

Field Summary

Fields inherited from interface javax.jms.Session
AUTO_ACKNOWLEDGE, CLIENT_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE, SESSION_TRANSACTED

Method Summary

TopicSubscriber	**createDurableSubscriber**(Topic topic, String name) Creates a durable subscriber to the specified topic.
TopicSubscriber	**createDurableSubscriber**(Topic topic, String name, String messageSelector, boolean noLocal) Creates a durable subscriber to the specified topic, using a message selector or specifying whether messages published by its own connection should be delivered to it.
TopicPublisher	**createPublisher**(Topic topic) Creates a publisher for the specified topic.
TopicSubscriber	**createSubscriber**(Topic topic) Creates a nondurable subscriber to the specified topic.
TopicSubscriber	**createSubscriber**(Topic topic, String messageSelector, boolean noLocal) Creates a nondurable subscriber to the specified topic, using a message selector or specifying whether messages published by its own connection should be delivered to it.
TemporaryTopic	**createTemporaryTopic**() Creates a TemporaryTopic object.
Topic	**createTopic**(String topicName) Creates a topic identity given a Topic name.
void	**unsubscribe**(String name) Unsubscribes a durable subscription that has been created by a client.

Methods inherited from interface javax.jms.Session
close, commit, createBrowser, createBrowser, createBytesMessage, createConsumer, createConsumer, createConsumer, createMapMessage, createMessage, createObjectMessage, createObjectMessage, createProducer, createQueue, createStreamMessage, createTemporaryQueue, createTextMessage, createTextMessage, getAcknowledgeMode, getMessageListener, getTransacted, recover, rollback, run, setMessageListener

Method Detail

createTopic

```
public Topic createTopic(String topicName)
                  throws JMSException
```

Creates a topic identity given a `Topic` name.

This facility is provided for the rare cases where clients need to dynamically manipulate topic identity. This allows the creation of a topic identity with a provider-specific name. Clients that depend on this ability are not portable.

Note that this method is not for creating the physical topic. The physical creation of topics is an administrative task and is not to be initiated by the JMS API. The one exception is the creation of temporary topics, which is accomplished with the `createTemporaryTopic` method.

Specified by:
`createTopic` in interface `Session`

Parameters:
`topicName` - the name of this `Topic`

Returns:
a `Topic` with the given name

Throws:
`JMSException` - if the session fails to create a topic due to some internal error.

---

createSubscriber

```
public TopicSubscriber createSubscriber(Topic topic)
                                 throws JMSException
```

Creates a nondurable subscriber to the specified topic.

A client uses a `TopicSubscriber` object to receive messages that have been published to a topic.

Regular `TopicSubscriber` objects are not durable. They receive only messages that are published while they are active.

In some cases, a connection may both publish and subscribe to a topic. The subscriber `NoLocal` attribute allows a subscriber to inhibit the delivery of messages published by its own connection.

The default value for this attribute is false.

Parameters:

`topic` - the `Topic` to subscribe to

Throws:

`JMSException` - if the session fails to create a subscriber due to some internal error.

`InvalidDestinationException` - if an invalid topic is specified.

---

createSubscriber

```
public TopicSubscriber createSubscriber(Topic topic,
                                        String messageSelector,
                                        boolean noLocal)
                                 throws JMSException
```

Creates a nondurable subscriber to the specified topic, using a message selector or specifying whether messages published by its own connection should be delivered to it.

A client uses a `TopicSubscriber` object to receive messages that have been published to a topic.

Regular `TopicSubscriber` objects are not durable. They receive only messages that are published while they are active.

Messages filtered out by a subscriber's message selector will never be delivered to the subscriber. From the subscriber's perspective, they do not exist.

In some cases, a connection may both publish and subscribe to a topic. The subscriber `NoLocal` attribute allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is false.

Parameters:

`topic` - the `Topic` to subscribe to

`messageSelector` - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.

`noLocal` - if set, inhibits the delivery of messages published by its own connection

Throws:

`JMSException` - if the session fails to create a subscriber due to some internal error.

`InvalidDestinationException` - if an invalid topic is specified.

`InvalidSelectorException` - if the message selector is invalid.

---

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber(Topic topic,
                                                String name)
                                         throws JMSException
```

Creates a durable subscriber to the specified topic.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it uses a durable `TopicSubscriber`. The JMS provider retains a record of this durable subscription and insures that all messages from the topic's publishers are retained until they are acknowledged by this durable subscriber or they have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies (within client identifier) each durable subscription it creates. Only one session at a time can have a `TopicSubscriber` for a particular durable subscription.

A client can change an existing durable subscription by creating a durable `TopicSubscriber` with the same name and a new topic and/or message selector. Changing a durable subscriber is equivalent to unsubscribing (deleting) the old one and creating a new one.

In some cases, a connection may both publish and subscribe to a topic. The subscriber `NoLocal` attribute allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is false.

Specified by:

`createDurableSubscriber` in interface `Session`

Parameters:

`topic` - the non-temporary `Topic` to subscribe to
`name` - the name used to identify this subscription

Throws:

`JMSException` - if the session fails to create a subscriber due to some internal error.
`InvalidDestinationException` - if an invalid topic is specified.

---

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber(Topic topic,
                                                String name,
                                                String messageSelector,
```

```
                                    boolean noLocal)
                             throws JMSException
```

Creates a durable subscriber to the specified topic, using a message selector or specifying whether messages published by its own connection should be delivered to it.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it uses a durable `TopicSubscriber`. The JMS provider retains a record of this durable subscription and insures that all messages from the topic's publishers are retained until they are acknowledged by this durable subscriber or they have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name which uniquely identifies (within client identifier) each durable subscription it creates. Only one session at a time can have a `TopicSubscriber` for a particular durable subscription. An inactive durable subscriber is one that exists but does not currently have a message consumer associated with it.

A client can change an existing durable subscription by creating a durable `TopicSubscriber` with the same name and a new topic and/or message selector. Changing a durable subscriber is equivalent to unsubscribing (deleting) the old one and creating a new one.

Specified by:

`createDurableSubscriber` in interface `Session`

Parameters:

`topic` - the non-temporary `Topic` to subscribe to

`name` - the name used to identify this subscription

`messageSelector` - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.

`noLocal` - if set, inhibits the delivery of messages published by its own connection

Throws:

`JMSException` - if the session fails to create a subscriber due to some internal error.

`InvalidDestinationException` - if an invalid topic is specified.

`InvalidSelectorException` - if the message selector is invalid.

---

createPublisher

```
public TopicPublisher createPublisher(Topic topic)
                               throws JMSException
```

Creates a publisher for the specified topic.

A client uses a `TopicPublisher` object to publish messages on a topic. Each time a client creates a `TopicPublisher` on a topic, it defines a new sequence of messages that have no ordering relationship with the messages it has previously sent.

Parameters:

`topic` - the `Topic` to publish to, or null if this is an unidentified producer

Throws:

`JMSException` - if the session fails to create a publisher due to some internal error.

`InvalidDestinationException` - if an invalid topic is specified.

---

createTemporaryTopic

```
public TemporaryTopic createTemporaryTopic()
                                  throws JMSException
```

Creates a `TemporaryTopic` object. Its lifetime will be that of the `TopicConnection` unless it is deleted earlier.

Specified by:

`createTemporaryTopic` in interface `Session`

Returns:

a temporary topic identity

Throws:

`JMSException` - if the session fails to create a temporary topic due to some internal error.

---

unsubscribe

```
public void unsubscribe(String name)
                 throws JMSException
```

Unsubscribes a durable subscription that has been created by a client.

This method deletes the state being maintained on behalf of the subscriber by its provider.

It is erroneous for a client to delete a durable subscription while there is an active `TopicSubscriber` for the subscription, or while a consumed message is part of a pending transaction or has not been acknowledged in the session.

Specified by:

`unsubscribe` in interface `Session`

Parameters:

`name` - the name used to identify this subscription

Throws:

`JMSException` - if the session fails to unsubscribe to the durable subscription due to some internal error.

`InvalidDestinationException` - if an invalid subscription name is specified.

Overview Package Class Tree Deprecated Index Help *Java™ 2 Platform Ent. Ed. v1.4*

PREV CLASS NEXT CLASS **FRAMES NO FRAMES All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Submit a bug or feature

jeckle.de home

TODO

Sie sind durch einen Verweis von (noch) fehlender Information auf jeckle.de hierher gelangt.

Hierbei handelt es sich zumeist um Vorankündigungen, zu denen noch nicht alle Detaildaten verfügbar sind oder Vorabversionen von Web-Seiten, die nochmals überarbeitet werden (sollen). Wenn eine durch Sie angeklickte Information noch nicht vorliegt, so sollte sie jedoch in Kürze auf meinen Web-Seiten bereitgestellt werden.

Für Ergänzungen, Kommentare oder Hinweise zur Behebung der aufgetretenen Lücke bin ich jederzeit dankbar!

Ihr Name:

Ihre Email-Adresse:

ab die Post!

Service provided by Mario Jeckle

Generated: 2004-05-24T13:30:44+01:00

Feedback SiteMap

This page's original location: http://www.jeckle.de/todo/

RDF description for this page

```
import java.util.Date;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Queue;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.Topic;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
public class SProducer {
	public static void main(String[] args) {
		final int NUM_MSGS;
		if ((args.length < 2) || (args.length > 3)) {
			System.out.println("Program takes two or three arguments: "+ "<dest_name> <queue|
topic> " + "[<number-of-messages>");
			System.exit(1);
		}
		String destName = new String(args[0]);
		String destType = new String(args[1]);
		System.out.println("Destination name is " + destName + ", type is " + destType);
		if (args.length == 3) {
			NUM_MSGS = (new Integer(args[2])).intValue();
		} else {
			NUM_MSGS = 1;
		}
		Context jndiContext = null;
		try {
			jndiContext = new InitialContext();
		} catch (NamingException e) {
			System.out.println("Could not create JNDI " + "context: "
					+ e.toString());
			System.exit(1);
		}
		ConnectionFactory connectionFactory = null;
		Destination dest = null;
		try {
			connectionFactory = (ConnectionFactory) jndiContext.lookup("jms/
QueueConnectionFactory");
			if (destType.equals("queue")) {
				dest = (Queue) jndiContext.lookup(destName);
			} else if (destType.equals("topic")) {
```

```
                dest = (Topic) jndiContext.lookup(destName);
            } else {
                throw new Exception("Invalid destination type" + "; must be queue or topic");
            }
        } catch (Exception e) {
            System.out.println("JNDI lookup failed: " + e.toString());
            e.printStackTrace();
            System.exit(1);
        }
        Connection connection = null;
        MessageProducer producer = null;
        try {
            connection = connectionFactory.createConnection();
            Session session = connection.createSession(false,
                    Session.AUTO_ACKNOWLEDGE);
            producer = session.createProducer(dest);
            TextMessage message = session.createTextMessage();
            for (int i = 0; i < NUM_MSGS; i++) {
                message.setText("This is message " + (i + 1) + " sent at "
                        + new Date());
                System.out.println("Sending message: " + message.getText());
                producer.send(message);
            }
            producer.send(session.createMessage());
        } catch (JMSException e) {
            System.out.println("Exception occurred: " + e.toString());
        } finally {
            if (connection != null) {
                try {
                    connection.close();
                } catch (JMSException e) {
                    System.err.println("JMSException occured while closing connection");
                }
            }
        }
    }
}
```

```
Destination name is jms/Queue, type is queue
Sending message: This is message 1 sent at Mon May 24 11:45:28 CEST 2004
Sending message: This is message 2 sent at Mon May 24 11:45:29 CEST 2004
Sending message: This is message 3 sent at Mon May 24 11:45:29 CEST 2004
Sending message: This is message 4 sent at Mon May 24 11:45:29 CEST 2004
Sending message: This is message 5 sent at Mon May 24 11:45:29 CEST 2004
```

```
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.Queue;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.Topic;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
public class SConsumer {
	public static void main(String[] args) {
		String destName = null;
		String destType = null;
		Context jndiContext = null;
		ConnectionFactory connectionFactory = null;
		Connection connection = null;
		Session session = null;
		Destination dest = null;
		MessageConsumer consumer = null;
		TextMessage message = null;
		if (args.length != 2) {
			System.out.println("Program takes two arguments: " + "<dest_name> <queue|topic>");
			System.exit(1);
		}
		destName = new String(args[0]);
		destType = new String(args[1]);
		System.out.println("Destination name is " + destName + ", type is "
				+ destType);
		try {
			jndiContext = new InitialContext();
		} catch (NamingException e) {
			System.out.println("Could not create JNDI " + "context: "
					+ e.toString());
			System.exit(1);
		}
		try {
			connectionFactory = (ConnectionFactory) jndiContext
					.lookup("jms/QueueConnectionFactory");
			if (destType.equals("queue")) {
				dest = (Queue) jndiContext.lookup(destName);
			} else if (destType.equals("topic")) {
```

```
                dest = (Topic) jndiContext.lookup(destName);
            } else {
                throw new Exception("Invalid destination type" + "; must be queue or topic");
            }
        } catch (Exception e) {
            System.out.println("JNDI lookup failed: " + e.toString());
            System.exit(1);
        }
        try {
            connection = connectionFactory.createConnection();
            session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            consumer = session.createConsumer(dest);
            connection.start();
            Message m;
            while (true) {
                m = consumer.receive(1);
                if (m != null) {
                    if (m instanceof TextMessage) {
                        message = (TextMessage) m;
                        System.out.println("Reading message: " + message.getText());
                    } else
                        break;
                }
            }
        } catch (JMSException e) {
            System.out.println("Exception occurred: " + e.toString());
        } finally {
            if (connection != null) {
                try {
                    connection.close();
                } catch (JMSException e) {
                    System.err.println("JMSException occured while closing connection");
                }
            }
        }
    }
}
```

```
Destination name is jms/Queue, type is queue
Reading message: This is message 1 sent at Mon May 24 11:45:28 CEST 2004
Reading message: This is message 2 sent at Mon May 24 11:45:29 CEST 2004
Reading message: This is message 3 sent at Mon May 24 11:45:29 CEST 2004
Reading message: This is message 4 sent at Mon May 24 11:45:29 CEST 2004
Reading message: This is message 5 sent at Mon May 24 11:45:29 CEST 2004
```

```
import javax.jms.DeliveryMode;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.Topic;
import javax.jms.TopicConnection;
import javax.jms.TopicConnectionFactory;
import javax.jms.TopicPublisher;
import javax.jms.TopicSession;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class PSProducer {
	public static void main(String[] args) {
		Topic t = null;
		TopicConnectionFactory tcf = null;
		TopicConnection tc = null;

		try {
			Context jndiContext = new InitialContext();
			tcf = (TopicConnectionFactory) jndiContext.lookup("jms/TopicConnectionFactory");
			//t = (Topic) jndiContext.lookup("StockQuote");
		} catch (NamingException ne) {
			System.out.println(ne+ " does not exist");
			System.exit(1);
		}

		try {
			tc = tcf.createTopicConnection();
			TopicSession ts = tc.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
			t = ts.createTopic("StockQuote");

			TopicPublisher tp = ts.createPublisher(t);

			TextMessage msg1 = ts.createTextMessage();
			TextMessage msg2 = ts.createTextMessage();

			msg1.setText("Quote");
			msg1.setStringProperty("Company","DCX");
			msg1.setDoubleProperty("USD",41.67);
			msg1.setDoubleProperty("EUR",34.99);
			msg1.setStringProperty("Date","2004-05-23");
```

```
                tp.publish(msg1, DeliveryMode.PERSISTENT, Message.DEFAULT_PRIORITY,
7*24*3600*1000L);

                try {
                    Thread.sleep(30000);
                } catch(InterruptedException ie) {
                    System.err.println("InterruptedException caught");
                }

                msg2.setText("Quote");
                msg2.setStringProperty("Company","AZ");
                msg2.setDoubleProperty("USD",10.04);
                msg2.setDoubleProperty("EUR",9.30);
                msg2.setStringProperty("Date","2004-05-23");

                tp.publish(msg2, DeliveryMode.PERSISTENT, Message.DEFAULT_PRIORITY,
7*24*3600*1000L);
            } catch (JMSException e) {
                System.err.println("JMSException caught");
                e.printStackTrace();
            }
        }
}
```

```
import java.util.Date;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.Topic;
import javax.jms.TopicConnection;
import javax.jms.TopicConnectionFactory;
import javax.jms.TopicSession;
import javax.jms.TopicSubscriber;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
public class PSConsumer {
	public static void main(String[] args) {
		Topic t = null;
		TopicConnectionFactory tcf = null;
		TopicConnection tc = null;

		try {
			Context jndiContext = new InitialContext();
			tcf = (TopicConnectionFactory) jndiContext.lookup("jms/TopicConnectionFactory");
			//t = (Topic) jndiContext.lookup("StockQuote");
		} catch (NamingException ne) {
			System.err.println(ne+" does not exist");
			System.exit(1);
		}

		try {
			tc = tcf.createTopicConnection();
			tc.setClientID("MariosClient");
			TopicSession ts = tc.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
			t = ts.createTopic("StockQuote");

			TopicSubscriber tSubDCX=ts.createDurableSubscriber(t,"mySub", "Company LIKE
'%DCX'",false);
			TopicSubscriber tSubAll=ts.createSubscriber(t);

			tSubDCX.setMessageListener( new MyTopicListener("DCX") );
			tSubAll.setMessageListener( new MyTopicListener("All"));

			tc.start();
```

```
                for (;;){
                    System.out.println("("+new Date()+") waiting for messages...");
                    try {
                        Thread.sleep(1000);
                    } catch(InterruptedException ie) {
                        System.err.println("InterruptedException caught");
                    }
                }

            } catch (JMSException e) {
                System.err.println("JMSException caught");
                e.printStackTrace();
            }
        }
}

class MyTopicListener implements MessageListener {
        private String title;

        public MyTopicListener(String title){
            this.title = title;
            System.out.println("Message listener "+title+" created");
        }

        public void onMessage(Message msg) {
            try {
            TextMessage tm = (TextMessage) msg;
            System.out.println("("+this.title+") Message received: "+tm.getText());
            System.out.println("Company="+tm.getStringProperty("Company"));
            System.out.println("USD="+tm.getDoubleProperty("USD"));
            System.out.println("EUR="+tm.getStringProperty("EUR"));
            System.out.println("--------------------------");
            } catch (JMSException je) {
                System.err.println("JMSException caught");
                je.printStackTrace();
            }
        }

}
```

javax.jms

Interface MessageListener

public interface **MessageListener**

A `MessageListener` object is used to receive asynchronously delivered messages.

Each session must insure that it passes messages serially to the listener. This means that a listener assigned to one or more consumers of the same session can assume that the `onMessage` method is not called with the next message until the session has completed the last call.

Version:

1.0 - 13 March 1998

Author:

Mark Hapner, Rich Burridge

Method Summary

`void`	**`onMessage`**`(Message message)` Passes a message to the listener.

Method Detail

onMessage

`public void` **`onMessage`**`(Message message)`

Passes a message to the listener.

Parameters:

`message` - the message passed to the listener

Submit a bug or feature

javax.ejb

Interface MessageDrivenBean

All Superinterfaces:

EnterpriseBean, Serializable

public interface **MessageDrivenBean**
extends EnterpriseBean

The MessageDrivenBean interface is implemented by every message-driven enterprise Bean class. The container uses the MessageDrivenBean methods to notify the enterprise Bean instances of the instance's life cycle events.

Method Summary

void	**ejbRemove**() A container invokes this method before it ends the life of the message-driven object.
void	**setMessageDrivenContext**(MessageDrivenContext ctx) Set the associated message-driven context.

Method Detail

setMessageDrivenContext

```
public void setMessageDrivenContext(MessageDrivenContext ctx)
                             throws EJBException
```

Set the associated message-driven context. The container calls this method after the instance creation.

The enterprise Bean instance should store the reference to the context object in an instance

variable.

This method is called with no transaction context.

Parameters:

`ctx` - A MessageDrivenContext interface for the instance.

Throws:

`EJBException` - Thrown by the method to indicate a failure caused by a system-level error.

---

ejbRemove

```
public void ejbRemove()
                throws EJBException
```

A container invokes this method before it ends the life of the message-driven object. This happens when a container decides to terminate the message-driven object.

This method is called with no transaction context.

Throws:

`EJBException` - Thrown by the method to indicate a failure caused by a system-level error.

```
import javax.ejb.CreateException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

public class MDBeanExample implements MessageDrivenBean, MessageListener{
	private MessageDrivenContext ctx;
	private TextMessage tm;

	public void setMessageDrivenContext(MessageDrivenContext ctx) {
		this.ctx = ctx;
	}

	public void ejbCreate() throws CreateException {
		System.err.println("Message-driven Bean created");
	}

	public void ejbRemove() {
		//does nothing
	}

	public void onMessage(Message msg)  {
		if (msg instanceof TextMessage) {
			this.tm = (TextMessage) msg;
			try {
				System.out.println("Message received: "+this.tm.getText());
				System.out.println("Company="+this.tm.getStringProperty("Company"));
				System.out.println("USD="+this.tm.getDoubleProperty("USD"));
				System.out.println("EUR="+this.tm.getStringProperty("EUR"));
				System.out.println("--------------------------");
			} catch (JMSException je) {
				System.err.println("JMSException caught");
				je.printStackTrace();
			}
		}
	}
}
```

Message-Oriented Middleware

Software Technology Roadmap

Status

Advanced

Note

We recommend Middleware *as prerequisite reading for this technology description.*

Purpose and Origin

Message-oriented middleware (MOM) is a client/server infrastructure that increases the *interoperability*, *portability*, and *flexibility* of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the *complexity* of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces- Application Programming Interfaces (APIs) that extend across diverse platforms and networks are typically provided by the MOM [Rao 95].

Technical Detail

Message-oriented middleware, as shown in Figure 22 [Steinke 95], is software that resides in both portions of a client/server architecture and typically supports asynchronous calls between the client and server applications. Message queues provide temporary storage when the destination program is busy or not connected. MOM reduces the involvement of application developers with the *complexity* of the master-slave nature of the client/server mechanism.

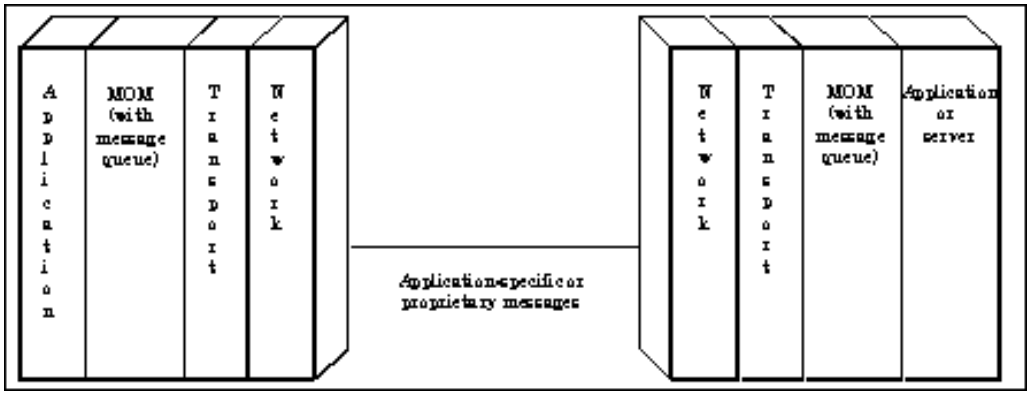


Figure 22: Message-Oriented Middleware

MOM increases the flexibility of an architecture by enabling applications to exchange messages with other programs without having to know what platform or processor the other application resides on within the network. The aforementioned messages can contain formatted data, requests for action, or both. Nominally, MOM systems provide a message queue between interoperating processes, so if the destination process is busy, the message is held in a temporary storage location until it can be processed. MOM is typically asynchronous and peer-to-peer, but most implementations support synchronous message passing as well.

Usage Considerations

MOM is most appropriate for event-driven applications. When an event occurs, the client application hands off to the messaging middleware application the responsibility of notifying a server that some action needs to be taken. MOM is also well-suited for object-oriented systems because it furnishes a conceptual mechanism for peer-to-peer communications between objects. MOM insulates developers from connectivity concerns- the application developers write to APIs that handle the complexity of the specific interfaces.

Asynchronous and synchronous mechanisms each have strengths and weaknesses that should be considered when designing any specific application. The asynchronous mechanism of MOM, unlike Remote Procedure Call (RPC) , which uses a a synchronous, blocking mechanism, does not guard against overloading a network. As such, a negative aspect of MOM is that a client process can continue to transfer data to a server that is not keeping pace. Message-oriented middleware's use of message queues, however, tends to be more flexible than RPC-based systems, because most implementations of MOM can default to synchronous and fall back to asynchronous communication if a server becomes unavailable [Steinke 95].

Maturity

Implementations of MOM first became available in the mid-to-late 1980s. Many MOM implementations currently exist that support a variety of protocols and operating systems. Many implementations support multiple protocols and operating systems simultaneously.

Some vendors provide tool sets to help extend existing interprocess communication across a heterogeneous network.

Costs and Limitations

MOM is typically implemented as a proprietary product, which means MOM implementations are nominally incompatible with other MOM implementations. Using a single implementation of a MOM in a system will most likely result in a dependence on the MOM vendor for maintenance support and future enhancements. This could have a highly negative impact on a system's flexibility, maintainability, portability, and interoperability.

The message-oriented middleware software (kernel) must run on every platform of a network. The impact of this varies and depends on the characteristics of the system in which the MOM will be used:

- Not all MOM implementations support all operating systems and

protocols. The flexibility to choose a MOM implementation may be dependent on the chosen application platform or network protocols supported, or vice versa.
- Local resources and CPU cycles must be used to support the MOM kernels on each platform. The performance impact of the middleware implementation must be considered; this could possibly require the user to acquire greater local resources and processing power.
- The administrative and maintenance burden would increase significantly for a network manager with a large distributed system, especially in a mostly heterogeneous system.
- A MOM implementation may cost more if multiple kernels are required for a heterogeneous system, especially when a system is maintaining kernels for old platforms and new platforms simultaneously.

Alternatives

Other infrastructure technologies that allow the distribution of processing across multiple processors and platforms are

- Object Request Broker (ORB)
- Distributed Computing Environment (DCE)
- Remote Procedure Call (RPC)
- Transaction Processing Monitor Technology
- Three Tier Software Architectures

Complementary Technologies

MOM can be effectively combined with remote procedure call (RPC) technology- RPC can be used for synchronous support by a MOM.

Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

Name of technology	Message-Oriented Middleware
Application category	Client/Server (AP.2.1.2.1) Client/Server Communication (AP.2.2.1)
Quality measures category	Maintainability (QM.3.1) Interoperability (QM.4.1) Portability (QM.4.2)
Computing reviews category	Distributed Systems (C.2.4) Network Architecture and Design (C.2.1)

References and Information Sources

[Rao 95]	Rao, B.R. "Making the Most of Middleware." *Data Communications International 24*, 12 (September 1995): 89-96.
[Steinke 95]	Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

Current Author/Maintainer

Cory Vondrak, TRW, Redondo Beach, CA

External Reviewers

Ed Morris, SEI

Modifications

10 Jan 97 (original)

URL: http://www.sei.cmu.edu/str/descriptions/momt_body.html
Last Modified: 9 January 2004

Section Explanations, References, Terms, Footnotes, and Related Topics

This frame provides additional information, including:

- Explanation of the purpose of various sections
- Full citations for references
- Definitions of italicized terms
- Expansion of footnotes
- Lists of related topics

Reference

Java Message Service API - FAQ

Downloads

Reference
- Documentation
- FAQs
- Licensees

Community
- Forums
- Bug Database

Learning
- Instructor-Led Courses
- Tutorials & Code Camps
- Online Sessions & Courses

FREQUENTLY ASKED QUESTIONS

General Questions

Technical questions

General questions

Q: What is the Java Message Service?
A: The Java Message Service (JMS) API is an API for accessing enterprise messaging systems. It is part of the Java 2 Platform, Enterprise Edition (J2EE).

The Java Message Service makes it easy to write business applications that asynchronously send and receive critical business data and events.

The Java Message Service defines a common enterprise messaging API that is designed to be easily and efficiently supported by a wide range of enterprise messaging products.

The Java Message Service supports both message queueing and publish-subscribe styles of messaging.

Q: Is the Java Message Service another mail API?
A: No. The term messaging is broadly defined in computing. It is used for describing various operating system concepts; it is used to describe email and fax systems; and with the JMS API, it is used to describe asynchronous communication between enterprise applications.

JMS messages are asynchronous requests, reports, or events that are consumed by enterprise applications, not humans. They contain vital information needed to coordinate these systems. They contain precisely formatted data that describe specific business actions. Through the exchange of these messages, each application tracks the progress of the enterprise.

Q: Is the Java Message Service a product?
A: No, the Java Message Service is the specification of a common API for enterprise messaging. A JMS provider supplied by an enterprise messaging vendor is required to use it.

Q: Where can I find the Java Message Service specification?
A: The Java Message Service 1.0.2 specification is available at http://java.sun.com/products/jms/docs.html. A link to the online JMS API documentation is at the same location.

Q: Who was involved with Sun in the creation of the specification of Java Message Service 1.0.2?
A: A number of important industry players initially collaborated with Sun to define the first draft of the Java Message Service specification. In addition, many comments were received from other companies, government and educational organizations, and others during the three-month public review period.

Q: What is compelling about the Java Message Service?
A: The Java Message Service is compelling for these reasons:

- It is the first enterprise messaging API that has achieved wide industry support.
- It simplifies the development of enterprise applications by providing standard messaging concepts and conventions that apply across a wide range of enterprise messaging systems.
- It leverages existing, enterprise-proven messaging systems.

Q: Why should developers use the Java Message Service?
A: The Java Message Service makes it easy to do the following:

- Write portable, message-based business applications
- Extend existing message-based applications by adding new JMS clients that interoperate fully with their existing non-JMS clients

Q: What does a programmer need to learn to use the Java Message Service?
A: Message-based applications are fundamentally different from remote procedure call based applications. Once a developer understands how best to employ both technologies, the JMS API makes writing message-based applications as easy as learning a few additional interfaces.

Q: What is the relationship between the Java Message Service and Enterprise JavaBeans components?
A: Since J2EE version 1.2 was released, Enterprise JavaBeans components have been able to use the JMS API to send enterprise messages and receive them synchronously. The Java 2 SDK, Enterprise Edition, version 1.3, now available, provides a new kind of enterprise bean, the message-driven bean, that allows an enterprise application to receive messages asynchronously.

Q: What is the relationship between the Java Message Service and the Java Naming and Directory Interface (JNDI) API?
A: The JMS API, like the other Java Enterprise APIs, uses the

JNDI API for administration. The JMS API defines ConnectionFactories and Destinations as administered objects that are configured and placed in a JNDI naming context. JMS clients then look up and use these preconfigured objects. This insures that JMS applications are easy to deploy and administer.

Q: What is the relationship between the Java Message Service and the Java DataBase Connectivity (JDBC) API?
A: JMS clients may also use the JDBC API. They may use both the JMS API and the JDBC API in the same transaction. In most cases, they will achieve this automatically by implementing these clients as enterprise beans. They may also use the Java Transaction API.

Q: What is the relationship between the Java Message Service, the Java Transaction API, and the Java Transaction Service?
A: The Java Transaction API (JTA) provides a client API for delimiting distributed transactions and an API for accessing a resource's ability to participate in a distributed transaction. A JMS client may use JTA to delimit distributed transactions. A JMS provider can optionally support distributed transactions via JTA.

The Java Transaction Service (JTS) can be used with the JMS API to form distributed transactions that combine message sends and receives with database updates and other JTS aware services. These services should be handled automatically when a JMS client is run from within an application server such as a J2EE server; however, it is also possible for JMS clients to program them explicitly.

Q: Where can I find a JMS API discussion group?
A: You can find a JMS API discussion forum at http://java.sun.com/j2ee/community/forums/index.html. You must be a member of the Java Developer Connection to join this forum.

Q: Why is the JMS API promoted as a technology for providing communication between components within an enterprise, but not for business-to-business (B2B) communication between enterprises over the Internet? What prevents its use over the Internet?
A: The main issue in providing Internet-based JMS API messaging is that JMS API vendors would have to agree on a common (interoperable) wire format, and the JMS specification would have to specify a router-router API.

B2B messaging is an important area with its own unique requirements. This area is being addressed by emerging standards such as the Electronic Business XML initiative, ebXML (see http://www.ebxml.org/), where Sun is playing an active role. The ebXML initiative currently defines a Messaging Service (see the ebXML Specifications page), which supports XML messaging over the Internet. Current ebXML specifications focus on providing point-to-point asynchronous communication between parties over the Internet using multiple transport protocols such as HTTP, mail, and FTP. Work is also planned in ebXML to provide publish/subscribe XML messaging. Sun and its partners are currently defining a new Java API for XML Messaging, JAXM (see JSR #000067, Java APIs for XML Messaging 1.0), which supports the ebXML Messaging Service specification. An early-access version of JAXM is available via the Java Developer Connection at The M Project 1.0 Early Access.

In summary, the JMS API is intended for use within an enterprise, while XML messaging between independent enterprises over the Internet is being supported by the ebXML initiative and the new Java APIs (including JAXM) that support various ebXML

specifications.

Technical questions

Q: What JDK release does the JMS API need?
A: The JMS API requires JDK/JRE release 1.1.x or higher.

Q: Does the JMS API provide a version of a distributed Java event model?
A: The JMS API can be used, in general, as a notification service; however, it does not define a distributed version of Java Events. One alternative for implementing distributed Java Events would be as JavaBeans components that transparently, to the event producer and listener beans, distribute the events via the JMS API.

Q: Why are there two separate JMS API domains, point-to-point and publish/subscribe, instead of just one?
A: Even though there are many similarities, providing separate domains still seems to be important. It means that vendors aren't forced to support facilities out of their domain and that client code can be a bit more portable because products more fully support a domain (as opposed to supporting less defined subsets of a merged domain).

Q: Why doesn't the JMS API specify a set of JavaBeans components?
A: The JMS API is a low-level API, and like other low-level Java APIs, it doesn't lend itself to direct representation as JavaBeans components.

Q: How is the JMS API aligned with the CORBA Notification Service?
A: The Notification Service adds filtering, delivery guarantee semantics, durable connections, and the assembly of event networks to the CORBA Event Service. It gets its delivery guarantee semantics from the CORBA Messaging Service (which defines asynchronous CORBA method invocation).

Java technology is well integrated with CORBA. It provides Java IDL and COS Naming. In addition, OMG has recently defined RMI over IIOP.

It is expected that most use of IIOP from Java will be via RMI. It is expected that most use of COS Naming from Java will be via the JNDI API (Java Naming and Directory Service). The JMS API is a Java specific API designed to be layered over a wide range of existing and future Message Oriented Middleware (MOM) systems (just as the JNDI API is layered over existing name and directory services).

Q: Why doesn't the JMS API provide end-to-end synchronous message delivery and notification of delivery?
A: Some messaging systems provide synchronous delivery to destinations as a mechanism for implementing reliable applications. Some systems provide clients with various forms of delivery notification so that the clients can detect dropped or ignored messages. This is not the model defined by the JMS API.

JMS API messaging provides guaranteed delivery via the once-and-only-once delivery semantics of `PERSISTENT` messages. In addition, message consumers can insure reliable processing of messages by using either `CLIENT_ACKNOWLEDGE` mode or transacted sessions. This achieves reliable delivery with minimum synchronization and is the enterprise messaging model most vendors and developers prefer.

The JMS API does not define a schema of systems messages (such as delivery notifications). If an application requires acknowledgment of message receipt, it can define an application-level acknowledgment message.

These issues are more clearly understood when they are examined in the context of publish/subscribe applications. In this context, synchronous delivery and/or system acknowledgment of receipt are not an effective mechanism for implementing reliable applications (because producers by definition are not, and do not want to be, responsible for end-to-end message delivery).

Q: Why doesn't the JMS API provide a send-to-list mechanism?
A: Currently the JMS API provides a number of message send options; however, messages can only be sent to one Destination at a time.

The benefit of send-to-list is slightly less work for the programmer and the potential for the JMS provider to optimize the fact that several destinations are being sent the same message.

The down side of a send-to-list mechanism is that the list is, in effect, a group that is implemented and maintained by the client. This would complicate the administration of JMS clients.

Instead of the JMS API providing a send-to-list mechanism, it is recommended that providers support configuring destinations that represent a group. This allows a client to reach all consumers with a single send, while insuring that groups are properly administrable.

Q: Why doesn't the JMS API provide subscription notification? If it were possible for a publisher to detect when subscribers for a topic existed, it could inhibit publication on unsubscribed topics.
A: Although there may be some benefit in providing publishers with a mechanism for inhibiting publication to unsubscribed topics, the complexity this would add to the JMS API and the additional provider overhead it would require are not justified by its potential benefits. Instead, JMS providers should insure that they minimize the overhead for handling messages published to an unsubscribed topic.

Q: Can a message be acknowledged after its message consumer is closed?
A: Yes. Since message acknowledgment processing is performed at the session level, message acknowledgement is still relevant after a consumer is closed. All messages consumed by the session are acknowledged for the following two examples:

```
// CLIENT_ACKNOWLEDGE session
Message msg1 = topicSubscriber1.receive();
Message msg2 = topicSubscriber2.receive();
topicSubscriber1.close();
msg2.acknowledge();

// transacted session
Message msg1 = queueReceiver.receive();
queueReceiver.close();
session.commit();
```

After the session has been closed, however, a call to the `Message.acknowledge` or `Session.commit` method throws an `IllegalStateException`. If `close` is called on a transacted session before the transaction in progress is committed, the transaction is rolled back.

Note that in order to prevent duplicate delivery of a message from a durable subscription or queue, a message that can still be acknowledged by a session cannot be redelivered to another message consumer. The message can only be redelivered to another message consumer when it can no longer be acknowledged by the session that initially received the message.

Q: What message distribution policy does the JMS API specify when there are two or more `QueueReceiver`s for the same queue at the same time?
A: The JMS API does not specify a message distribution policy when two or more `QueueReceiver`s are registered with a destination at the same time. JMS API semantics account for only one `QueueReceiver` at any point in time. The JMS API does not prohibit multiple `QueueReceiver`s for a queue; it just does not define the behavior for this case.

Q: How does `CLIENT_ACKNOWLEDGE` mode work when there are two or more `TopicSubscriber`s to the same topic within a session?
A: From Section 6.11 of the JMS 1.0.2 Specification:

> A `TopicSession` allows the creation of multiple `TopicSubscriber`s per destination, it will deliver each message for a destination to each `TopicSubscriber` eligible to receive it. Each copy of the message is treated as a completely separate message. Work done on one copy has no affect on the other; acknowledging one does not acknowledge the other; one message may be delivered immediately while another waits for its consumer to process messages ahead of it.

The `Message.acknowledge` method is documented to acknowledge the receipt of all messages consumed by the session. Thus, if `msgA` is delivered to topic subscribers `TS1` and `TS2` of the same session, and `TS1` synchronously receives its copy of `msgA` and acknowledges it, `TS2` will still be able to synchronously receive its copy of the message. However, if `TS1` receives its copy of `msgA` and then `TS2` receives its copy of `msgA`, acknowledging receiving the message from `TS2` will acknowledge both the `msgA`s received by `TS1` and `TS2`, based on the fact that `TS1`'s copy of `msgA` was received by the session before `TS2`'s copy of `msgA` was received by the session.

Q: What is the recoverability of messages for closed consumers?
A: For messages consumed from a queue or durable subscription, a rollback ensures that the messages are not acknowledged and that they will be delivered to the next consumer of these persistent entities. The JMS specification states that messages delivered to non-durable subscribers may be dropped. This statement means that the JMS API does not define the recoverability of messages consumed from a non-durable subscriber.

Q: Why can't I run the JMS API sample programs using the J2EE 1.2.x SDK?
A: The J2EE 1.2.x Reference Implementation did not include an implementation of the JMS API. It included only the interfaces. The JMS Reference Implementation is available as part of J2EE 1.3 and you can run the sample programs with the J2EE 1.3 server. See the JMS Tutorial for information on running simple JMS client programs with the J2EE 1.3 server.

Please see Java Message Service API Licensees for a list of

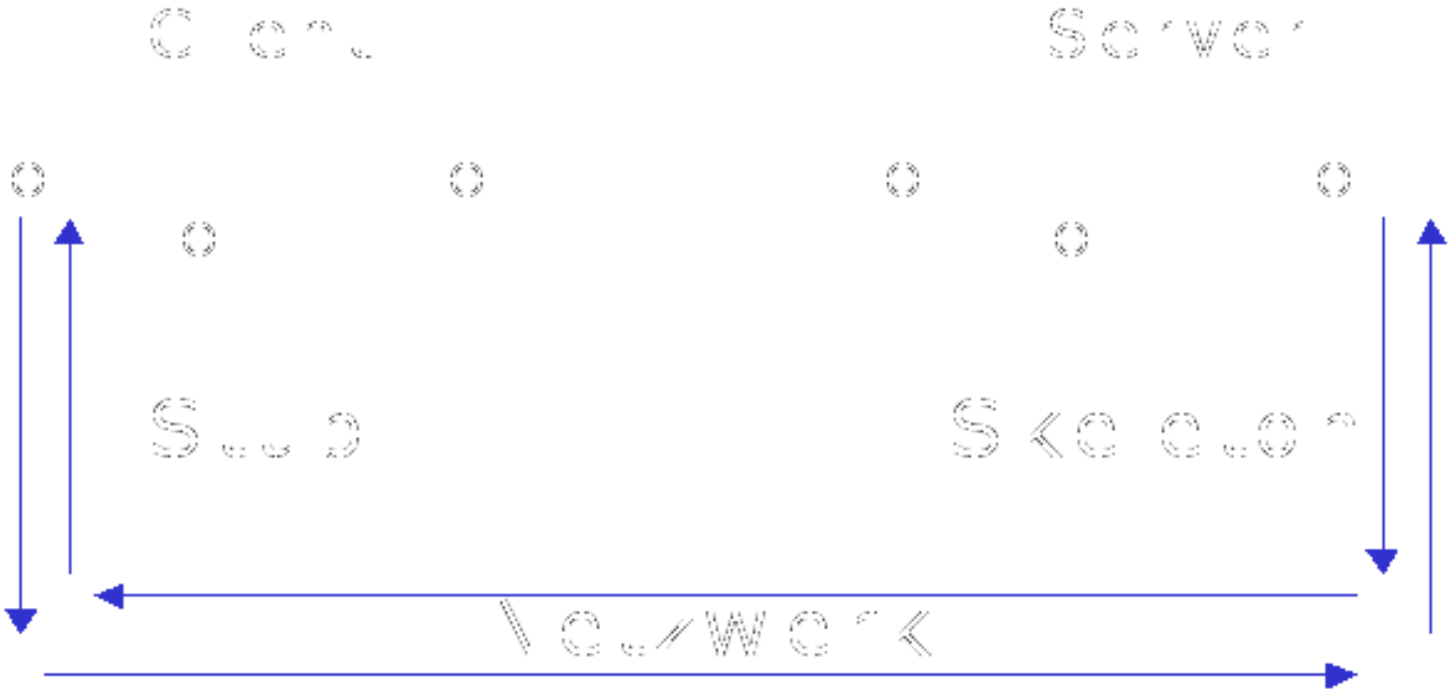
vendors that provide implementations of the JMS API. Many of these vendors have demonstration copies of their JMS API implementations that you can use to run the JMS API samples that you downloaded from the website.

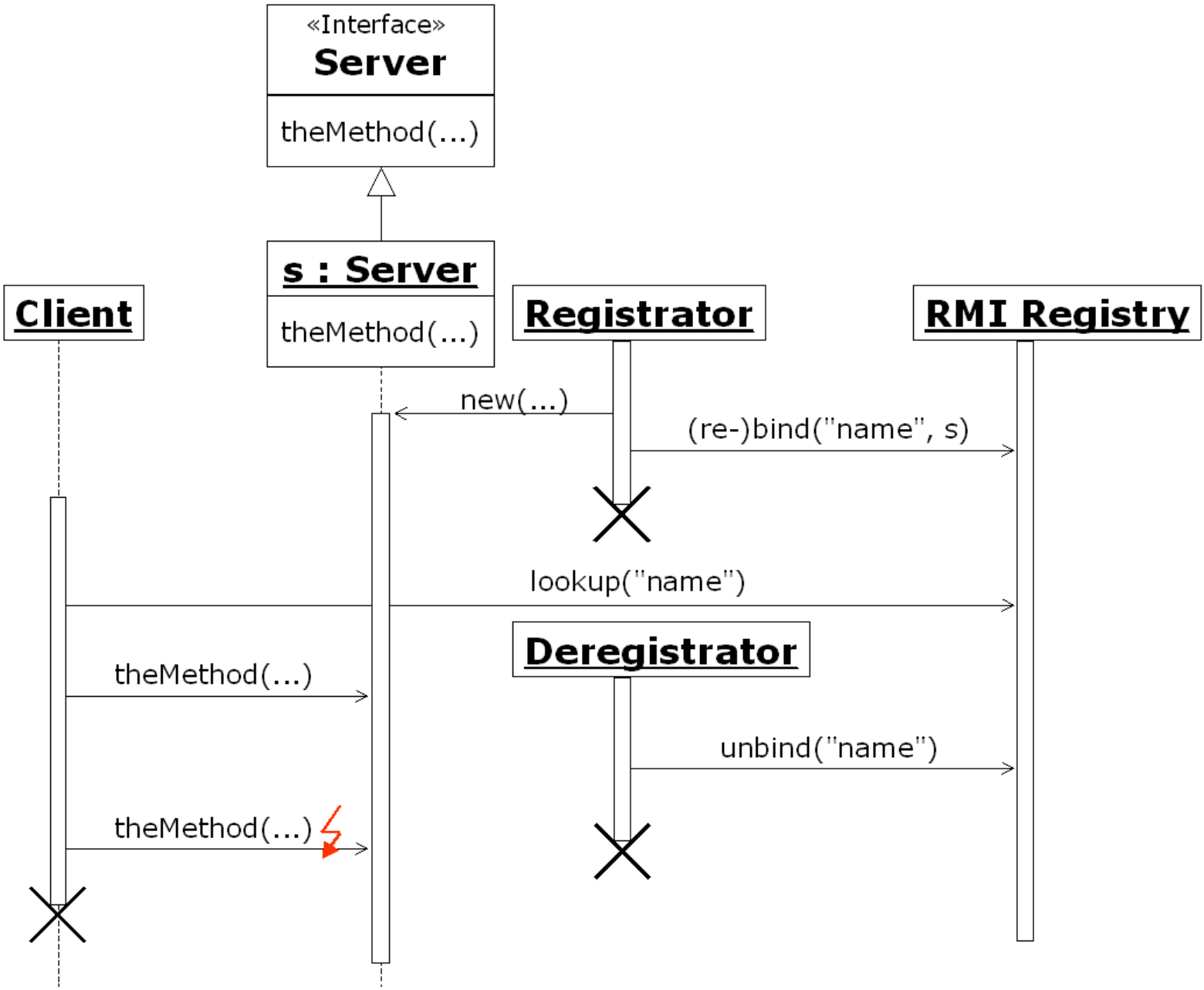
Q: According to the specification, "Getting a property value for a name which has not been set is handled as if the the property exists with a null value." In the case of `getStringProperty`, does this mean a null value or a null string?
A: Either is permissible. The reason for this ambiguity is that some MOM systems do not have the native concept of a null string, so they convert all null-valued strings at send time to empty strings.

Q: I used the `Connection.setExceptionListener (ExceptionListener)` method to try to handle connection problems, but when I stop the server, no `JMSException` seems to be triggered; the `onException` method is not called. Is this a bug?
A: The JMS specification does not specify exactly what exceptions are delivered to an `ExceptionListener` or when they are delivered, so providers vary in how they handle connection problems. Check with your JMS provider if you have questions.

Q: According to Section 3.4 of the JMS Specification, a client can effectively set only three of the message header fields. All the others are set either by the JMS provider or by the `send` or `publish` method. Why do all the fields have a setter method, when any client setting will be overridden when the message is sent or when the JMS provider delivers it? Wouldn't it make sense to allow only getter methods for most of the fields?
A: The specification requires setter methods mainly for consistency, so that each message header field has both a getter and setter method. Another reason is to allow a receiving client to change a field after a message is received but before it is passed to some other part of an application for processing. Although most setter methods are used very rarely, they are provided to give programmers flexibility.

Q: Can two different JMS services talk to each other? For instance, if A and B are two different JMS providers, can Provider A send messages directly to Provider B? If not, then can a subscriber to Provider A act as a publisher to Provider B? NEW
A: The answers are no to the first question and yes to the second. The JMS specification does not require that one JMS provider be able to send messages directly to another provider. However, the specification does require that a JMS client must be able to accept a message created by a different JMS provider, so a message received by a subscriber to Provider A can then be published to Provider B. One caveat is that the publisher to Provider B is not required to handle a `JMSReplyTo` header that refers to a destination that is specific to Provider A.


Client
Server
Stub
Skeleton
Network


«Interface»
Server
theMethod(...)
s : Server
theMethod(...)
Client
Registrator
RMI Registry
new(...)
(re-)bind("name", s)
lookup("name")
Deregistrator
theMethod(...)
unbind("name")
theMethod(...)

---

java.rmi

Class Naming

`java.lang.Object`
  └ **`java.rmi.Naming`**

---

public final class **Naming**
extends Object

The `Naming` class provides methods for storing and obtaining references to remote objects in a remote object registry. Each method of the `Naming` class takes as one of its arguments a name that is a `java.lang.String` in URL format (without the scheme component) of the form:

```
//host:port/name
```

where `host` is the host (remote or local) where the registry is located, `port` is the port number on which the registry accepts calls, and where `name` is a simple string uninterpreted by the registry. Both `host` and `port` are optional. If `host` is omitted, the host defaults to the local host. If `port` is omitted, then the port defaults to 1099, the "well-known" port that RMI's registry, `rmiregistry`, uses.

Binding a name for a remote object is associating or registering a name for a remote object that can be used at a later time to look up that remote object. A remote object can be associated with a name using the `Naming` class's `bind` or `rebind` methods.

Once a remote object is registered (bound) with the RMI registry on the local host, callers on a remote (or local) host can lookup the remote object by name, obtain its reference, and then invoke remote methods on the object. A registry may be shared by all servers running on a host or an individual server process may create and use its own registry if desired (see `java.rmi.registry.LocateRegistry.createRegistry` method for details).

Since:

JDK1.1

See Also:

`Registry`, `LocateRegistry`, `LocateRegistry.createRegistry(int)`

Method Summary

static void	**bind**(String name, Remote obj) Binds the specified name to a remote object.
static String[]	**list**(String name) Returns an array of the names bound in the registry.
static Remote	**lookup**(String name) Returns a reference, a stub, for the remote object associated with the specified name.
static void	**rebind**(String name, Remote obj) Rebinds the specified name to a new remote object.
static void	**unbind**(String name) Destroys the binding for the specified name that is associated with a remote object.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

lookup

```
public static Remote lookup(String name)
                     throws NotBoundException,
                            MalformedURLException,
                            RemoteException
```

Returns a reference, a stub, for the remote object associated with the specified name.

Parameters:

name - a name in URL format (without the scheme component)

Returns:

a reference for a remote object

Throws:

NotBoundException - if name is not currently bound

RemoteException - if registry could not be contacted
AccessException - if this operation is not permitted
MalformedURLException - if the name is not an appropriately formatted URL

Since:

JDK1.1

---

bind

```
public static void bind(String name,
                        Remote obj)
                 throws AlreadyBoundException,
                        MalformedURLException,
                        RemoteException
```

Binds the specified `name` to a remote object.

Parameters:

`name` - a name in URL format (without the scheme component)
`obj` - a reference for the remote object (usually a stub)

Throws:

AlreadyBoundException - if name is already bound
MalformedURLException - if the name is not an appropriately formatted URL
RemoteException - if registry could not be contacted
AccessException - if this operation is not permitted (if originating from a non-local host, for example)

Since:

JDK1.1

---

unbind

```
public static void unbind(String name)
                  throws RemoteException,
                         NotBoundException,
                         MalformedURLException
```

Destroys the binding for the specified name that is associated with a remote object.

Parameters:

`name` - a name in URL format (without the scheme component)

Throws:

NotBoundException - if name is not currently bound

MalformedURLException - if the name is not an appropriately formatted URL

RemoteException - if registry could not be contacted

AccessException - if this operation is not permitted (if originating from a non-local host, for example)

Since:

JDK1.1

---

rebind

```
public static void rebind(String name,
                          Remote obj)
                   throws RemoteException,
                          MalformedURLException
```

Rebinds the specified name to a new remote object. Any existing binding for the name is replaced.

Parameters:

name - a name in URL format (without the scheme component)

obj - new remote object to associate with the name

Throws:

MalformedURLException - if the name is not an appropriately formatted URL

RemoteException - if registry could not be contacted

AccessException - if this operation is not permitted (if originating from a non-local host, for example)

Since:

JDK1.1

---

list

```
public static String[] list(String name)
                    throws RemoteException,
                           MalformedURLException
```

Returns an array of the names bound in the registry. The names are URL-formatted (without the scheme component) strings. The array contains a snapshot of the names present in the registry at the time of the call.

Parameters:

`name` - a registry name in URL format (without the scheme component)

Returns:

an array of names (in the appropriate format) bound in the registry

Throws:

`MalformedURLException` - if the name is not an appropriately formatted URL

`RemoteException` - if registry could not be contacted.

Since:

JDK1.1

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloInterface extends Remote {
	public String sayHello() throws RemoteException;
}
```

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloServer
	extends UnicastRemoteObject
	implements HelloInterface {

	protected HelloServer() throws RemoteException {
		super();
		try {
			Naming.rebind("rmi://localhost:1099/HelloService", this);
		} catch (RemoteException re) {
			re.printStackTrace();
		} catch (MalformedURLException mue) {
			mue.printStackTrace();
		}
	}

	public static void main(String argv[]) throws RemoteException {
		new HelloServer();
	}

	public String sayHello() throws RemoteException {
		return (new String("Hello world!"));
	}
}
```

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class HelloClient {
	public static void main(String[] args) {
		HelloInterface obj = null;
		try {
			obj = (HelloInterface) Naming.lookup("rmi://53.16.71.55:1099/HelloService");
		} catch (MalformedURLException mue) {
			mue.printStackTrace();
		} catch (RemoteException re) {
			System.err.println("Lookup error");
			re.printStackTrace();
		} catch (NotBoundException nbe) {
			System.out.println("Object not bount to registry");
			nbe.printStackTrace();
		}
		System.out.println("retrieved object's oid=" + obj);
		try {
			System.out.println(obj.sayHello());
		} catch (RemoteException re) {
			System.err.println("Error while invoking sayHello");
			re.printStackTrace();
		}
	}
}
```

java.rmi

Class RemoteException

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.io.IOException
              └ java.rmi.RemoteException
```

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AccessException, ActivateFailedException, ConnectException, ConnectIOException, ExportException, InvalidTransactionException, MarshalException, NoSuchObjectException, ServerError, ServerException, ServerRuntimeException, SkeletonMismatchException, SkeletonNotFoundException, StubNotFoundException, TransactionRequiredException, TransactionRolledbackException, UnexpectedException, UnknownHostException, UnmarshalException

---

public class **RemoteException**
extends IOException

A `RemoteException` is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call. Each method of a remote interface, an interface that extends `java.rmi.Remote`, must list `RemoteException` in its throws clause.

As of release 1.4, this exception has been retrofitted to conform to the general purpose exception-chaining mechanism. The "wrapped remote exception" that may be provided at construction time and accessed via the public `detail` field is now known as the *cause*, and may be accessed via the `Throwable.getCause()` method, as well as the aforementioned "legacy field."

Since:

JDK1.1

See Also:

Serialized Form

Field Summary

Throwable	**detail** Nested Exception to hold wrapped remote exception.

Constructor Summary

RemoteException() Constructs a RemoteException with no specified detail message.
RemoteException(String s) Constructs a RemoteException with the specified detail message.
RemoteException(String s, Throwable ex) Constructs a RemoteException with the specified detail message and nested exception.

Method Summary

Throwable	**getCause**() Returns the wrapped remote exception (the *cause*).
String	**getMessage**() Returns the detail message, including the message from the nested exception if there is one.

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

detail

```
public Throwable detail
```

Nested Exception to hold wrapped remote exception.

This field predates the general-purpose exception chaining facility. The `Throwable.getCause()` method is now the preferred means of obtaining this information.

Constructor Detail

RemoteException

```
public RemoteException()
```

Constructs a `RemoteException` with no specified detail message.

RemoteException

```
public RemoteException(String s)
```

Constructs a `RemoteException` with the specified detail message.

Parameters:

`s` - the detail message

RemoteException

```
public RemoteException(String s,
                       Throwable ex)
```

Constructs a `RemoteException` with the specified detail message and nested exception.

Parameters:

`s` - the detail message
`ex` - the nested exception

Method Detail

getMessage

```
public String getMessage()
```

Returns the detail message, including the message from the nested exception if there is one.

Overrides:
`getMessage` in class `Throwable`

Returns:
the detail message, including nested exception message if any

---

getCause

```
public Throwable getCause()
```

Returns the wrapped remote exception (the *cause*).

Overrides:
`getCause` in class `Throwable`

Returns:
the wrapped remote exception, which may be `null`.

Since:
1.4

---

java.rmi.server

Class UnicastRemoteObject

```
java.lang.Object
  └ java.rmi.server.RemoteObject
      └ java.rmi.server.RemoteServer
          └ java.rmi.server.UnicastRemoteObject
```

All Implemented Interfaces:

Remote, Serializable

Direct Known Subclasses:

ActivationGroup

---

public class **UnicastRemoteObject**
extends RemoteServer

The UnicastRemoteObject class defines a non-replicated remote object whose references are valid only while the server process is alive. The UnicastRemoteObject class provides support for point-to-point active object references (invocations, parameters, and results) using TCP streams.

Objects that require remote behavior should extend RemoteObject, typically via UnicastRemoteObject. If UnicastRemoteObject is not extended, the implementation class must then assume the responsibility for the correct semantics of the hashCode, equals, and toString methods inherited from the Object class, so that they behave appropriately for remote objects.

Since:

JDK1.1

See Also:

RemoteServer, RemoteObject, Serialized Form

Field Summary

Fields inherited from class java.rmi.server.RemoteObject
ref

Constructor Summary

protected	**UnicastRemoteObject**() Creates and exports a new UnicastRemoteObject object using an anonymous port.
protected	**UnicastRemoteObject**(int port) Creates and exports a new UnicastRemoteObject object using the particular supplied port.
protected	**UnicastRemoteObject**(int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf) Creates and exports a new UnicastRemoteObject object using the particular supplied port and socket factories.

Method Summary

Object	**clone**() Returns a clone of the remote object that is distinct from the original.
static RemoteStub	**exportObject**(Remote obj) Exports the remote object to make it available to receive incoming calls using an anonymous port.
static Remote	**exportObject**(Remote obj, int port) Exports the remote object to make it available to receive incoming calls, using the particular supplied port.
static Remote	**exportObject**(Remote obj, int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf) Exports the remote object to make it available to receive incoming calls, using a transport specified by the given socket factory.
static boolean	**unexportObject**(Remote obj, boolean force) Removes the remote object, obj, from the RMI runtime.

Methods inherited from class java.rmi.server.RemoteServer
getClientHost, getLog, setLog

Methods inherited from class java.rmi.server.RemoteObject
equals, getRef, hashCode, toString, toStub

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

UnicastRemoteObject

```
protected UnicastRemoteObject()
                        throws RemoteException
```

Creates and exports a new UnicastRemoteObject object using an anonymous port.

Throws:

RemoteException - if failed to export object

Since:

JDK1.1

UnicastRemoteObject

```
protected UnicastRemoteObject(int port)
                        throws RemoteException
```

Creates and exports a new UnicastRemoteObject object using the particular supplied port.

Parameters:

`port` - the port number on which the remote object receives calls (if `port` is zero, an anonymous port is chosen)

Throws:

RemoteException - if failed to export object

Since:

1.2

UnicastRemoteObject

```
protected UnicastRemoteObject(int port,
                              RMIClientSocketFactory csf,
                              RMIServerSocketFactory ssf)
                       throws RemoteException
```

Creates and exports a new UnicastRemoteObject object using the particular supplied port and socket factories.

Parameters:

`port` - the port number on which the remote object receives calls (if `port` is zero, an anonymous port is chosen)
`csf` - the client-side socket factory for making calls to the remote object
`ssf` - the server-side socket factory for receiving remote calls

Throws:

`RemoteException` - if failed to export object

Since:

1.2

Method Detail

clone

```
public Object clone()
            throws CloneNotSupportedException
```

Returns a clone of the remote object that is distinct from the original.

Overrides:

`clone` in class `Object`

Returns:

the new remote object

Throws:

`CloneNotSupportedException` - if clone failed due to a RemoteException.

Since:

JDK1.1

See Also:

`Cloneable`

---

exportObject

```
public static RemoteStub exportObject(Remote obj)
                               throws RemoteException
```

Exports the remote object to make it available to receive incoming calls using an anonymous port.

Parameters:
`obj` - the remote object to be exported
Returns:
remote object stub
Throws:
`RemoteException` - if export fails
Since:
JDK1.1

---

exportObject

```
public static Remote exportObject(Remote obj,
                                  int port)
                           throws RemoteException
```

Exports the remote object to make it available to receive incoming calls, using the particular supplied port.

Parameters:
`obj` - the remote object to be exported
`port` - the port to export the object on
Returns:
remote object stub
Throws:
`RemoteException` - if export fails
Since:
1.2

---

exportObject

```
public static Remote exportObject(Remote obj,
```

```
                                int port,
                                RMIClientSocketFactory csf,
                                RMIServerSocketFactory ssf)
                         throws RemoteException
```

Exports the remote object to make it available to receive incoming calls, using a transport specified by the given socket factory.

Parameters:

`obj` - the remote object to be exported
`port` - the port to export the object on
`csf` - the client-side socket factory for making calls to the remote object
`ssf` - the server-side socket factory for receiving remote calls

Returns:

remote object stub

Throws:

`RemoteException` - if export fails

Since:

1.2

---

unexportObject

```
public static boolean unexportObject(Remote obj,
                                     boolean force)
                              throws NoSuchObjectException
```

Removes the remote object, obj, from the RMI runtime. If successful, the object can no longer accept incoming RMI calls. If the force parameter is true, the object is forcibly unexported even if there are pending calls to the remote object or the remote object still has calls in progress. If the force parameter is false, the object is only unexported if there are no pending or in progress calls to the object.

Parameters:

`obj` - the remote object to be unexported
`force` - if true, unexports the object even if there are pending or in-progress calls; if false, only unexports the object if there are no pending or in-progress calls

Returns:

true if operation is successful, false otherwise

Throws:

`NoSuchObjectException` - if the remote object is not currently exported

Since:

1.2

Submit a bug or feature

For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Resource Page

NEW 2003-07-19 Modula-3 Mailing Lists

There are now three Modula-3 mailing lists, concerned with announcements, development, and CVS log messages of CM3 and PM3. You can browse and search the mailing list archives at `http://www.elegosoft.com/Zope/m3/m3announce` `http://www.elegosoft.com/Zope/m3/m3devel` `http://www.elegosoft.com/Zope/m3/m3commit`

NEW 2003-07-17 CM3 5.2.6

CM3 5.2.6 has been released. This release adds the PPC_LINUX target platform, a new code generator based on gcc 3.2.3, and several bug fixes and small extensions. Have a look at the RELNOTES_5_2_6 for details. CM3 5.2.6 also includes the errno fixes that make it work together with glibc 2.3.2 (see below).

NEW 2003-04-16 Ezm3 1.1

Ezm3 1.1 is now available! This new release includes an updated code generator (based on gcc 3.2.1) which allows support of newer platforms. This release adds support for FreeBSD/sparc64. It also works correctly with Linux systems based on glibc 2.3.2, including Red Hat 9, SuSE 8.2, and CRUX 1.1.

More (old :-) News

Implementations

PM3
SRC M3
CM3
EZM3

Resources

Language Definition
FAQ
Bibliography
Concise Bibliography
Threads Newsletter
Modula-3 Mailing Lists

Hot Links

Modula-3 Home Page
SPIN Home Page
Critical Mass
PM3 Home Page at Elego
Persistent Modula-3

Misc

Modula-3 News Group
Deja News
Obliq
CVSup
CVSup in the FreeBSD HandBook
Quotes about M3
Banners
Binary IO
Latest UDP code (historical), better use the package download from elego.

Books

Modula-3
Systems Programming with Modula-3
Algorithms in Modula-3
Programming in Modula-3

About M3.ORG

who we are, why we're here, and how to become a member

News & Announcements

keep up with the latest events concerning Modula-3

About Modula-3

history and general information about Modula-3

Education Material

information that assists in teaching Modula-3

Related Sites

other sites on the web with information relating to Modula-3

Download

grab the Modula-3 compiler and libraries for a variety of platforms

---

You are visitor number 00077719 since June 22th, 1998.
Last Updated: March 14, 2002
Webmaster

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ActivatableHelloInterface extends Remote {
	public String sayHello() throws RemoteException;
}
```

```
import java.rmi.MarshalledObject;
import java.rmi.RemoteException;
import java.rmi.activation.Activatable;
import java.rmi.activation.ActivationID;

public class ActivatableHelloServer
	extends Activatable
	implements ActivatableHelloInterface {

	public ActivatableHelloServer(ActivationID id, MarshalledObject data)
		throws RemoteException {
		super(id, 0);
	}

	public String sayHello() throws RemoteException {
		return (new String("Hello world!"));
	}
}
```

```
import java.net.MalformedURLException;
import java.rmi.MarshalledObject;
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.rmi.activation.Activatable;
import java.rmi.activation.ActivationDesc;
import java.rmi.activation.ActivationException;
import java.rmi.activation.ActivationGroup;
import java.rmi.activation.ActivationGroupDesc;
import java.rmi.activation.ActivationGroupID;
import java.rmi.activation.UnknownGroupException;
import java.util.Properties;

public class ActivatableSetup {
	public static void main(String argv[]) throws RemoteException {
		System.setSecurityManager(new RMISecurityManager());
		Properties props = new Properties();
		props.put("java.security.policy", "policy");
		ActivationGroupDesc.CommandEnvironment ace = null;
		ActivationGroupDesc exampleGroup = new ActivationGroupDesc(props, ace);
		ActivationGroupID agi = null;
		try {
			agi = ActivationGroup.getSystem().registerGroup(exampleGroup);
		} catch (RemoteException re) {
			re.printStackTrace();
		} catch (ActivationException ae) {
			ae.printStackTrace();
		}
		String location = "file:/home/mario/RMITest/activation/";
		MarshalledObject data = null;
		ActivationDesc desc =
			new ActivationDesc(agi, "ActivatableHelloServer", location, data);

		ActivatableHelloInterface stub = null;
		try {
			stub = (ActivatableHelloInterface) Activatable.register(desc);
		} catch (UnknownGroupException uge) {
			uge.printStackTrace();
		} catch (RemoteException re) {
			re.printStackTrace();
		} catch (ActivationException ae) {
			ae.printStackTrace();
		}
		try {
```

```
            Naming.rebind("rmi://localhost:1099/HelloObj", stub);
        } catch (RemoteException re) {
            re.printStackTrace();
        } catch (MalformedURLException mue) {
            mue.printStackTrace();
        }

    }
}
```

java.rmi.activation

Class Activatable

```
java.lang.Object
  └ java.rmi.server.RemoteObject
      └ java.rmi.server.RemoteServer
          └ java.rmi.activation.Activatable
```

All Implemented Interfaces:

Remote, Serializable

---

public abstract class **Activatable**
extends RemoteServer

The `Activatable` class provides support for remote objects that require persistent access over time and that can be activated by the system.

Since:

1.2

See Also:

Serialized Form

Field Summary

Fields inherited from class java.rmi.server.RemoteObject
ref

Constructor Summary

protected	**Activatable**(ActivationID id, int port) Constructor used to activate/export the object on a specified port.

protected	**Activatable**(ActivationID id, int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf) Constructor used to activate/export the object on a specified port.
protected	**Activatable**(String location, MarshalledObject data, boolean restart, int port) Constructor used to register and export the object on a specified port (an anonymous port is chosen if port=0) .
protected	**Activatable**(String location, MarshalledObject data, boolean restart, int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf) Constructor used to register and export the object on a specified port (an anonymous port is chosen if port=0) .

Method Summary

static Remote	**exportObject**(Remote obj, ActivationID id, int port) Export the activatable remote object to the RMI runtime to make the object available to receive incoming calls.
static Remote	**exportObject**(Remote obj, ActivationID id, int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf) Export the activatable remote object to the RMI runtime to make the object available to receive incoming calls.
static ActivationID	**exportObject**(Remote obj, String location, MarshalledObject data, boolean restart, int port) This `exportObject` method may be invoked explicitly by an "activatable" object, that does not extend the `Activatable` class, in order to both a) register the object's activation descriptor, constructed from the supplied `location`, and `data`, with the activation system (so the object can be activated), and b) export the remote object, `obj`, on a specific port (if port=0, then an anonymous port is chosen).

static ActivationID	**exportObject**(Remote obj, String location, MarshalledObject data, boolean restart, int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf) This exportObject method may be invoked explicitly by an "activatable" object, that does not extend the Activatable class, in order to both a) register the object's activation descriptor, constructed from the supplied location, and data, with the activation system (so the object can be activated), and b) export the remote object, obj, on a specific port (if port=0, then an anonymous port is chosen).
protected ActivationID	**getID**() Returns the object's activation identifier.
static boolean	**inactive**(ActivationID id) Informs the system that the object with the corresponding activation id is currently inactive.
static Remote	**register**(ActivationDesc desc) Register an object descriptor for an activatable remote object so that is can be activated on demand.
static boolean	**unexportObject**(Remote obj, boolean force) Remove the remote object, obj, from the RMI runtime.
static void	**unregister**(ActivationID id) Revokes previous registration for the activation descriptor associated with id.

Methods inherited from class java.rmi.server.RemoteServer
getClientHost, getLog, setLog

Methods inherited from class java.rmi.server.RemoteObject
equals, getRef, hashCode, toString, toStub

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

Activatable

```
protected Activatable(String location,
                      MarshalledObject data,
                      boolean restart,
                      int port)
               throws ActivationException,
                      RemoteException
```

Constructor used to register and export the object on a specified port (an anonymous port is chosen if port=0) . A concrete subclass of this class must call this constructor to register and export the object during *initial* construction. As a side-effect of activatable object construction, the remote object is both "registered" with the activation system and "exported" (on an anonymous port if port=0) to the RMI runtime so that it is available to accept incoming calls from clients.

Parameters:

`location` - the location for classes for this object
`data` - the object's initialization data
`port` - the port on which the object is exported (an anonymous port is used if port=0)
`restart` - if true, the object is restarted (reactivated) when either the activator is restarted or the object's activation group is restarted after an unexpected crash; if false, the object is only activated on demand. Specifying `restart` to be `true` does not force an initial immediate activation of a newly registered object; initial activation is lazy.

Throws:

`ActivationException` - if object registration fails.
`RemoteException` - if either of the following fails: a) registering the object with the activation system or b) exporting the object to the RMI runtime.

Since:

1.2

Activatable

```
protected Activatable(String location,
                      MarshalledObject data,
                      boolean restart,
                      int port,
                      RMIClientSocketFactory csf,
                      RMIServerSocketFactory ssf)
               throws ActivationException,
                      RemoteException
```

Constructor used to register and export the object on a specified port (an anonymous port is

chosen if port=0) .

A concrete subclass of this class must call this constructor to register and export the object during *initial* construction. As a side-effect of activatable object construction, the remote object is both "registered" with the activation system and "exported" (on an anonymous port if port=0) to the RMI runtime so that it is available to accept incoming calls from clients.

Parameters:

`location` - the location for classes for this object
`data` - the object's initialization data
`restart` - if true, the object is restarted (reactivated) when either the activator is restarted or the object's activation group is restarted after an unexpected crash; if false, the object is only activated on demand. Specifying `restart` to be `true` does not force an initial immediate activation of a newly registered object; initial activation is lazy.
`port` - the port on which the object is exported (an anonymous port is used if port=0)
`csf` - the client-side socket factory for making calls to the remote object
`ssf` - the server-side socket factory for receiving remote calls

Throws:

`ActivationException` - if object registration fails.
`RemoteException` - if either of the following fails: a) registering the object with the activation system or b) exporting the object to the RMI runtime.

Since:

1.2

Activatable

```
protected Activatable(ActivationID id,
                      int port)
               throws RemoteException
```

Constructor used to activate/export the object on a specified port. An "activatable" remote object must have a constructor that takes two arguments:

- the object's activation identifier (`ActivationID`), and
- the object's initialization data (a `MarshalledObject`).

A concrete subclass of this class must call this constructor when it is *activated* via the two parameter constructor described above. As a side-effect of construction, the remote object is "exported" to the RMI runtime (on the specified `port`) and is available to accept incoming calls from clients.

Parameters:

`id` - activation identifier for the object

`port` - the port number on which the object is exported

Throws:

`RemoteException` - if exporting the object to the RMI runtime fails

Since:

1.2

Activatable

```
protected Activatable(ActivationID id,
                      int port,
                      RMIClientSocketFactory csf,
                      RMIServerSocketFactory ssf)
               throws RemoteException
```

Constructor used to activate/export the object on a specified port. An "activatable" remote object must have a constructor that takes two arguments:

- the object's activation identifier (`ActivationID`), and
- the object's initialization data (a `MarshalledObject`).

A concrete subclass of this class must call this constructor when it is *activated* via the two parameter constructor described above. As a side-effect of construction, the remote object is "exported" to the RMI runtime (on the specified `port`) and is available to accept incoming calls from clients.

Parameters:

`id` - activation identifier for the object
`port` - the port number on which the object is exported
`csf` - the client-side socket factory for making calls to the remote object
`ssf` - the server-side socket factory for receiving remote calls

Throws:

`RemoteException` - if exporting the object to the RMI runtime fails

Since:

1.2

Method Detail

getID

```
protected ActivationID getID()
```

Returns the object's activation identifier. The method is protected so that only subclasses can

obtain an object's identifier.

Returns:
the object's activation identifier
Since:
1.2

---

register

```
public static Remote register(ActivationDesc desc)
                       throws UnknownGroupException,
                              ActivationException,
                              RemoteException
```

Register an object descriptor for an activatable remote object so that is can be activated on demand.

Parameters:
`desc` - the object's descriptor
Returns:
the stub for the activatable remote object
Throws:
`UnknownGroupException` - if group id in `desc` is not registered with the activation system
`ActivationException` - if activation system is not running
`RemoteException` - if remote call fails
Since:
1.2

---

inactive

```
public static boolean inactive(ActivationID id)
                        throws UnknownObjectException,
                               ActivationException,
                               RemoteException
```

Informs the system that the object with the corresponding activation `id` is currently inactive. If the object is currently active, the object is "unexported" from the RMI runtime (only if there are no pending or in-progress calls) so the that it can no longer receive incoming calls. This

call informs this VM's ActivationGroup that the object is inactive, that, in turn, informs its ActivationMonitor. If this call completes successfully, a subsequent activate request to the activator will cause the object to reactivate. The operation may still succeed if the object is considered active but has already unexported itself.

Parameters:

`id` - the object's activation identifier

Returns:

true if the operation succeeds (the operation will succeed if the object in currently known to be active and is either already unexported or is currently exported and has no pending/executing calls); false is returned if the object has pending/executing calls in which case it cannot be deactivated

Throws:

`UnknownObjectException` - if object is not known (it may already be inactive)

`ActivationException` - if group is not active

`RemoteException` - if call informing monitor fails

Since:

1.2

---

unregister

```
public static void unregister(ActivationID id)
                       throws UnknownObjectException,
                              ActivationException,
                              RemoteException
```

Revokes previous registration for the activation descriptor associated with `id`. An object can no longer be activated via that `id`.

Parameters:

`id` - the object's activation identifier

Throws:

`UnknownObjectException` - if object (`id`) is unknown

`ActivationException` - if activation system is not running

`RemoteException` - if remote call to activation system fails

Since:

1.2

---

exportObject

```
public static ActivationID exportObject(Remote obj,
                                        String location,
                                        MarshalledObject data,
                                        boolean restart,
                                        int port)
                                 throws ActivationException,
                                        RemoteException
```

This `exportObject` method may be invoked explicitly by an "activatable" object, that does not extend the `Activatable` class, in order to both a) register the object's activation descriptor, constructed from the supplied `location`, and `data`, with the activation system (so the object can be activated), and b) export the remote object, `obj`, on a specific port (if port=0, then an anonymous port is chosen). Once the object is exported, it can receive incoming RMI calls.

This method does not need to be called if `obj` extends `Activatable`, since the first constructor calls this method.

Parameters:

`obj` - the object being exported
`location` - the object's code location
`data` - the object's bootstrapping data
`restart` - if true, the object is restarted (reactivated) when either the activator is restarted or the object's activation group is restarted after an unexpected crash; if false, the object is only activated on demand. Specifying `restart` to be `true` does not force an initial immediate activation of a newly registered object; initial activation is lazy.
`port` - the port on which the object is exported (an anonymous port is used if port=0)

Returns:

the activation identifier obtained from registering the descriptor, `desc`, with the activation system the wrong group

Throws:

`ActivationException` - if activation group is not active
`RemoteException` - if object registration or export fails

Since:

1.2

---

exportObject

```
public static ActivationID exportObject(Remote obj,
                                        String location,
                                        MarshalledObject data,
```

```
                                boolean restart,
                                int port,
                                RMIClientSocketFactory csf,
                                RMIServerSocketFactory ssf)
                         throws ActivationException,
                                RemoteException
```

This `exportObject` method may be invoked explicitly by an "activatable" object, that does not extend the `Activatable` class, in order to both a) register the object's activation descriptor, constructed from the supplied `location`, and `data`, with the activation system (so the object can be activated), and b) export the remote object, `obj`, on a specific port (if port=0, then an anonymous port is chosen). Once the object is exported, it can receive incoming RMI calls.

This method does not need to be called if `obj` extends `Activatable`, since the first constructor calls this method.

Parameters:

`obj` - the object being exported
`location` - the object's code location
`data` - the object's bootstrapping data
`restart` - if true, the object is restarted (reactivated) when either the activator is restarted or the object's activation group is restarted after an unexpected crash; if false, the object is only activated on demand. Specifying `restart` to be `true` does not force an initial immediate activation of a newly registered object; initial activation is lazy.
`port` - the port on which the object is exported (an anonymous port is used if port=0)
`csf` - the client-side socket factory for making calls to the remote object
`ssf` - the server-side socket factory for receiving remote calls

Returns:

the activation identifier obtained from registering the descriptor, `desc`, with the activation system the wrong group

Throws:

`ActivationException` - if activation group is not active
`RemoteException` - if object registration or export fails

Since:

1.2

---

exportObject

```
public static Remote exportObject(Remote obj,
                                  ActivationID id,
```

```
                                int port)
                         throws RemoteException
```

Export the activatable remote object to the RMI runtime to make the object available to receive incoming calls. The object is exported on an anonymous port, if `port` is zero.

During activation, this `exportObject` method should be invoked explicitly by an "activatable" object, that does not extend the `Activatable` class. There is no need for objects that do extend the `Activatable` class to invoke this method directly; this method is called by the second constructor above (which a subclass should invoke from its special activation constructor).

Parameters:

`obj` - the remote object implementation
`id` - the object's activation identifier
`port` - the port on which the object is exported (an anonymous port is used if port=0)

Returns:

the stub for the activatable remote object

Throws:

`RemoteException` - if object export fails

Since:

1.2

---

exportObject

```
public static Remote exportObject(Remote obj,
                                  ActivationID id,
                                  int port,
                                  RMIClientSocketFactory csf,
                                  RMIServerSocketFactory ssf)
                           throws RemoteException
```

Export the activatable remote object to the RMI runtime to make the object available to receive incoming calls. The object is exported on an anonymous port, if `port` is zero.

During activation, this `exportObject` method should be invoked explicitly by an "activatable" object, that does not extend the `Activatable` class. There is no need for objects that do extend the `Activatable` class to invoke this method directly; this method is called by the second constructor above (which a subclass should invoke from its special activation constructor).

Parameters:

`obj` - the remote object implementation
`id` - the object's activation identifier
`port` - the port on which the object is exported (an anonymous port is used if port=0)
`csf` - the client-side socket factory for making calls to the remote object
`ssf` - the server-side socket factory for receiving remote calls

Returns:

the stub for the activatable remote object

Throws:

`RemoteException` - if object export fails

Since:

1.2

---

unexportObject

```
public static boolean unexportObject(Remote obj,
                                     boolean force)
                              throws NoSuchObjectException
```

Remove the remote object, obj, from the RMI runtime. If successful, the object can no longer accept incoming RMI calls. If the force parameter is true, the object is forcibly unexported even if there are pending calls to the remote object or the remote object still has calls in progress. If the force parameter is false, the object is only unexported if there are no pending or in progress calls to the object.

Parameters:

`obj` - the remote object to be unexported
`force` - if true, unexports the object even if there are pending or in-progress calls; if false, only unexports the object if there are no pending or in-progress calls

Returns:

true if operation is successful, false otherwise

Throws:

`NoSuchObjectException` - if the remote object is not currently exported

Since:

1.2

---

documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class ActivatableHelloClient {
	public static void main(String[] args) {
		ActivatableHelloInterface obj = null;
		try {
			obj = (ActivatableHelloInterface) Naming.lookup("rmi://53.16.71.55:1099/HelloObj");
		} catch (MalformedURLException mue) {
			mue.printStackTrace();
		} catch (RemoteException re) {
			System.err.println("Lookup error");
			re.printStackTrace();
		} catch (NotBoundException nbe) {
			System.out.println("Object not bount to registry");
			nbe.printStackTrace();
		}
		System.out.println("retrieved object's oid=" + obj);
		try {
			System.out.println(obj.sayHello());
		} catch (RemoteException re) {
			System.err.println("Error while invoking sayHello");
			re.printStackTrace();
		}
	}
}
```

java.rmi

Class RMISecurityManager

```
java.lang.Object
  └ java.lang.SecurityManager
      └ java.rmi.RMISecurityManager
```

public class **RMISecurityManager**
extends SecurityManager

`RMISecurityManager` provides an example security manager for use by RMI applications that use downloaded code. RMI's class loader will not download any classes from remote locations if no security manager has been set. `RMISecurityManager` does not apply to applets, which run under the protection of their browser's security manager.

To use the `RMISecurityManager` in your application , add the following statement to your code (it needs to be executed before RMI can download code from remote hosts, so it most likely needs to appear in the `main` of your application):

```
System.setSecurityManager(new RMISecurityManager());
```

Since:

JDK1.1

Field Summary

Fields inherited from class java.lang.SecurityManager
`inCheck`

Constructor Summary

RMISecurityManager() Constructs a new RMISecurityManager.

Methods inherited from class java.lang.SecurityManager
checkAccept, checkAccess, checkAccess, checkAwtEventQueueAccess, checkConnect, checkConnect, checkCreateClassLoader, checkDelete, checkExec, checkExit, checkLink, checkListen, checkMemberAccess, checkMulticast, checkMulticast, checkPackageAccess, checkPackageDefinition, checkPermission, checkPermission, checkPrintJobAccess, checkPropertiesAccess, checkPropertyAccess, checkRead, checkRead, checkRead, checkSecurityAccess, checkSetFactory, checkSystemClipboardAccess, checkTopLevelWindow, checkWrite, checkWrite, classDepth, classLoaderDepth, currentClassLoader, currentLoadedClass, getClassContext, getInCheck, getSecurityContext, getThreadGroup, inClass, inClassLoader

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

RMISecurityManager

```
public RMISecurityManager()
```

Constructs a new RMISecurityManager.

Since:

JDK1.1

For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

java.rmi.activation

Class ActivationGroupDesc

java.lang.Object
  └ **java.rmi.activation.ActivationGroupDesc**

All Implemented Interfaces:

Serializable

---

public final class **ActivationGroupDesc**
extends Object
implements Serializable

An activation group descriptor contains the information necessary to create/recreate an activation group in which to activate objects. Such a descriptor contains:

- the group's class name,
- the group's code location (the location of the group's class), and
- a "marshalled" object that can contain group specific initialization data.

The group's class must be a concrete subclass of `ActivationGroup`. A subclass of `ActivationGroup` is created/recreated via the `ActivationGroup.createGroup` static method that invokes a special constructor that takes two arguments:

- the group's `ActivationGroupID`, and
- the group's initialization data (in a `java.rmi.MarshalledObject`)

Since:

1.2

See Also:

`ActivationGroup`, `ActivationGroupID`, Serialized Form

Nested Class Summary

static class	**ActivationGroupDesc.CommandEnvironment** Startup options for ActivationGroup implementations.

Constructor Summary

ActivationGroupDesc(Properties overrides, ActivationGroupDesc.CommandEnvironment cmd)

Constructs a group descriptor that uses the system defaults for group implementation and code location.

ActivationGroupDesc(String className, String location, MarshalledObject data, Properties overrides, ActivationGroupDesc.CommandEnvironment cmd)

Specifies an alternate group implementation and execution environment to be used for the group.

Method Summary

boolean	**equals**(Object obj) Compares two activation group descriptors for content equality.
String	**getClassName**() Returns the group's class name (possibly `null`).
ActivationGroupDesc.CommandEnvironment	**getCommandEnvironment**() Returns the group's command-environment control object.
MarshalledObject	**getData**() Returns the group's initialization data.
String	**getLocation**() Returns the group's code location.
Properties	**getPropertyOverrides**() Returns the group's property-override list.
int	**hashCode**() Produce identical numbers for similar `ActivationGroupDesc`s.

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ActivationGroupDesc

```
public ActivationGroupDesc(Properties overrides,
                           ActivationGroupDesc.
CommandEnvironment cmd)
```

Constructs a group descriptor that uses the system defaults for group implementation and code location. Properties specify Java environment overrides (which will override system properties in the group implementation's VM). The command environment can control the exact command/options used in starting the child VM, or can be `null` to accept rmid's default.

This constructor will create an `ActivationGroupDesc` with a `null` group class name, which indicates the system's default `ActivationGroup` implementation.

Parameters:

`overrides` - the set of properties to set when the group is recreated.
`cmd` - the controlling options for executing the VM in another process (or `null`).

Since:

1.2

ActivationGroupDesc

```
public ActivationGroupDesc(String className,
                           String location,
                           MarshalledObject data,
                           Properties overrides,
                           ActivationGroupDesc.
CommandEnvironment cmd)
```

Specifies an alternate group implementation and execution environment to be used for the group.

Parameters:

`className` - the group's package qualified class name or `null`. A `null` group class name indicates the system's default `ActivationGroup` implementation.
`location` - the location from where to load the group's class
`data` - the group's initialization data contained in marshalled form (could contain properties, for example)

`overrides` - a properties map which will override those set by default in the subprocess environment (will be translated into `-D` options), or `null`.
`cmd` - the controlling options for executing the VM in another process (or `null`).

Since:

1.2

Method Detail

getClassName

```
public String getClassName()
```

Returns the group's class name (possibly `null`). A `null` group class name indicates the system's default `ActivationGroup` implementation.

Returns:

the group's class name

Since:

1.2

---

getLocation

```
public String getLocation()
```

Returns the group's code location.

Returns:

the group's code location

Since:

1.2

---

getData

```
public MarshalledObject getData()
```

Returns the group's initialization data.

Returns:

the group's initialization data

Since:

1.2

---

getPropertyOverrides

`public Properties getPropertyOverrides()`

Returns the group's property-override list.

Returns:

the property-override list, or `null`

Since:

1.2

---

getCommandEnvironment

`public ActivationGroupDesc.CommandEnvironment getCommandEnvironment()`

Returns the group's command-environment control object.

Returns:

the command-environment object, or `null`

Since:

1.2

---

equals

`public boolean equals(Object obj)`

Compares two activation group descriptors for content equality.

Overrides:

`equals` in class `Object`

Parameters:

`obj` - the Object to compare with

Returns:

true if these Objects are equal; false otherwise.

Since:

1.2

See Also:

`Hashtable`

---

hashCode

```
public int hashCode()
```

Produce identical numbers for similar `ActivationGroupDesc`s.

Overrides:

`hashCode` in class `Object`

Returns:

an integer

See Also:

`Hashtable`

---

java.rmi.activation

Interface ActivationSystem

All Superinterfaces:

Remote

public interface **ActivationSystem**
extends Remote

The `ActivationSystem` provides a means for registering groups and "activatable" objects to be activated within those groups. The `ActivationSystem` works closely with the `Activator`, which activates objects registered via the `ActivationSystem`, and the `ActivationMonitor`, which obtains information about active and inactive objects, and inactive groups.

Since:

1.2

See Also:

`Activator`, `ActivationMonitor`

Field Summary

`static int`	**`SYSTEM_PORT`** The port to lookup the activation system.

Method Summary

`ActivationMonitor`	**`activeGroup`**`(ActivationGroupID id, ActivationInstantiator group, long incarnation)` Callback to inform activation system that group is now active.
`ActivationDesc`	**`getActivationDesc`**`(ActivationID id)` Returns the activation descriptor, for the object with the activation identifier, `id`.
`ActivationGroupDesc`	**`getActivationGroupDesc`**`(ActivationGroupID id)` Returns the activation group descriptor, for the group with the activation group identifier, `id`.

ActivationGroupID	**registerGroup**(ActivationGroupDesc desc) Register the activation group.
ActivationID	**registerObject**(ActivationDesc desc) The registerObject method is used to register an activation descriptor, desc, and obtain an activation identifier for a activatable remote object.
ActivationDesc	**setActivationDesc**(ActivationID id, ActivationDesc desc) Set the activation descriptor, desc for the object with the activation identifier, id.
ActivationGroupDesc	**setActivationGroupDesc**(ActivationGroupID id, ActivationGroupDesc desc) Set the activation group descriptor, desc for the object with the activation group identifier, id.
void	**shutdown**() Shutdown the activation system.
void	**unregisterGroup**(ActivationGroupID id) Remove the activation group.
void	**unregisterObject**(ActivationID id) Remove the activation id and associated descriptor previously registered with the ActivationSystem; the object can no longer be activated via the object's activation id.

Field Detail

SYSTEM_PORT

```
public static final int SYSTEM_PORT
```

The port to lookup the activation system.

See Also:

Constant Field Values

Method Detail

registerObject

```
public ActivationID registerObject(ActivationDesc desc)
                            throws ActivationException,
                                   UnknownGroupException,
```

RemoteException

The `registerObject` method is used to register an activation descriptor, `desc`, and obtain an activation identifier for a activatable remote object. The `ActivationSystem` creates an `ActivationID` (a activation identifier) for the object specified by the descriptor, `desc`, and records, in stable storage, the activation descriptor and its associated identifier for later use. When the `Activator` receives an `activate` request for a specific identifier, it looks up the activation descriptor (registered previously) for the specified identifier and uses that information to activate the object.

Parameters:

`desc` - the object's activation descriptor

Returns:

the activation id that can be used to activate the object

Throws:

`ActivationException` - if registration fails (e.g., database update failure, etc).

`UnknownGroupException` - if group referred to in `desc` is not registered with this system

`RemoteException` - if remote call fails

Since:

1.2

---

unregisterObject

```
public void unregisterObject(ActivationID id)
                      throws ActivationException,
                             UnknownObjectException,
                             RemoteException
```

Remove the activation id and associated descriptor previously registered with the `ActivationSystem`; the object can no longer be activated via the object's activation id.

Parameters:

`id` - the object's activation id (from previous registration)

Throws:

`ActivationException` - if unregister fails (e.g., database update failure, etc).

`UnknownObjectException` - if object is unknown (not registered)

`RemoteException` - if remote call fails

Since:

1.2

---

registerGroup

```
public ActivationGroupID registerGroup(ActivationGroupDesc desc)
```

```
                                    throws ActivationException,
                                           RemoteException
```

Register the activation group. An activation group must be registered with the `ActivationSystem` before objects can be registered within that group.

Parameters:
`desc` - the group's descriptor
Returns:
an identifier for the group
Throws:
`ActivationException` - if group registration fails
`RemoteException` - if remote call fails
Since:
1.2

---

activeGroup

```
public ActivationMonitor activeGroup(ActivationGroupID id,
                                     ActivationInstantiator group,
                                     long incarnation)
                              throws UnknownGroupException,
                                     ActivationException,
                                     RemoteException
```

Callback to inform activation system that group is now active. This call is made internally by the `ActivationGroup.createGroup` method to inform the `ActivationSystem` that the group is now active.

Parameters:
`id` - the activation group's identifier
`group` - the group's instantiator
`incarnation` - the group's incarnation number
Returns:
monitor for activation group
Throws:
`UnknownGroupException` - if group is not registered
`ActivationException` - if group is already active
`RemoteException` - if remote call fails
Since:
1.2

---

unregisterGroup

```
public void unregisterGroup(ActivationGroupID id)
                     throws ActivationException,
                            UnknownGroupException,
                            RemoteException
```

Remove the activation group. An activation group makes this call back to inform the activator that the group should be removed (destroyed). If this call completes successfully, objects can no longer be registered or activated within the group. All information of the group and its associated objects is removed from the system.

Parameters:

`id` - the activation group's identifier

Throws:

`ActivationException` - if unregister fails (e.g., database update failure, etc).

`UnknownGroupException` - if group is not registered

`RemoteException` - if remote call fails

Since:

1.2

---

shutdown

```
public void shutdown()
             throws RemoteException
```

Shutdown the activation system. Destroys all groups spawned by the activation daemon and exits the activation daemon.

Throws:

`RemoteException` - if failed to contact/shutdown the activation daemon

Since:

1.2

---

setActivationDesc

```
public ActivationDesc setActivationDesc(ActivationID id,
                                        ActivationDesc desc)
                                 throws ActivationException,
                                        UnknownObjectException,
                                        UnknownGroupException,
                                        RemoteException
```

Set the activation descriptor, `desc` for the object with the activation identifier, `id`. The change will take effect upon subsequent activation of the object.

Parameters:

`id` - the activation identifier for the activatable object
`desc` - the activation descriptor for the activatable object

Returns:

the previous value of the activation descriptor

Throws:

`UnknownGroupException` - the group associated with `desc` is not a registered group
`UnknownObjectException` - the activation `id` is not registered
`ActivationException` - for general failure (e.g., unable to update log)
`RemoteException` - if remote call fails

Since:

1.2

See Also:

`getActivationDesc(java.rmi.activation.ActivationID)`

---

setActivationGroupDesc

```
public ActivationGroupDesc setActivationGroupDesc(ActivationGroupID id,
                                                  ActivationGroupDesc desc)
                                           throws ActivationException,
                                                  UnknownGroupException,
                                                  RemoteException
```

Set the activation group descriptor, `desc` for the object with the activation group identifier, `id`. The change will take effect upon subsequent activation of the group.

Parameters:

`id` - the activation group identifier for the activation group
`desc` - the activation group descriptor for the activation group

Returns:

the previous value of the activation group descriptor

Throws:

`UnknownGroupException` - the group associated with `id` is not a registered group
`ActivationException` - for general failure (e.g., unable to update log)
`RemoteException` - if remote call fails

Since:

1.2

See Also:

`getActivationGroupDesc(java.rmi.activation.ActivationGroupID)`

---

getActivationDesc

```
public ActivationDesc getActivationDesc(ActivationID id)
```

```
                                    throws ActivationException,
                                           UnknownObjectException,
                                           RemoteException
```

Returns the activation descriptor, for the object with the activation identifier, `id`.

Parameters:
`id` - the activation identifier for the activatable object
Returns:
the activation descriptor
Throws:
`UnknownObjectException` - if `id` is not registered
`ActivationException` - for general failure
`RemoteException` - if remote call fails
Since:
1.2
See Also:
`setActivationDesc(java.rmi.activation.ActivationID, java.rmi.activation.ActivationDesc)`

---

getActivationGroupDesc

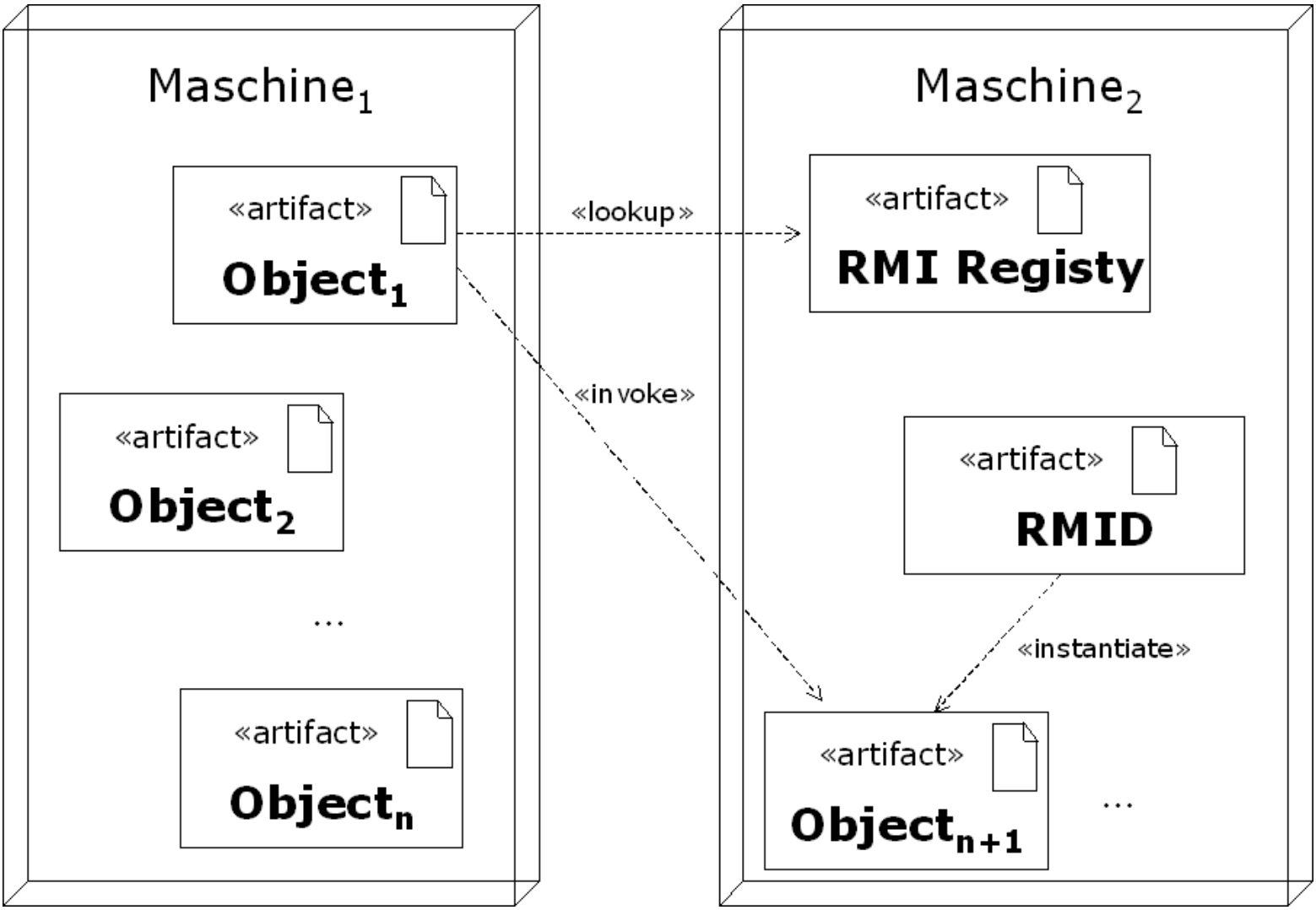
```
public ActivationGroupDesc getActivationGroupDesc(ActivationGroupID id)
                                           throws ActivationException,
                                                  UnknownGroupException,
                                                  RemoteException
```

Returns the activation group descriptor, for the group with the activation group identifier, `id`.

Parameters:
`id` - the activation group identifier for the group
Returns:
the activation group descriptor
Throws:
`UnknownGroupException` - if `id` is not registered
`ActivationException` - for general failure
`RemoteException` - if remote call fails
Since:
1.2
See Also:
`setActivationGroupDesc(java.rmi.activation.ActivationGroupID, java.rmi.activation.ActivationGroupDesc)`

Submit a bug or feature

For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.


Maschine1
Maschine2
«artifact»
Object1
«lookup»
«artifact»
RMI Registy
«invoke»
«artifact»
Object2
«artifact»
RMID
...
«instantiate»
«artifact»
Objectn
«artifact»
Objectn+1
...

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
	String sayHello() throws RemoteException;
}
```

```
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class HelloClient {

	private static String message = "";

	public static void main(String args[]) {

		System.setSecurityManager(new RMISecurityManager());

		try {
			Hello obj =
				(Hello) Naming.lookup("rmi://53.16.71.55:1099/HelloServer");
			message = obj.sayHello();
			System.out.println(message);

		} catch (Exception e) {
			System.out.println("HelloClient exception: " + e.getMessage());
			e.printStackTrace();
		}
	}

}
```

```
import java.rmi.Naming;
import java.rmi.RMISecurityManager;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject implements Hello {
	public HelloImpl(String protocol, byte[] pattern) throws RemoteException {
		super(
			0,
			new XorClientSocketFactory(protocol, pattern),
			new XorServerSocketFactory(protocol, pattern));
	}

	public String sayHello() throws RemoteException {
		StringBuffer sb = new StringBuffer();
		for (int i=0;i<100;i++)
			sb.append("X");
		return sb.toString();
	}

	public static void main(String args[]) {

		System.setSecurityManager(new RMISecurityManager());
		byte[] aPattern = {(byte) 1011 };
		try {
			HelloImpl obj = new HelloImpl("xor", aPattern);
			Naming.rebind("rmi://53.16.71.55:1099/HelloServer", obj);
			System.out.println("HelloServer bound in registry");
		} catch (Exception e) {
			System.out.println("HelloImpl err: " + e.getMessage());
			e.printStackTrace();
		}
	}
}
```

```
import java.io.IOException;
import java.io.Serializable;
import java.net.Socket;
import java.rmi.server.RMIClientSocketFactory;
import java.rmi.server.RMISocketFactory;

public class XorClientSocketFactory
	implements RMIClientSocketFactory, Serializable {
	private static RMISocketFactory defaultFactory =
		RMISocketFactory.getDefaultSocketFactory();

	private String protocol;
	private byte[] data;

	public XorClientSocketFactory(String protocol, byte[] data) {
		this.protocol = protocol;
		this.data = data;
	}
	public Socket createSocket(String host, int port) throws IOException {
		if (data == null || data.length != 1)
			throw new IOException("invalid argument for XOR protocol");
		return new XorSocket(host, port, data[0]);
	}
}
```

```
import java.io.FilterInputStream;
import java.io.IOException;
import java.io.InputStream;

class XorInputStream extends FilterInputStream {

	private byte pattern = (byte) (170 & 0xFF);

	public XorInputStream(InputStream in, byte pattern) {
		super(in);
		if (pattern != 0)
			this.pattern = pattern;
	}

	public int read() throws IOException {
		int b = in.read();
		//If not end of file or an error, truncate b to one byte
		if (b != -1)
			b = (b ^ pattern) & 0xFF;

		return b;
	}

	public int read(byte b[], int off, int len) throws IOException {
		int numBytes = in.read(b, off, len);

		if (numBytes <= 0)
			return numBytes;

		int i = 0;
		for (; i < numBytes; i++) {
			b[off + i] = (byte) ((b[off + i] ^ pattern) & 0xFF);
		}
		return i;
	}
}
```

```
import java.io.FilterOutputStream;
import java.io.IOException;
import java.io.OutputStream;

class XorOutputStream extends FilterOutputStream {

	private byte pattern = (byte) (170 & 0xFF);

	public XorOutputStream(OutputStream out, byte pattern) {
		super(out);
		if (pattern != 0)
			this.pattern = pattern;
	}

	public void write(int b) throws IOException {
		out.write(b ^ pattern);
		out.flush();
	}

	public void write(byte b[], int off, int len) throws IOException {
		for (int i = 0; i < len; i++)
			write(b[off + i]);
	}
}
```

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

class XorServerSocket extends ServerSocket {

	private byte pattern;

	public XorServerSocket(int port, byte pattern) throws IOException {
		super(port);
		this.pattern = pattern;
	}

	public Socket accept() throws IOException {
		Socket s = new XorSocket(pattern);
		implAccept(s);
		return s;
	}
}
```

```
import java.io.IOException;
import java.io.Serializable;
import java.net.ServerSocket;
import java.rmi.server.RMIServerSocketFactory;
import java.rmi.server.RMISocketFactory;

public class XorServerSocketFactory
	implements RMIServerSocketFactory, Serializable {
	private static RMISocketFactory defaultFactory =
		RMISocketFactory.getDefaultSocketFactory();

	private String protocol;
	private byte[] data;

	public XorServerSocketFactory(String protocol, byte[] data) {
		this.protocol = protocol;
		this.data = data;
	}

	public ServerSocket createServerSocket(int port) throws IOException {

		System.out.println("using xor sockets");
		if (data == null || data.length != 1)
			throw new IOException("invalid argument for XOR protocol");
		return new XorServerSocket(port, data[0]);
	}
}
```

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

class XorSocket extends Socket {

	private byte pattern;

	private InputStream in = null;

	private OutputStream out = null;

	public XorSocket(byte pattern) throws IOException {
		super();
		this.pattern = pattern;
	}

	public XorSocket(String host, int port, byte pattern) throws IOException {
		super(host, port);
		this.pattern = pattern;
	}

	public InputStream getInputStream() throws IOException {
		if (in == null) {
			in = new XorInputStream(super.getInputStream(), pattern);
		}
		return in;
	}

	public OutputStream getOutputStream() throws IOException {
		if (out == null) {
			out = new XorOutputStream(super.getOutputStream(), pattern);
		}
		return out;
	}
}
```

Core Java

Java 2 Platform, Standard Edition v 1.4.1

Printable Page

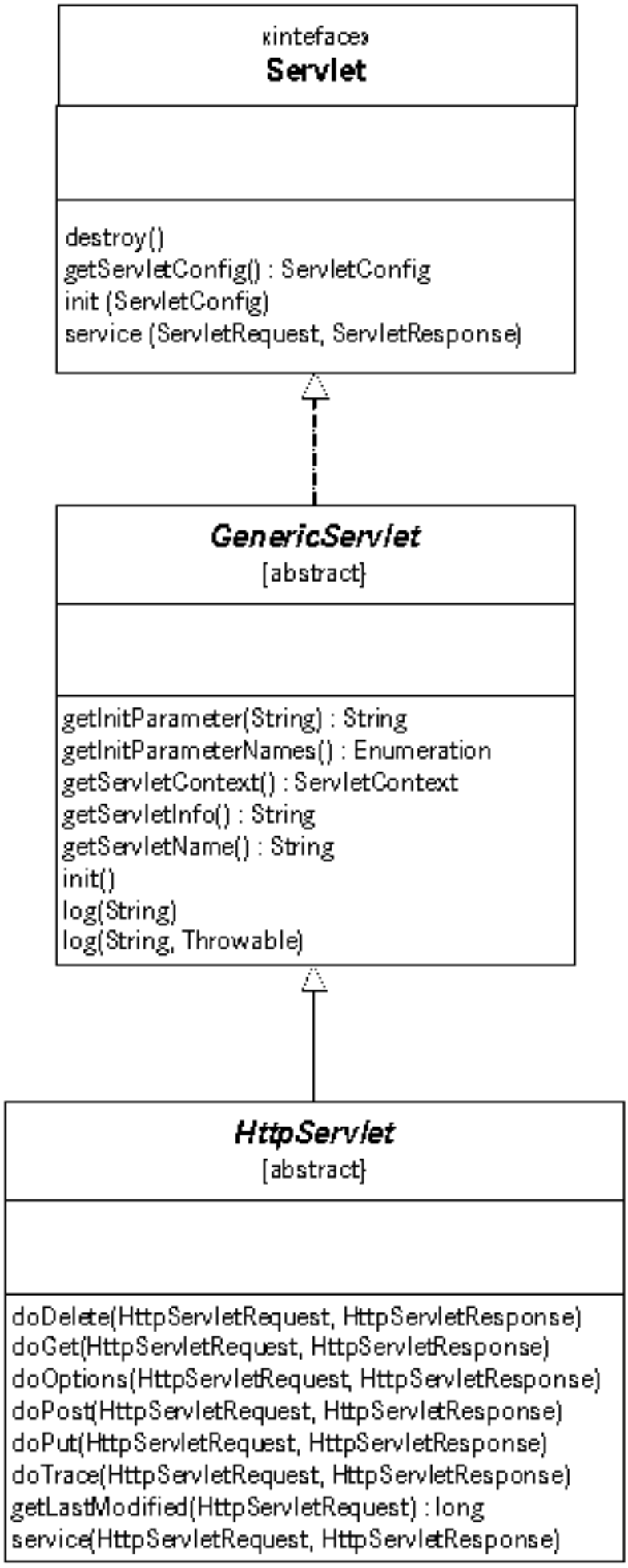
Version 1.4.1
Software

Products listed on this page have completed the Sun End of Life process.

Users should upgrade to current product versions.

These products are no longer supported by Sun.

For developer requirements, all products that have completed the EOL transition period have moved to the Archive area.


«inteface»
Servlet
destroy()
getServletConfig() : ServletConfig
init (ServletConfig)
service (ServletRequest, ServletResponse)
GenericServlet
[abstract]
getInitParameter(String) : String
getInitParameterNames() : Enumeration
getServletContext() : ServletContext
getServletInfo() : String
getServletName() : String
init()
log(String)
log(String, Throwable)
HttpServlet
[abstract]
doDelete(HttpServletRequest, HttpServletResponse)
doGet(HttpServletRequest, HttpServletResponse)
doOptions(HttpServletRequest, HttpServletResponse)
doPost(HttpServletRequest, HttpServletResponse)
doPut(HttpServletRequest, HttpServletResponse)
doTrace(HttpServletRequest, HttpServletResponse)
getLastModified(HttpServletRequest) : long
service(HttpServletRequest, HttpServletResponse)

```
import java.io.IOException; import java.io.PrintWriter; import java.util.Date; import javax.servlet.
ServletException; import javax.servlet.http.HttpServlet; import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse; public class HelloWorld extends HttpServlet { public
void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException,
ServletException { response.setContentType("text/html"); PrintWriter out = response.getWriter(); out.
println(""); out.println(""); out.println(""); out.println(""); out.println(""); out.println("
```

Hello World!

```
"); out.println("

Current Date: "+new Date().toString()+"

"); out.println(""); out.println(""); } }
```

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SimpleService extends HttpServlet {
	public void doGet(HttpServletRequest req, HttpServletResponse resp)
		throws IOException {
		resp.setContentType("text/xml");
		PrintWriter out = resp.getWriter();
		out.println(
			Integer.parseInt(
				req.getParameter("a")
					+ Integer.parseInt(req.getParameter("b"))));
	}
}
```

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;

import javax.servlet.ServletInputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Methods extends HttpServlet {
	private HashMap hm;
	public void init() {
		hm = new HashMap();
	}

	public void doDelete(HttpServletRequest req, HttpServletResponse resp) {
		hm.remove(req.getParameter("key"));
	}

	public void doGet(HttpServletRequest req, HttpServletResponse resp)
		throws IOException {
		resp.setContentType("text/html");
		PrintWriter out = resp.getWriter();
		out.println(hm.get(req.getHeader("key")));
	}

	public void doPost(HttpServletRequest req, HttpServletResponse resp) {
		byte line[] = new byte[100];
		String key = null;
		String value;
		ServletInputStream sip = null;
		try {
			sip = req.getInputStream();
		} catch (IOException e) {
			e.printStackTrace();
		}
		try {
			sip.readLine(line, 0, 100);
		} catch (IOException e1) {
			e1.printStackTrace();
		}
		String lineStr = new String(line, 0, line.length);
		key = lineStr.substring(lineStr.indexOf(":") + 1, lineStr.indexOf("&"));
		value =
			lineStr.substring(lineStr.lastIndexOf(":") + 1, lineStr.length());
```

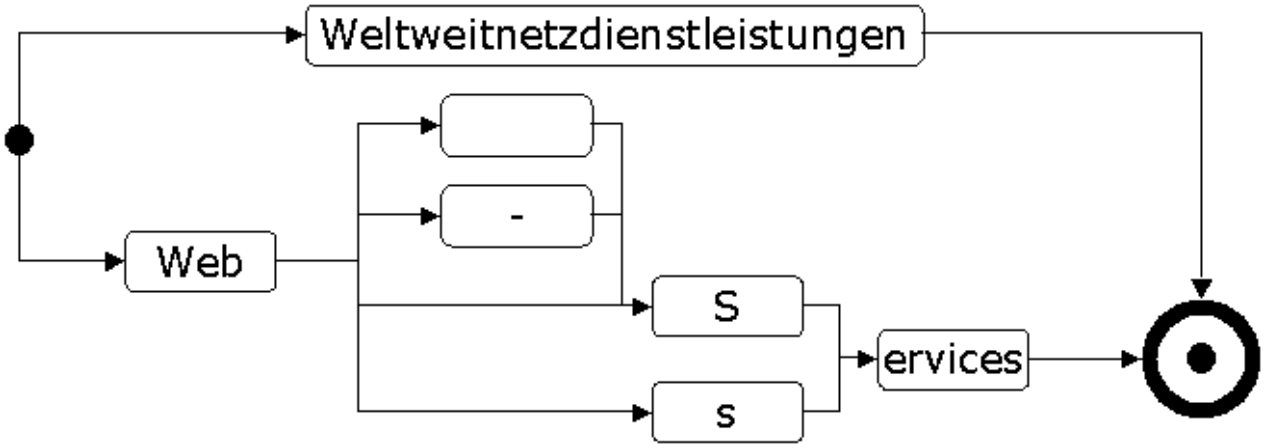
```
            hm.put(key, value);
        }
}
```

Diese Seite versammelt einige grundlegende Informationen zum im Entstehen begriffenen Technikgebiet *Web Services*.
Neben einer Übersicht verschiedener Begriffsdefinitionen und einigen Links findet sich auch ein Motivationsbeispiel, das die Umsetzung dreier kooperierender Web-Dienste zeigt.

Nachgestichelt: Wie schreibt man eigentlich Web Service?!

Oder: Ein Web-Service ist ein Web Service ist ein WebService ist ein Webservice ...?!
Neben der unten dargestellten Vielfalt der Sinngebungen was ein Web-Service *ist* oder *sein soll* existiert parallel dazu ein beachtlich unterhaltsam geführter Streit wie die korrekte Schreibung des Begriffs zu handhaben sei, was ein weiteres Indiz dafür liefert, daß im Deutschen bisweilen wahrhafte orthographische Grabenkämpfe hinsichtlich Fragen des „richtigen“ Buchstabengebrauchs ausgefochten werden.
So konstatierte jüngst die Zeitschrift DER SPIEGEL --- nach überstandenem Angriff einer Phalanx unberechtigt gesetzter Apostrophierungen und Heeren verwaister öffnender Klammern --- einen weiteren Todesritt auf die kulturelle Identität des (untergehenden) Abendlandes. So droht der unkontrollierte Zustrom einer Flut angelsächsischer Leerzeichen die germanischen Bindestriche von ihrem angestammten Platze im Symbolsystem zu verdrängen und gleichzeitig die hergebrachten Bindungen ehedem konkateniert existierender Worte zu zerstören.
So ist die Schreibung „Web Service“ in deutschen Texten gradewegs abzulehenen, selbst wenn sich das Schriftstück mit modernster Web- und Fadentechnik beschäftigt. Ebenso ist jeglich teutonischer Assimilierungsversuch durch Bindestriche (à la „Web-Service“) zum Scheitern verurteilt, auch wenn dieser im vorliegenden Falle versucht eine Brücke zwischen den (Sprach-)Welten zu schlagen ...
Noch durchsichtiger müssen in diesem Lichte Ansätze der Zusammenführung wirken die Konglomerate wie „WebServices“ zu „Webservices“ umzuschreiben suchen ...

Daher! Angesichts der Bedrohung kann die einzig korrekte Schreibung, welche sich bis dato aus nicht nachvollziehbaren Gründen nicht im Duden findet, ausschließlich „Weltweitnetzdienstleistung“ (W2NDL) lauten!
... Allerdings müssen wir diese puristische Konkretisierung mit einem geringen -- vernachlässigbaren -- Mächtigkeitsverlust unserer Kommunikation bezahlen. Schlichtweg scheint auch hier -- wie allzuoft in der Informatik oder IT -- die deutsche muttersprachliche Korrektheit dem vereinfachten Bedeutungstransport diametral entgegengesetzt zu sein.
In diesem Sinne versammelt diese Seite Informationen über ...



;)

<Nachtrag>
Nichts scheint so nebensächlich, als daß es nicht Gegenstand von durchaus hitzig geführten Debatten werden könnte ...
Kurze Zeit nach dem ich meine Betrachtungen zur *richtigen* Schreibung des Begriffs Web Service ins Netz stellte entspann sich in den W3C Arbeitsgruppen *Web Service Description* und *Web Service Architecture* eine durchaus heiter zu beobachtender Streit über eben jene Schreibung ...
(Vorerst) Normativ kann daher festgestellt werden, daß sich bisher knapp *Web Service* gegenüber *Web service* durchgesetzt zu haben scheint. Die Verfechter der Groß-Klein-Variante haben jedoch schon angekündigt die Entscheidung zentraler Bedeutung und Tragweite im Kreise der *Technical Architecture Group* des W3C diskutiert wissen zu wollen, da ausschließlich dieser Entscheidungen über die grundlegenden Prinzipien des Web obliegen -- und Neuschreib scheint inzwischen eins davon zu sein. **</Nachtrag>**

Begriffsdefinitionen

Mike Ricciuti

Web services is an esoteric data exchange technology that has mostly been used as a platform for connecting information infrastructures within companies.

Quelle: Will Jini-like wishes come true?

Matthew McDonald

>

Web Services are a modified version of COM with a little difference. They are individual units of programming logic that exist on a Web server and can be integrated into Web applications created with ASP .NET as well as desktop applications created with VB.NET or C#. Web Services can be created with .NET but consumed on other platforms.

Quelle: ASP.NET: The Complete Reference, Tata McGraw Hill

IBM und Unisys

Web Services are a new, standards-based approach to build integrated applications that run across an intranet, extranet, or the Internet. The approach represents a major evolution in how systems connect and interact with each other.

Quelle: *CWM Web Services Specification*, OMG-Dokument ad/2001-10-07

Computer Associates (Dimitri Tcherevik)

A Service, any service, can be defined as a software component that can be:

- *Described* in a formal language
- *Published* to a registry of services
- *Discovered* through standard mechanisms
- *Invoked* over a network
- *Composed* with other services

Quelle

Frankfurter Allgemeine Zeitung

Web Services sind Softwarebausteine, die Programme, die auf unterschiedlichen Netzwerkrechnern laufen, über das Internet zu einer Anwendung miteinenader verknüpfen.
F.A.Z. vom 14. Oktober 2003, S. 18

Andreas Schmidt

Web Services sind verteilte, lose gekoppelte und wiederverwendbare Softwarekomponenten, auf die über Standard-Internetprotokolle programmatisch zugegriffen werden kann.
Quelle

Dieter Fensel

Web Services sind Services, die über das Web erreicht werden können.
Quelle

Adam Bosworth

The term Web Services refers to an architecture that allows applications to talk to each other.

Quelle

XML Key Management Specification (XKMS)

A service that is accessible by means of messages sent using standard web protocols, notations and naming conventions.
Quelle

Munich Web Services Circle

Web Services sind eigenständige, selbst beschreibende Anwendungen, für eine professionelle und komfortable Nutzung des Internets. Sie eignen sich sehr gut, um beispielsweise Geschäftsprozesse von der Auftragserteilung bis zur Versandabwicklung inclusive einer Produktversicherung zu vereinfachen. Diese Web Services werden in der Sprache XML geschrieben, strukturiert und in einem Register abgelegt. Übertragen werden sie über das Internet mittels eines Browsers, entweder zu einem PC oder einer anderen Anwendung im Unternehmen, wie z.B. zu einem mobilen Personal Da Assistant (PDA).
Quelle.

Anmerkung: Was unter einem „Personal Da Assistant“ verstanden wird, kann ich leider auch nicht aufklären; Vermutlich ein Personal Assistant der immer da ist, zumindest wenn man ihn braucht.

Arbeitskreis *Web Services* der Gesellschaft für Informatik

Web-Services sind selbstbeschreibende, gekapselte Software-Komponenten, die eine Schnittstelle anbieten, über die ihre Funktionen entfernt aufgerufen, und die lose durch den Austausch von Nachrichten miteinander gekoppelt werden können. Zur Erreichung universeller Interoperabilität werden für die Kommunikation die herkömmlichen Kanäle des Internets verwendet. Web-Services basieren auf den drei Standards WSDL, SOAP und UDDI: Mit WSDL wird die Schnittstelle eines Web-Services spezifiziert, via SOAP werden Prozedurfernaufrufe übermittelt und mit UDDI, einem zentralen Verzeichnisdienst für angebotene Web Services, können andere Web-Services aufgefunden werden.
Quelle: Ankündigung des Symposiums *Entwicklung Web-Service-basierter Anwendungen*

Java Web Services

A Web Service is a piece of business logic, located somewhere on the Internet, that is accessible through standard-based Internet protocols such as SMTP or HTTP.
Quelle: Java Web Services, S. 1

Loosely Coupled

A Web Service is nothing more than an application which other applications can communicate via a network, most often using XML.
Quelle: Loosely Coupled, S. 26

SCO

Web Services allow applications to share data and invoke capabilities from other applications without regard to how those applications were built, what platform they run on, or what devices are used to access them.

World Wide Web Consortium (Web Service Architecture Working Group)

A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols.
Source

Wrox Press: Professional Web Services

Web Services are modular, self-describing applications that can be published, located and invoked from just about anywhere on the Web or a local network. The provider and the consumer of the XML Web service do not have to worry about operating system, language, environment, or component model used to create or access the XML Web service, as they are based on ubiquitous and open Internet standards, such as XML, HTTP, and SMTP.

Intel

Web services encompass a vision of a fully integrated computing network that include PCs, servers, handheld devices, programs, applications and network equipment, all working together. This network can perform distributed computation with the best-matched device for the task and deliver the information on a timely basis in the form needed by the user.
Quelle

IBM

Web services are self-describing, self-contained, modular applications that can be mixed and matched with other Web services to create innovative products, processes, and value chains. Web services are Internet applications that fulfill a specific task or a set of tasks that work with many other web services in a manner to carry out their part of a complex work flow or a business transaction. In essence, they enable just-in-time application integration and these web applications can be dynamically changed by the creation of new web services.
Quelle

H. Kreger

A Web Service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging.
Source

SUN

A Web service is, simply put, application functionality made available on the World Wide Web. A Web service consists of a network-accessible service, plus a formal description of how to connect to and use the service. The language for formal description of a Web service is an application of XML. A Web service description defines a contract for how another system can access the service for data, or in order to get something done. Development tools, or even autonomous software agents, can automatically discover and bind existing Web services into new applications, based on the description of the service.

M. Pawlan

A Web service describes specific business functionality exposed by a company, usually through an Internet connection, for the purpose of providing a way for another company or software program to use the service.
Source

D. Reichardt

Web services are self-describing components that can discover and engage other Web services or applications to complete complex tasks over the Internet.
Source

J. Kao (in a presentation prepared for SUN)

A web service is an application that accepts requests from other systems across the Internet or an Intranet, mediated by lightweight, vendor-neutral communications technologies. These communications technologies allow any network-enabled systems to interact. As technologies mature, a web service will encompass additional special functionality geared towards performing multiparty B2B collaboration.
Quelle.

M. Fisher

Web services, in the general meaning of the term, are services offered via the Web. In a typical Web services scenario, a business application sends a request to a service at a given URL using the SOAP protocol over HTTP. The service receives the request, processes it, and returns a response. An often-cited example of a Web service is that of a stock quote service, in which the request asks for the current price of a specified stock, and the response gives the stock price. This is one of the simplest forms of a Web service in that the request is filled almost immediately, with the request and response being parts of the same method call.
Source

Microsoft

A Web Service is a unit of application logic providing data and services to other applications. Applications access Web Services via ubiquitous Web protocols and data formats such as HTTP, XML, and SOAP, with no need to worry about how each Web Service is implemented. Web Services combine the best aspects of component-based development and the Web, and are a cornerstone of the Microsoft .NET programming model.

Gartner Group I

Web services are loosely coupled software components delivered over Internet standard technologies. A Web service represents a business function or a business service and can be accessed by another application - e.g., a client, a server or another Web service - over public networks using generally available protocols and transport, i.e., SOAP over HTTP. It can also be accessed directly by a user through a client interface. This business service can be implemented as a stand-alone application or as a series of applications linked together by an application integration infrastructure. From the point of view of the calling application, the Web service is a modular entity that delivers services on demand through a well-defined programmatic interface. In effect, Web services comprise the overall capability of finding and running programs across the Internet - much in the way that software programs run on a PC. As such, Web services will eventually act as a quasi-Internet "operating system" and use protocols to link and manage an array of Internet-connected servers, PCs, appliances and wireless devices. The underlying business goal of Web services is to enable enterprises to focus on their core competencies while outsourcing all other functions. Web services would be the "tap" to turn on to access, deploy and manage such essential outsourced functions as credit authorization, just-in-time productivity applications and quality-of-service monitoring. Recent e-business trends have included a movement toward outsourcing applications and hosting systems through service providers. Web services allow the delivery of core business competencies in the form of software processes accessible in a given value chain. This allows each member of the chain to produce and consume business services without the need for an intermediary such as an application service provider (ASP). In essence, each enterprises takes on the task of being both a service consumer and a service producer. Through 2004, the primary competitive value derived from Web services will be the delivery of core business competencies directly into trading communities rather than through an intermediate service provider (0.7 probability).

Gartner Group II

A Web service is a custom end-to-end application that interoperates with other commercial and custom software through a family of XML interfaces (like SOAP, UDDI and WSDL) to perform useful business functions.
Quelle.

WebServices.org

WebServices are encapsulated, loosely coupled contracted functions offered via standard protocols. Hereby the term encapsulated means the inability of a user to gather information about the internal service realization. Loosely Coupled refers to the possibility of changing the implementation of one function without requiring changes to clients invoking the function. Encapsulation builds a prerequisite of loosely coupled systems. If a description of a function is publicly available the interface offered by the function to invoke it can be considered to be contracted. This means the behavior of the function and its input and output parameters are fixed. Finally, the usage of standard protocols refers to a technical infrastructure which relies on open, widely published and freely available protocols.

World Wide Web Consortium

The same way programmatic interfaces have been been available since the early days of the World Wide Web via HTML forms, programs are now accessible by exchanging XML data through an interface, e.g. by using SOAP Version 1.2, the XML-based protocol produced by the XML Protocol Working Group. The services

provided by those programs are called Web services.

Hewlett-Packard

Web services are modular and reusable software components that are created by wrapping a business application inside a Web service interface. Web services communicate directly with other web services via standards-based technologies. These standards-based communications allow Web services to be accessed by customers, suppliers, and trading partners independent of hardware, operating system, or even programming environment.

Quelle

Globus Projekt

The term *Web services* describes an important emerging distributed computing paradigm that differs from other approaches such as DCE, CORBA, and Java RMI in its focus on simple, Internet-based standards (e.g., eXtensible Markup Language: XML) to address heterogeneous distributed computing. Web services define a technique for describing software components to be accessed, methods for accessing these components, and discovery methods that enable the identification of relevant service providers. Web services are programming language-, programming model-, and system software-neutral.

Oellermann: *Architecting Web Services*

Any process that can be integrated into external systems though valid XML documents over Internet protocols.

Intercea

The term *Web Services* is a misnomer. Consider the way in which the word Web has entered into everyday language. It is used as a generic term for the Internet by those outside the realms of business technology. The term is open to interpretation and without a clear definition will continue to be misconstrued. The term *Web services* has been used to describe functional services offered by websites as well as the services offered by IT professional services firms. It is no wonder there is widespread confusion as neither of these is correct.
Source

IDC

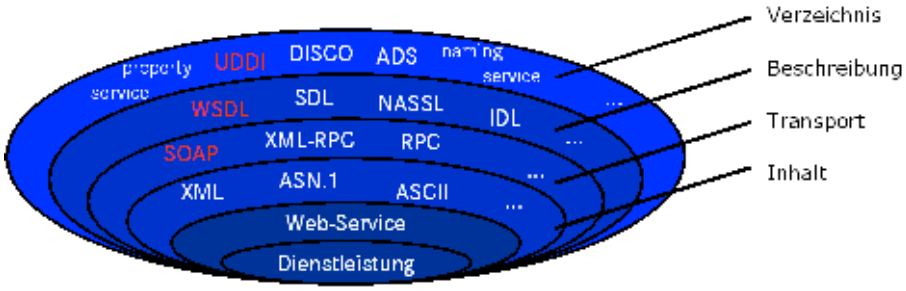
Web Services are a standards-based software technology that lets programmers and integrators combine existing and new systems or applications in new ways over the Internet, within a company's boundaries, or across many companies. Web Services allow interoperability between software written in different programming languages, developed by different vendors, or running on different operating systems or platforms.
Web Services: Analyst: Sophie Janne Mayo. March 2002

Weitere Definitionen

- What are Web Services @ devx.com

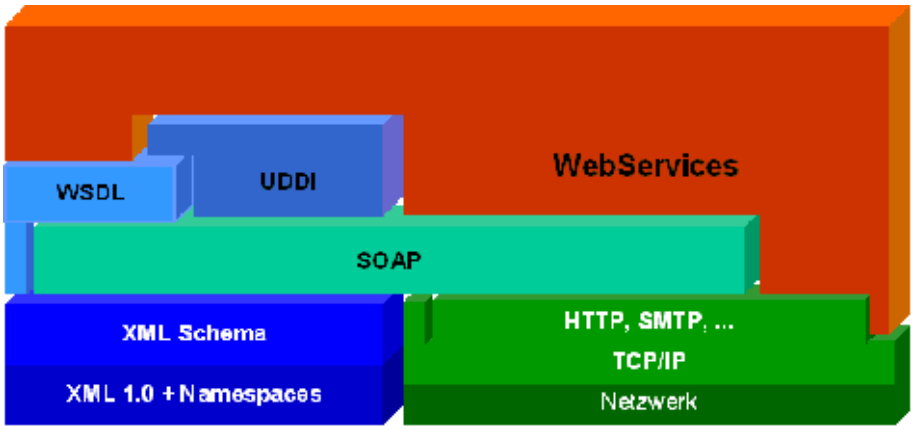
Eigene Definition

Das Zwiebelschalenmodell definiert einen Web-Service zunächst als Dienstleistung im klassischen Sinne (mit Merkmalen wie Immaterialität und damit nicht-Lagerfähigkeit, standortunabhängiger Produktion und der mangelnden Entkopplung von Erzeugung und Konsumption), die über das World Wide Web erbracht wird.
Die einzigen hierbei verbindlich benötigten Standards sind die namensgebenden Web-Techniken. Hierunter fallen neben der physischen Hardwareinfrastruktur auch die benötigten Internet-Protokolle wie TCP/IP.
Die darauf aufsetzenden Stufen erhöhen sukzessive die Interoperabiltität der durch Web-Mittel erbrachten Dienstleistungen und daher in der Konsequenz auch die Austauschbarkeit des Dienstes durch den Konsumenten.
Werden Web-Dienste zunächst nur auf Basis der im Internet gebräuchlichen Vermittlungsstandards erbracht, so fügt die Verwendung der Extensible Markup Language (XML) zur Darstellung des Dienstleistungsinhaltes eine weitere Stufe der Vereinheitlichung hinzu. Durch sie werden sowohl die Dienstanforderung selbst als auch die Rückübermittlung des Dienstergebnisses im selben Metaformat dargestellt. Hierin zeigt sich nochmals die diskutierte neue Rolle der Standards.
XML legt dabei nicht ein konkretes Nachrichtenformat fest, sondern eröffnet dem Dienstanbieter die Möglichkeit, „sein“ Format auf Basis der Metasprache selbst zu definieren.



Offenkundig ließe sich dieses Ziel auch durch andere Inhaltsdarstellungen -- wie die in der Graphik beispielhaft erwähnten *ASN.1* oder *EDI* -- erreichen. Der XML-Einsatz stellt hierbei keineswegs einen revolutionären Ansatz dar, sondern liegt vielmehr schon allein wegen des Verbreitungsgrades und den damit einhergehenden Auswirkungen wie Werkzeugunterstützung und Ähnlichem nahe.
Ausgehend von normiert ausgedrückten Inhalten drängt sich die Betrachtung des Kommunikationsprozesses nachgerade auf. Gelingt es, den Interaktionsablauf ebenfalls standardisiert abzuwickeln, so würde damit das Einsatzfeld der angebotenen Dienstleistung nochmals erweitert. Einen Ansatz hierzu stellen die bekannten Mechanismen zur Abwicklung entfernter Methodenaufrufe sog. *Remote Procedure Calls* (RPC) dar. Unter Berücksichtigung der Auswahl von XML als Inhaltsformat liegen Ansätze wie XML-RPC und dessen Weiterentwicklung, das *Simple Object Access Protocol* (SOAP), auf der Hand, da sie die Idee neutraler Funktionsaufrufe mit XML-codierten Inhalten kombinieren.
Auf dieser Interoperabilitätsstufe umfaßt die Definition bereits über das Web erbrachte Dienstleistungen, die mittels standardisierter Kommunikationsmechanismen abgestimmte Inhalte in einem ebenfalls abgestimmten Format transportieren.
Immernoch bleibt jedoch die Natur der angebotenen Leistung für den Interessenten im Dunkeln.
Diesem Manko abzuhelfen schicken sich Beschreibungssprachen wie die *Web Service Description Language* (WSDL) an. Ihre Konzeption orientiert sich an Darstellungsformen zur Beschreibung technischer Schnittstellen wie die bekannte CORBA Interface Definition Language (IDL). In jüngerer Zeit wurden einige XML-

Vokabulare wie die *Network Accessible Service Specification Language* (NASSL) oder die *Service Definition Language* (SDL) vorgeschlagen, die allerdings keine übergreifende Unterstützung fanden. Diese wird jedoch der durch Microsoft und IBM gemeinsam entwickelten *Web Service Description Language* zuteil. Ihr Ansatz findet breite Zustimmung in der Anwendergemeinde und wird inzwischen auch durch eine W3C-Arbeitsgruppe als Grundlage eines Beschreibungsstandards für Web-Dienste diskutiert.
Die durch WSDL bereitgestellten deskriptiven Inhalte erlauben es potentiellen Nutzern, sich über Dienste hinsichtlich ihres Angebots und der Abwicklungsmodalitäten wie etwa Aufrufkonventionen oder zu übergebende Parameter zu informieren. Offen bleibt in diesem Zusammenhang die Frage nach der Ermittlung der für den potentiellen Nutzer interessanten Dienste aus dem Angebot im Web.
Die Erreichung dieser letzten Interoperabilitätsebene setzt sich der Ansatz *Universal Service Description, Discovery, and Integration* zum Ziel. Die grundlegende Organisation des Verzeichnisdienstes orientiert sich dabei an den bekannten Gelben Seiten. Ähnlich der dort anzutreffenden Mimik werden mit UDDI Dienst-Angebote klassifiziert und durch eine Reihe vordefinierter Zugriffsroutinen angeboten.
Auch an dieser Stelle greifen die Web-Service-Architekten auf bewährte Ideen wie die CORBA Services *naming* und *property* zurück. Sie bildeten einen der Ausgangspunkte der inzwischen nicht mehr weiterentwickelten UDDI-Vorläuferprotokolle DISCO und ADS.



Zusammenfassend läßt sich daher ein Web-Service als Komponente auffassen, die ihre Funktionalität über eine veröffentlichte Schnittstelle anbietet und über ein offenes, im Internet verwendetes Protokoll zugreifbar ist.
In der technischen Umsetzung einer Web-Service-Architektur sind jedoch nicht alle Protokolle, Sprach- und Beschreibungsebenen zwingend zu realisieren.

Konferenzen und Veranstaltungen

- Web Services Workshop 2004 @ Berliner XML Tage New
- European Conferece on Web Services 2004 (ECOWS'04)
- Second European Workshop on Object Orientation and Web Service
- Workshop on Semantic Web Services and Web Process Composition New
- Semantic Web services: Preparing to meet the World of Business Applications New

Vergangene Veranstaltungen

- Web Services/XML
- XML Web Services for the Telecom market
- Symposium *Entwicklung Web-Service-basierter Anwendungen*

- Web Services Workshop @ Berliner XML Tage
- International Conference on Web Services (ICWS'03)
- International Conference on Web Services Europe 2003 (ICWS'03-Europe)
- 2nd Nordic Conference on Web Services (NCWS'03)
- 1st Web Services Quality Workshop

Publikationen

Nachfolgend sind einige periodisch erscheinende Publikationen (Zeitschriften, Journals) zusammengestellt, die sich immer wieder, schwerpunktmäßig oder ausschließlich mit Web Services beschäftigen.

- International Journal on Web Services Research
  Englischsprachiges Journal, welches sich unter anderem folgenden Gebieten widmet:
  - Mathematic foundations for service oriented computing
  - Web Services architecture
  - Web Services security
  - Frameworks for building Web Service applications
  - Composite Web Service creation and enabling infrastructures
  - Web Services discovery
  - Resource management for Web Services
  - Solution management for Web Services
  - Dynamic invocation mechanisms for Web Services
  - Quality of service for Web Services
  - Web Services modeling
  - Web Services performance
  - UDDI enhancements
  - SOAP enhancements
  - Case studies for Web Services
  - E-Commerce applications using Web Services
  - Grid-based Web Services applications (e.g. OGSA)
  - Business process integration and management using Web Services
  - Multimedia applications using Web Services
  - Communication applications using Web Services
  - Interactive TV applications using Web Services
  - Semantic services computing

Links

- .NET Sample Web Services
- 2003: Year of B2B Web services?
- A birds-eye view of Web services
- Alliances forged on Web services standards -- W3C wird (vorerst) nicht WS-I-Mitglied
- All We Want For Christmas is a WSDL Working Group
- Applying Design Issues and Patterns in Web Services
- Assess the Progress of Web Services Adoption
- Building XML-based Web Services (white paper)
- Business Explorer for Web Services
- Code Reuse: From Objects to Components to Services
- Creating a complete Web service
- Deploying Web services with WSDL
- Developer's Guide to Building XML-based Web Services

- Ein Blick in die Kristallkugel: Was weissagen Analysten zu Web Services in 2003?
- Fat Protocols Slow Web Services
- Five Barriers to Implementing Web Services
- Gartner: What Web Services will and won't do
- Getting Started on Developing Web Services
- Getting Your Arms Around Web Services
- Infoworld: Thomson Financial invests in Web services
- Initial Thoughts on Web Services @ SUN
- Integration Brokers and Web Services
- Java gets services-ready
- Legacy systems graduate to web services
- Making Web Services More Flexible --- Web Services und asynchroner Nachrichtenaustausch mit JMS kombiniert
- Munic Web Services Circle
- My wish list for Web services
- Nordic Web Services - Hype or Reality
- Physics Institute Turns To Web Services
- Practical Web services rule the real world
- Praxisbericht: Web Services vermehren sich wie Karnickel
- Q&A: Web services security
- R. Cunnings, R. Salz: SOAP Extensions: Basic and Digest Authentication
- Real world, real Web services
- Resultate eines Web Service Quiz In Teilen ganz aufschlußreiche Einsichten in die Bewertung gewisser Web Service-Techniken am Markt.
- Sample application using Web services on WebSphere
- SAP embraces Web Services
- Second Generation Web Services
- Securing Web Services
- Securing Web Services using the Java Platform and XML
- SOAP und XML-Protocols @ jeckle.de
- Sun ships Java tools for Web services
- Syndication enhances Web services
- The ASP is being replaced by the Web service provider. How come?
- The IDL that isn't
- Tools for building Web services
- Understanding quality of service for Web services
- Using Web services for e-Commerce single sign-in
- Vote rigging illustrates importance of web services
- W3C Web Service Activity
- W3C Web Services Architecture Working Group
- W3C Web Services Description Working Group
- Web, Services and Beyond (Session @ JavaOne 2001)
- WebService Entwicklung mit .NET
- Web services: The confusion continues
- WebServices.org
- WebServices @ IBM Developer Works
- Web Services Acronyms, Demystified
- WebServices Architect
- Web services are insecureInteressanter Artikel, der nicht zuletzt den Blick für Web-Services schärft, die nicht mittels HTTP transportiert werden.
- Web Services Best Practice @ SUN
- Web Services Conversation Language (WSCL) 1.0
- Web Services Demo @ IBM Developerworks
- Web Services Glossary @ SUN
- Web Services Hosting Technology
- Web Services made easier (technical white paper @ SUN. Es führt in die APIs des Java XML-Packs ein und illustriert ein Anwendungsszenario.)
- Web Service Sublimation
- Web services will hamstring businesses

- Webserville.com -- Die beste Web Service Satire im Netz!
- What are Web services anyway?
- WSFL in Action
- WSIndex.org
- XML-based Web Services (Teil der *Designing Enterprise Applications* Seiten @ SUN)
- XMLBus
- XML Web Services Basics

Rund um die *Web Service Interoperability* Initiative

- Web Service Interoperability Organization
- Basic Profile 1.0
- Will Sun join the Web services group?
- Web services push attracts a crowd
- WSIO @ xmlhack.com

Web Service Verzeichnisse und Suchmaschinen

- Artikel @ IBM: Registering and publishing your Web service
- eXtended JEDDI -- eine kostenfreie UDDI-Implementierung
- IBMs UDDI Repository
- Microsofts UDDI Repository
- mySantra
- Realnames UDDI Repository
- RISKebiz
- The Evolution of UDDI
- UDDI-Registry @ Soapclient.com siehe auch: WebService Verzeichnisse und Suchmaschinen
- UDDI Spezifikation
- Web Service of the Day
- XMethods

Spezifikationen und offizielle Dokumente

- SOAP v1.1
- SOAP v1.2 (Part1)
- SOAP v1.2 (Part2)
- UDDI
- Web Services Inspection Language
- WSDL

Web Services vs. REST

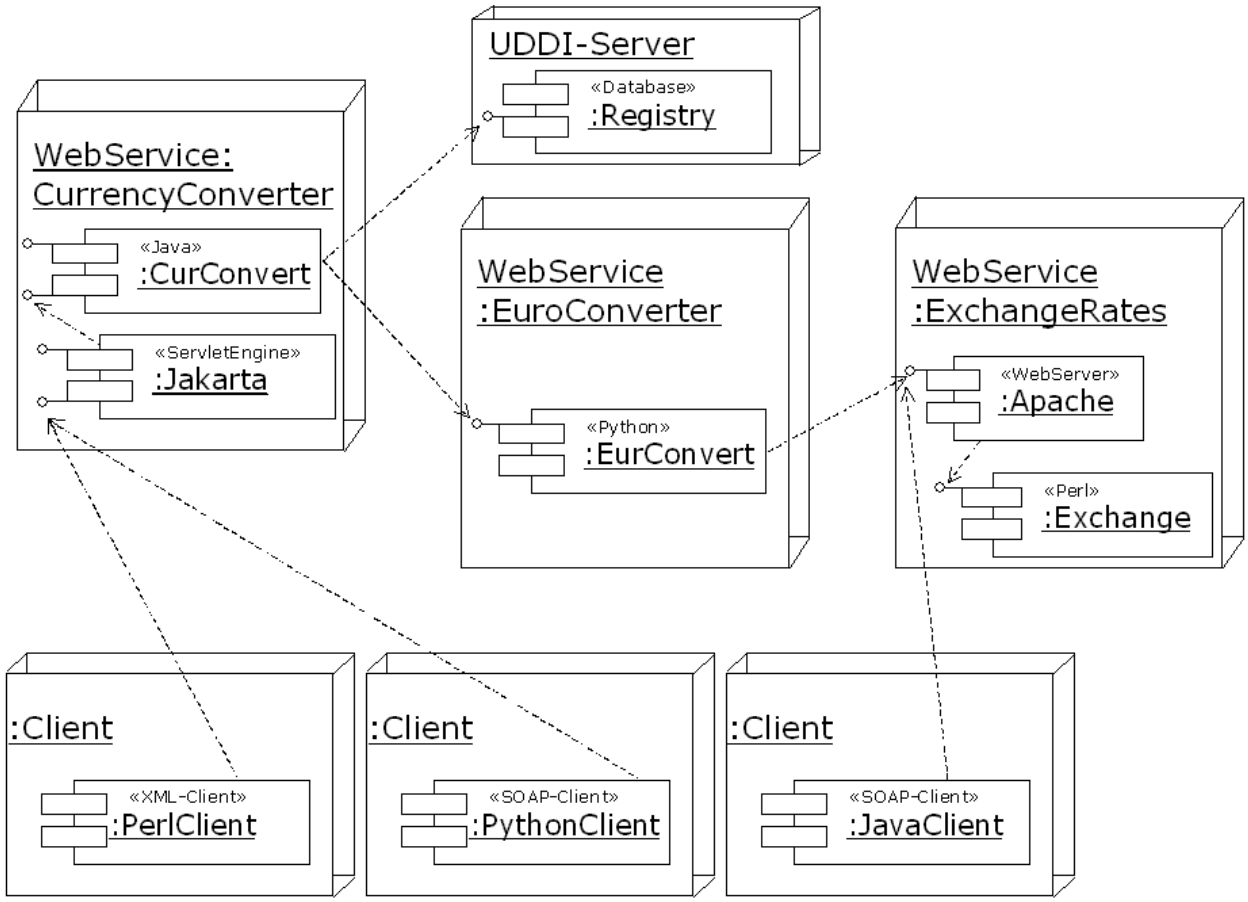
Links zum Thema

Seit einiger Zeit wird das Für-und-Wider eines XML-basierten Kommunikationsprotokolls (welches im Falle der HTTP-Verwendung die Übergabeparameter vorzugsweise per HTTP-POST versendet) gegenüber der apostrophierten Einfachheit einer rein HTTP-basierten Lösung (welche die

Übergabeparameter in einen HTTP-GET-Aufruf verpackt und als Resultat ein beliebiges XML-Dokument liefert) ausgiebig diskutiert.
Während die erste Variante im wesentlichen der aktuellen Realisierungsform SOAP-basierter Web Services entspricht wurde die das zweite Kommunikationsparadigma unter dem Namen *Representational State Transfer* (REST) bekannt.

- Building Web Services the REST Way
- eBay adds SOAP
- Interview mit Jeff Barr über die Amazon.com-Web-Services
- REST, SOAP, and the future
- REST (Dissertation von R. Fielding)
- Roots of the REST/SOAP Debate
- W3C TAG: When to use GET
- What is REST?

Ein Beispiel



Das UML-Verteilungsdiagramm (zur Notation siehe UML-Resource Center) zeigt die drei Web-Dienste `CurrencyConverter`, `EuroConverter` und `ExchangeRates` sowie nutzende Client-Applikationen und den UDDI-Verzeichnisdienst.

Alle angebotenen Web-Services sind über Verwendungsbeziehungen, die als gerichtete Kanten ausgehend vom nutzenden Dienst hin zum benutzten dargestellt sind, verknüpft und gleichzeitig eigenständig nutzbar.
Zusammen erbringen sie den Dienst der wechselseitigen Umrechnung von Geldbeträgen des europäischen Währungsraumes. Hierzu wird das durch die Europäische Zentralbank vorgegebene Verfahren der Berechnung „Zwischen-Nutzung“ des Euros (sog. *Triangulation*) benutzt. Dabei wird zunächst die Ursprungslandeswährung in das Euro-Äquivalent überführt, welches dann in die gewünschte Zielwährung umgerechnet wird.

Die Verteilung der Aufgaben ist hierbei wie folgt:

- ExchangeRates liefert auf Anfrage mit dem amtlichen Währungskürzel den entsprechenden, durch die Europäische Zentralbank festgelegten, Umrechnungskurs.
- EuroConverter rechnet einen Betrag der Landeswährung in die Eurosumme um, bzw. umgekehrt.
  Zur Berechnung wird der durch ExchangeRates bereitgestellte Wechselkurs benötigt.
- CurrencyConvert erlaubt die Umrechnung zwischen allen Währungen des europäischen Währungsraumes.
  Gemäß der Festlegung durch die Europäische Zentralbank darf dieser Vorgang ausschließlich durch Umrechnung in Euro ausgeführt werden (Triangulation). Daher muß zunächst der Ursprungsbetrag nach Euro, anschließend dieser Wert in die Zielwährung konvertiert werden.
  Zur Realisierung nutzt CurrencyConvert die durch EuroConverter angebotenen Funktionen.

Zur Realisierung der verschiedenen Dienste wurden neben *Java* auch die Skriptsprachen *Perl* und *Python* eingesetzt. Die angesprochenen Aufgaben werden durch drei verteilte SOAP-Dienste wahrgenommen:
Im Detail: ExchangeRates ist als Perl-Skript unter Verwendung des SOAP::Lite Pakets implementiert und wird über den CGI-Mechanismus an einen Apache Webserver angebunden, d.h. das Skript realisiert die Kommunikation über Standardein- und Ausgabeschnittstellen. Die festgelegten Wechselkurse sind in einem assoziativen Array gespeichert, welches direkt über eine Zeichenkette indiziert wird. Falls ein Währungskürzel nicht bekannt ist, so wird ein SOAP Client Fehler, Client.unknownCurrency, erzeugt.
Der als Python-Script (unter Verwendung der SOAPpy-Bibliothek) umgesetzte EuroConverter empfängt seine Dienstanforderungen durch direkte Überwachung des TCP-Ports 80. Zur Erfüllung der Funktion wird bei jeder Anfrage der Dienst ExchangeRates für den jeweiligen Wechselkurs konsultiert.
Der dritte Dienst, CurrencyConvert, ist in Java unter Verwendung der SOAP-Bibliothek des Apache-Projektes implementiert. Innerhalb des Web-Servers übernimmt ein Servlet die Verteilung der eingehenden SOAP-Aufrufe an die registrierten Dienste. Aus diesem Grunde kommunizieren alle Nutzer mit derselben Adresse, dem sog. *SOAP-Endpunkt*; die Auswahl der zu verwendenden Java-Klasse und die notwendige Objekterzeugung werden für den Dienstanbieter im Betrieb transparent durch die Servlet-Komponente bereitgestellt. Lediglich durch die Konfigurationsdatei des *Deployment Deskriptors* werden die als Web-Dienste angebotenen Methoden der Java-Klasse namentlich in einem XML-Dokument benannt.
Invers zur im Diagramm hervorgehobenen Aufrufbeziehung verläuft der Kommunikationsweg zur Rückgabe des Dienstergebnisses, der auch zur Übermittlung der Fehlerinformation herangezogen wird.

Die einzelnen Implementierungen im Quellcode

- Die Dienste ...
  - XML-Deployment Descriptor des Currency_Convert Dienstes
  - Python-Implementierung des Euro_Converter Dienstes
  - Perl-Implementierung des Exchange_Rates Dienstes
  - Java-Implementierung des Currency_Convert Dienstes
- Einige Clients ...
  - Java-Client für Perl-implementierten Exchange_Rates Dienst
  - Python-Client für Python-implementierten Euro_Converter Dienst
  - Java-Client für Perl-implementierten Exchange_Rates Dienst
  - Java-Client für den Java-implementierten Currency_Convert Dienst

    - Python-Client für den Java-implementierten `Currency_Convert` Dienst
    - Perl-Client für den Java-implementierten `Currency_Convert` Dienst
      Als Besonderheit nutzt diese Implementierung *keine SOAP-Bibliothek* sondern verarbeitet die benötigten XML-Dokumente direkt
- Das Ganze in Java ...
    - Server
        - `Exchange_Rates`
        - `Euro_Converter`
        - `Currency_Convert`
    - Deployment Descriptoren für Apache SOAP
        - `DDExchangeRates`
        - `DDEuroConverter`
        - `DDCurrencyConvert`
    - Beispielhafte Testclients für die Java-Implementierungen
        - `testExchangeRates.java`
        - `testEuroConverter.java`
        - `testCurrencyConvert`

Service provided by Mario Jeckle

Generated: 2004-05-24T13:36:10+01:00

Feedback SiteMap

This page's original location: http://www.jeckle.de/webServices/index.html

RDF description for this page

INFORMATION BOARD

Middleware That Works
SEMINAR SERIES

OMG Home Vendor Commitments Standards Success Stories Getting Started

Submit success story Submit vendor commitment

DCOM

The Distributed Component Object Model (DCOM) is a protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. Previously called "Network OLE," DCOM is designed for use across multiple network transports, including Internet protocols such as HTTP. DCOM is based on the Open Software Foundation's DCE-RPC spec and will work with both Java applets and ActiveX® components through its use of the Component Object Model (COM). Follow the links below to learn more about DCOM.

Articles in the Press

A listing of media coverage on DCOM and related technologies from various publications.

White Papers

A listing of technical white papers, FAQs and other documentation on DCOM and related technologies.

Case Studies

A collection of case studies that show how customers are building solutions with DCOM.

Samples

A collection of developer samples for DCOM and related technologies.

Specs

Comprehensive documentation on DCOM.

Web Sites

A listing of other Microsoft and external developer Web sites that offer technical information, resources and training on DCOM and related technologies.

Books

A collection of noteworthy books on DCOM and related technologies.

< ! -

About Web services · Web Services Activity statement
Administrative page · Web Services CG

Web Services Architecture Working Group

The charter period of this Working Group ended in January 2004. Technical discussion of the Web Services Architecture still takes place on the www-ws-arch public mailing list.

Charter · Documents · Issues list · Meeting records · Discussion lists · Participants (IPR declarations) · Related resources

Documents

Published Documents

- Web Services Architecture (2004-02-11)

Supporting documents:

- Web Services Architecture Requirements (2004-02-11)
- Web Services Glossary (2004-02-11)
- Web Services Architecture Usage Scenarios (2004-02-11)
- Web Service Management: Service Life Cycle (2004-02-11)
- OWL ontology

Earlier publications

- Web Services Architecture Requirements (2002-11-14)
- Web Services Architecture (2003-08-08)
- Web Services Glossary (2003-08-08)
- Web Services Architecture Usage Scenarios (2003-05-14)
- Web Services Architecture (2003-05-14)
- Web Services Glossary (2003-05-14)
- Web Services Architecture (2002-11-14)
- Web Services Glossary (2002-11-14)

- Web Services Architecture Usage Scenarios (2002-07-30)
- Web Services Architecture Requirements (2002-10-11)
- Web Services Architecture Requirements (2002-08-19)
- Web Services Architecture Requirements (2002-04-29)

Issues list

The Working Group has maintained an issues list. The process used to administer the issues list is documented in the Web Services Architecture Issues Process.

Editors' copies (out of date)

Latest versions

For the very latest changes on the documents, please see the Working Group's CVS area:

- Web Services Architecture (latest version)
- Web Services Glossary (latest version)
- Web Services Architecture Usage Scenarios (latest version)
- Web Services Architecture Requirements (latest version)

See also the editorial to-do list.

Previous versions

- Web Services Architecture (2003-07-21) — see announcement
- Web Services Architecture (2003-05-01) — see announcement
- Web Services Glossary (2003-04-30) — see announcement
- Web Services Architecture (2003-03-28)
- Web Services Glossary (2003-02-14) — see announcement
- Web Services Architecture (2002-08-21)
- Web Services Architecture Requirements (2002-06-26)
- Web Services Architecture Usage Scenarios (2002-07-16)
- Web Services Glossary (2002-06-05)
- Web Services Architecture (2002-06-06)

Task forces proposals

This section gathers proposals that have been made by task forces that the Working Group created. These proposals do not necessarily represent consensus of the Working Group.

Management task force
    MTF output

Draft of Management Concern Section (2002-11-11)
Draft of proposals for details for the requester, provider and discovery agencies roles (2002-11-11)
Draft of proposal for lifecycle and state for Web services (2002-11-11)
Web Services Endpoint Management Architecture Requirements Draft (2003-02-28)

Security task force

Proposal for a security section (2003-04-24)

Discussion lists

Technical discussions of the Web Services Architecture take place on the public www-ws-arch@w3.org mailing list:

Browse the www-ws-arch archive.
Search the archive:

Search Help!

In order to subscribe to the www-ws-arch mailing list, please check the subscription procedure.

Administrative issues were discussed on the Member only mailing list w3c-ws-arch. See the administrative page (Member only) for more details.

Both mailing lists use the archive approval system.

Task force mailing lists (closed)

- Architecture Document Refactoring Task Force mailing list
- Management Task Force mailing list
- Properties and Features Task Force mailing list
- Security Task Force mailing list
- Glossary mailing list

Meeting records

Meetings in 2004:

- 2004-01-26 to 2004-01-29: Face-to-face meeting: 2004-01-26 minutes, 2004-01-27

minutes, 2004-01-28 minutes, 2004-01-29 minutes
- 2004-01-22: Teleconference
- 2004-01-15: Teleconference
- 2004-01-08: Teleconference

Meetings in 2003:

- 2003-12-18: Teleconference
- 2003-12-11: Teleconference
- 2003-12-04: Teleconference
- 2003-11-20: Teleconference
- 2003-11-13: Teleconference
- 2003-11-05 to 2003-11-07: Face-to-face meeting: 2003-11-05 minutes, 2003-11-06 minutes, 2003-11-07 minutes
- 2003-10-30: Teleconference
- 2003-10-23: Teleconference
- 2003-10-16: Teleconference
- 2003-10-09: Teleconference
- 2003-10-02: Teleconference
- 2003-09-24 to 2003-09-26: Face-to-face meeting: 2003-09-24 minutes, 2003-09-25 minutes, 2003-09-26 minutes
- 2003-09-18: Teleconference
- 2003-09-11: Teleconference
- 2003-09-04: Teleconference
- 2003-08-21: Teleconference
- 2003-08-07: Teleconference (minutes lost in a laptop crash)
- 2003-07-28 to 2003-07-30: Face-to-face meeting: 2003-07-28 minutes, 2003-07-29 minutes, 2003-07-30 minutes
- 2003-07-24: Teleconference
- 2003-07-17: Teleconference
- 2003-07-10: Teleconference
- 2003-07-03: Teleconference
- 2003-06-26: Teleconference
- 2003-06-19: Teleconference
- 2003-06-12: Teleconference
- 2003-06-05: Teleconference
- 2003-05-29: Teleconference
- 2003-05-22: Teleconference
- 2003-05-14 to 2003-05-16: Face-to-face meeting: 2003-05-14 minutes, 2003-05-15 minutes, 2003-05-16 minutes
- 2003-05-08: Teleconference

- 2003-05-01: Teleconference
- 2003-04-24: Teleconference
- 2003-04-17: Teleconference
- 2003-04-10: Teleconference
- 2003-04-03: Teleconference
- 2003-03-27: Teleconference
- 2003-03-20: Teleconference
- 2003-03-16: Teleconference
- 2003-03-06 to 2003-03-07: Face-to-face meeting: 2003-03-06 minutes, 2003-03-07 minutes
- 2003-02-27: Teleconference
- 2003-02-20: Teleconference
- 2003-02-13: Teleconference
- 2003-02-06: Teleconference
- 2003-01-30: Teleconference
- 2003-01-22 to 2003-01-24: Face-to-face meeting: Wednesday 22 January minutes, Thursday 23 January minutes, Friday 24 January minutes
- 2003-01-16: Teleconference
- 2003-01-09: Teleconference

Meetings in 2002:

- 19 December 2002: Teleconference
- 12 December 2002: Teleconference
- 5 December 2002: Teleconference
- 21 November 2002: Teleconference
- 13-15 November 2002: Face-to-face meeting: 13 November minutes, 14 November minutes, 15 November minutes
- 7 November 2002: Teleconference
- 31 October 2002: Teleconference
- 24 October 2002: Teleconference
- 17 October 2002: Teleconference
- 10 October 2002: Teleconference
- 3 October 2002: Teleconference
- 26 September 2002: Teleconference
- 19 September 2002: Teleconference
- 11-13 September 2002: Face-to-face meeting
- 5 September 2002: Teleconference
- 29 August 2002: Informal teleconference
- 22 August 2002: Informal teleconference
- 15 August 2002: Informal teleconference

- 8 August 2002: Informal teleconference
- 1 August 2002: Informal teleconference
- 25 July 2002: Teleconference
- 18 July 2002: Teleconference
- 11 July 2002: Teleconference
- 27 June 2002: Teleconference
- 27 June 2002: Teleconference
- 20 June 2002: Teleconference
- 12-14 June 2002: Face-to-face meeting
- 6 June 2002: Teleconference
- 30 May 2002: Teleconference
- 23 May 2002: Teleconference
- 16 May 2002: Teleconference
- 9 May 2002: Teleconference
- 2 May 2002: Teleconference
- 25 April 2002: Teleconference
- 18 April 2002: Teleconference
- 8-10 April 2002: Face-to-face meeting
- 4 April 2002: Teleconference
- 28 March 2002: Teleconference
- 21 March 2002: Teleconference
- 14 March 2002: Teleconference
- 7 March 2002: Teleconference
- 28 February 2002: Teleconference
- 21 February 2002: Teleconference
- 14 February 2002: Teleconference
- 6 February 2002: Teleconference

Working Group Participants

The Web Services Architecture Working Group was composed of:

List of Working Group participants

Name	Organization	IPR claims
Geoff Arnold	Sun Microsystems, Inc.	no
Mukund Balasubramanian	Infravio, Inc.	no
Mike Ballantyne	EDS	no
Abbie Barbir	Nortel Networks	no

David Booth	W3C	no
Mike Brumbelow	Apple	no
Doug Bunting	Sun Microsystems, Inc.	no
Greg Carpenter	Nokia	incomplete disclosure (see Nokia's declaration)
Tom Carroll	W. W. Grainger, Inc.	no
Alex Cheng	Ipedo	no
Michael Champion	Software AG	no (see Mike's declaration)
Martin Chapman	Oracle Corporation	no
Ugo Corda	SeeBeyond Technology Corporation	no
Roger Cutler	ChevronTexaco	no
Jonathan Dale	Fujitsu	no
Suresh Damodaran	Sterling Commerce(SBC)	no
James Davenport	MITRE Corporation	no
Paul Denning	MITRE Corporation	no
Gerald Edgar	The Boeing Company	no
Shishir Garg	France Telecom	no
Hugo Haas	W3C	no
Hao He	The Thomson Corporation	no
Dave Hollander	Contivo	no (see Dave's declaration)
Yin-Leng Husband	Hewlett-Packard Company	yes (RF) (see Hewlett-Packard's declaration)
Mario Jeckle	DaimlerChrysler Research and Technology	no
Heather Kreger	IBM	no
Sandeep Kumar	Cisco Systems Inc	no
Hal Lockhart	OASIS	no
Michael Mahan	Nokia	incomplete disclosure (see Nokia's declaration)
Francis McCabe	Fujitsu	no
Michael Mealling	VeriSign, Inc.	no

Jeff Mischkinsky	Oracle Corporation	no
Eric Newcomer	IONA	no
Mark Nottingham	BEA Systems	no
David Orchard	BEA Systems	no
Bijan Parsia	MIND Lab	no
Adinarayana Sakala	IONA	no
Waqar Sadiq	EDS	no
Igor Sedukhin	Computer Associates	no
Hans-Peter Steiert	DaimlerChrysler Research and Technology	no
Katia Sycara	Carnegie Mellon University	no (see Katya's declaration)
Bryan Thompson	Hicks & Associates, Inc.	no
Sinisa Zimek	SAP	no

Previous members of the Working Group were: Assaf Arkin (Intalio, Inc.), Daniel Austin (W. W. Grainger, Inc.), Mark Baker (Idokorro Mobile, Inc. / Planetfred, Inc.), Tom Bradford (XQRL, Inc.), Allen Brown (Microsoft Corporation), Dipto Chakravarty (Artesia Technologies), Jun Chen (MartSoft Corp.), Alan Davies (SeeBeyond Technology Corporation), Glen Daniels (Macromedia), Ayse Dilber (AT&T), Zulah Eckert (Hewlett-Packard Company), Colleen Evans (Sonic Software), Chris Ferris (IBM), Daniela Florescu (XQRL Inc.), Sharad Garg (Intel), Mark Hapner (Sun Microsystems, Inc.), Joseph Hui (Exodus/Digital Island), Michael Hui (Computer Associates), Nigel Hutchison (Software AG), Marcel Jemio (DISA), Mark Jones (AT&T), Timothy Jones (CrossWeave, Inc.), Tom Jordahl (Macromedia), Jim Knutson (IBM), Steve Lind (AT&T), Mark Little (Arjuna), Bob Lojek (Intalio, Inc.), Anne Thomas Manes (Systinet), Jens Meinkoehn (T-Nova Deutsche Telekom Innovationsgesellschaft), Nilo Mitra (Ericsson), Don Mullen (TIBCO Software, Inc.), Himagiri Mukkamala (Sybase, Inc.), Joel Munter (Intel), Henrik Frystyk Nielsen (Microsoft Corporation), Duane Nickull (XML Global Technologies), David Noor (Rogue Wave Software), Srinivas Pandrangi (Ipedo), Kevin Perkins (Compaq), Mark Potts (Talking Blocks, Inc.), Fabio Riccardi (XQRL, Inc.), Don Robertson (Documentum), Darran Rolls (Waveset Technologies, Inc.), Krishna Sankar (Cisco Systems Inc), Jim Shur (Rogue Wave Software), Patrick Thompson (Rogue Wave Software), Steve Vinoski (IONA), Scott Vorthmann (TIBCO Software, Inc.), Jim Webber (Arjuna), Prasad Yendluri (webMethods, Inc.), Jin Yu (MartSoft Corp.) .

List for the roll call and for the acknowledgments section of the specifications.

Patent disclosures

Please refer to the Web Services Architecture Working Group patent disclosures document.

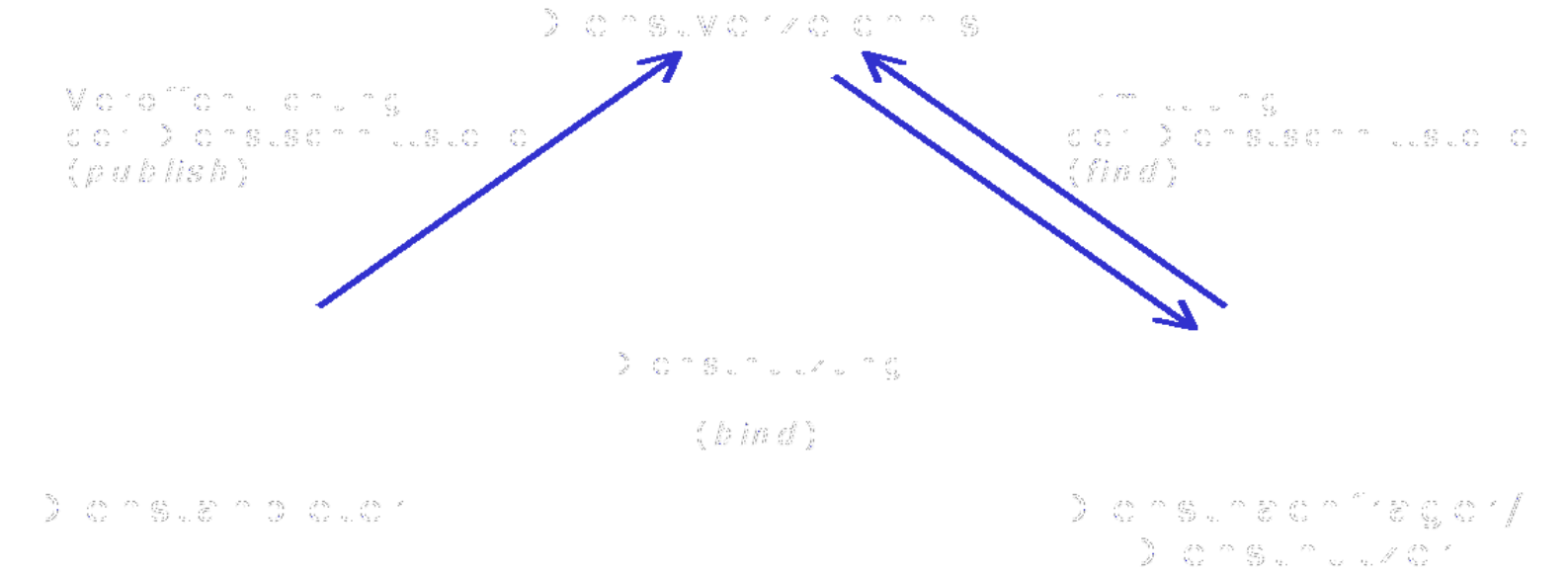
Related Resources

The following is suggested background reading:

- Architecture of the World Wide Web
- Web Architecture from 50,000 feet
- From Roy Fielding's dissertation: Designing the Web Architecture: Problems and Insights and Representational State Transfer (REST)

See also the documents produced by the TAG and their homework section.

Mike Champion <Mike.Champion@SoftwareAG-USA.com> and Dave Hollander <dmh@contivo.com>, Chairs
Hugo Haas <hugo@w3.org> and David Booth <dbooth@w3.org>, W3C Team Contacts
$Id: Overview.html,v 1.335 2004/05/06 19:51:09 dbooth Exp $


(publish)
(find)
(bind)

Simple cross-platform distributed computing, based on the standards of the Internet.

Home

Spec

News

Mail List

Directory

Discuss

New Topic

HowTo

Top 50

Referers

SOAP

RSS

OPML

XML

Dave

XML-RPC Home Page

"Does distributed computing have to be any harder than this? I don't think so." -- Byte.

What is XML-RPC?

It's a spec and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet.

It's remote procedure calling using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.

XML-RPC Directory

- Specifications
- Implementations
- Services
- Blogging APIs
- Communities
- Tutorials/Press

The XML-RPC community

The implementations page lists the accomplishments of the community, a set of compatible XML-RPC implementations that span all operating systems, programming languages, dynamic and static environments, open source and commercial, for Perl, Python, Java, Frontier, C/C++, Lisp, PHP, Microsoft .NET, Rebol, Real Basic, Tcl, Delphi, WebObjects and Zope, and more are coming all the time.

The services page lists the next-level-up, public applications, or "web services" that are accessible through XML-RPC.

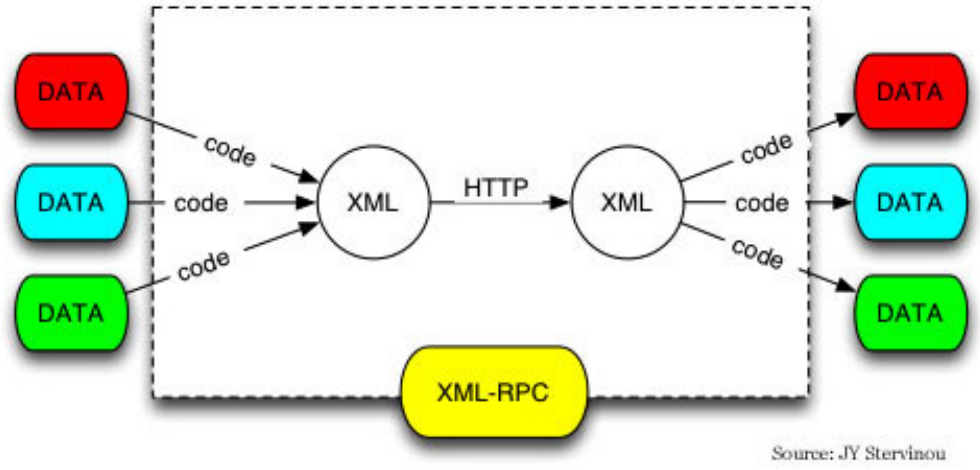
The communities page tries to organize all the activity around XML-RPC on mail lists, websites and search engines.

Finally, the tutorials/press page points to articles written about XML-RPC.

Members

Join Now

Login



How to participate

Read Eric Kidd's fantastic XML-RPC HowTo.

Join the mail list, or become a member of this site, and post an introduction or ask a question.

Each page with links in this directory has an easy Suggest-A-Link feature, look towards the bottom of the page, click the link, fill in the form, click on Submit.

Test your implementation on the XML-RPC Validator page.

See who's linking to us on the Referers page.

Lest anyone forget

The first implementation of XML-RPC was in Frontier, in April 1998.

Web Services

XML Protocol Working Group

Deliverables · Membership · Charter/History
Related Resources (Technical - Administrative (member only))
· Mailing List archive · XML CG

The XML Protocol WG is part of the Web Services Activity.

The WG holds a regular teleconference, for more details, see the administrative page (member only).

The WG's discussions take place on the xml-dist-app@w3.org mailing list (Archived). Administrative and member-confidential topics are discussed on the w3c-xml-protocol-wg@w3.org mailing list (Archived)

To subscribe to the public mailing list, send an email to xml-dist-app-request@w3.org with "subscribe" in the subject, to unsubscribe from this list, send an email to xml-dist-app-request@w3.org with "unsubscribe" in the subject.

To become a WG member, ask your AC representative to fill this form (W3C member only)

Milestones/Deliverables

Initially, per the charter. Changes to the schedule will be negotiated with the relevant parties via the XML CG. The Working Group welcomes comments on the xml-dist-app mailing list (archive).

- **Feedback on SOAP 1.2 Recommendation** (generated from XML) - *This list is for Recommendation documents.*
- **Proposed Recommendation Issues List** (generated from XML using XSL) - *This issues list is for documents with Proposed Recommendations status.*
- **Implementation Summary** - The current state of implementation efforts.
- **Candidate Recommendation Issues List** (generated from XML using XSL) - *This issues list is for all Candidate Recommendations.*
- **Last Call Issues List** (generated from XML using XSL) - *This issues list is for all documents in Last Call.*

- **Issues List** (generated from XML using XSL) - *This issues list is for documents that are not in Last Call.*

Recommendations

- SOAP Version 1.2 Part 0: Primer
- SOAP Version 1.2 Part 1: Messaging Framework
- SOAP Version 1.2 Part 2: Adjuncts
- SOAP Version 1.2 Specification Assertions and Test Collection

Documents in progress

Scheduled Milestone	Document	Revision Date
XML Protocol Requirements	**WD:** XML Protocol (XMLP) Requirements	2002-06-26
	Editors' latest draft: *not for review!* • Editors' copy (XML version)	-
XML Protocol Usage Scenarios	**WD:** XML Protocol Usage Scenarios	2002-06-26
	Editors' latest draft: *not for review!* • Editors' copy (XML version)	-
XML Protocol Abstract Model	**WD:** XML Protocol Abstract Model (announcement)	2003-02-20
SOAP Version 1.2 spec	**REC:** SOAP Version 1.2 Part 1: Messaging Framework	2003-06-24
	REC: SOAP Version 1.2 Part 2: Adjuncts	2003-06-24
	Editors' latest draft: *not for review!* • Editors' copy of Part 1 (XML version) • Editors' copy of Part 2 (XML version) • Editors To-Do List	-
SOAP Version 1.2 Primer	**REC:** SOAP Version 1.2 Part 0: Primer	2003-06-24

	Editors' latest draft: • Editors' copy of Part 0	-
SOAP Version 1.2 Specification Assertions and Test Collection	**REC:** SOAP Version 1.2 Specification Assertions and Test Collection	2003-06-24
	Call for contributions	-
	Editor's latest draft: • Editors' copy of Test Collection	-
SOAP 1.2 Attachment Feature	**WD:** SOAP 1.2 Attachment Feature	2002-09-24
	Editors' latest draft: *not for review!* • Editors' copy of Attachment Feature	-
SOAP Optimized Serialization Use Cases and Requirements	**WD:** SOAP Optimized Serialization Use Cases and Requirements	2003-11-12
	Editors' latest draft: *not for review!* • Editors' copy of SOAP Optimized Serialization Use Cases and Requirements	-
SOAP Message Transmission Optimization Mechanism	**WD:** SOAP Message Transmission Optimization Mechanism	2004-02-09
	Editors' latest draft: *not for review!* • Editors' copy of SOAP Message Transmission Optimization Mechanism	-
XML-binary Optimized Packaging	**WD:** XML-binary Optimized Packaging	2004-02-09
	Editors' latest draft: *not for review!* • Editors' copy of XML-binary Optimized Packaging	-
	WD: SOAP Resource Representation Header	2004-04-28

SOAP Resource Representation Header	**Editors' latest draft:** *not for review!* • Editors' copy of XML-binary Optimized Packaging	-
SOAP Version 1.2 Email Binding	**Note** SOAP Version 1.2 Email Binding	2002-06-26
SOAP Version 1.2 Message Normalization	**Note** SOAP Version 1.2 Message Normalization	2003-10-08
The "application/soap+xml" media type	• The"application/soap+xml" media type, Internet Draft, 29 May, 2003	

Below are listed other documents produced by the working group:

Other Documents	Document	Revision Date	Status/History
Terminology Section	Now included in the SOAP 1.2 spec	-	-
Transport binding framework & bindings	SOAP Underlying Protocol Binding Framework	2001-10-16	Editors' copy
	Default HTTP binding	2001-10-11	Editors' copy
	Transport Message Exchange Pattern: Single-Request-Response	2001-10-11	Editors' copy

For patents information about those documents, please see the XMLP WG patent disclosure statements.

Translations

Note that those documents are not produced by the Working Group, but done by third parties. The authoritative version is **always** the original publication made by the Working Group.

Document	Language	Original	Translator	Date
SOAP Version 1.2, partie 0: Préliminaire	French	Rec: SOAP Version 1.2 Part 0: Primer	Carine Bournez	24 June 2003

SOAP Version 1.2 Partie 1: Structure pour l'échange de messages	French	REC: SOAP Version 1.2 Part 1: Messaging Framework	Carine Bournez	24 June 2003
SOAP Version 1.2 Partie 2: Ajouts	French	PR: SOAP Version 1.2 Part 2: Adjuncts	Carine Bournez	24 June 2003

Charter and History

The XML Protocol Working Group charter has a specific timeline, which is reported below. When more precise events will be scheduled, they will be included, together with a status report and meeting minutes, so to indicate historical progress of the Working Group.

The charter has been updated in february 2004, here is the list of previous charters

- Revised Charter - Oct 2002
- Initial Charter - Sep 2000

Meeting records of subgroups are also available:

- Abstract Model Group
- Transport Binding Task Force
- RPC Task Force
- Encoding Task Force

Below is a list of the Working Group's meeting minutes:

April 21, 2004
Teleconference - minutes
April 14, 2004
Teleconference - minutes
March 31, 2004
Teleconference - minutes
March 24, 2004
Teleconference - minutes
March 17, 2004
Teleconference - minutes
March 10, 2004
Teleconference - minutes
March 1-2, 2003
Face to face meeting, Mandelieu, France. - minutes
February 25, 2004

Teleconference - minutes

February 18, 2004

Teleconference - minutes

February 11, 2004

Teleconference - minutes

February 4, 2004

Teleconference - minutes

January 28, 2004

Teleconference - minutes

January 21, 2004

Teleconference - minutes

January 14, 2004

Teleconference - minutes

January 7, 2004

Teleconference - minutes

December 17, 2003

Teleconference - minutes

December 10, 2003

Teleconference - minutes

December 2-3, 2003

Face to face meeting, San Francisco, CA, USA. - minutes

November 19, 2003

Teleconference - minutes

November 12, 2003

Teleconference - minutes

November 5, 2003

Teleconference - minutes

October 29, 2003

Teleconference - minutes

October 22, 2003

Teleconference - minutes

October 15, 2003

Teleconference - minutes

October 8, 2003

Teleconference - minutes

October 1, 2003

Teleconference - minutes

September 24, 2003

Teleconference - minutes

September 17, 2003

Teleconference - minutes

September 10, 2003

Teleconference - minutes

September 3, 2003
Teleconference - minutes
July 24-25, 2003
Face to face meeting, Sophia-Antipolis, France. - minutes
July 16, 2003
Teleconference - minutes
July 9, 2003
Teleconference - minutes
July 2, 2003
Teleconference - minutes
June 25, 2003
Teleconference - minutes
June 18, 2003
Teleconference - minutes
June 11, 2003
Teleconference - minutes
June 4, 2003
Teleconference - minutes
May 28, 2003
Teleconference - minutes
May 21, 2003
Teleconference - minutes
May 14, 2003
Teleconference - minutes
May 7, 2003
Teleconference - minutes
April 30, 2003
Teleconference - minutes
April 23, 2003
Teleconference - minutes
April 16, 2003
Teleconference - minutes
April 9, 2003
Teleconference - minutes
April 2, 2003
Teleconference - minutes
March 26, 2003
Teleconference - minutes
March 19, 2003
Teleconference - minutes
March 12, 2003
Teleconference - minutes
March 6-7, 2003

Face to face meeting, Boston, MA. - minutes

February 19, 2003

Teleconference - minutes

February 11-12, 2003

CR Teleconference - minutes

February 5, 2003

Teleconference - minutes

January 29, 2003

Teleconference - minutes

January 22, 2003

Teleconference - minutes

January 15, 2003

Teleconference - minutes

January 8, 2003

Teleconference - minutes

December 18, 2002

Teleconference - minutes

December 11, 2002

Teleconference - minutes

December 4, 2002

Teleconference - minutes

November 27, 2002

Teleconference - minutes

November 20, 2002

Teleconference - minutes

November 6, 2002

Teleconference - minutes

October 29, 2002

Face to face meeting, Berford, MA, USA - minutes

October 23, 2002

Teleconference - minutes

October 16, 2002

Teleconference - minutes

October 9, 2002

Teleconference - minutes

October 2, 2002

Teleconference - minutes

September 25, 2002

Teleconference - minutes

September 18, 2002

Teleconference - minutes

September 11, 2002

Teleconference - minutes

September 4, 2002
Teleconference - minutes
July 30 - Aug 1, 2002
Face to face meeting, Palo Alto, Ca, USA - minutes day 1, day 2 and day 3.
July 24, 2002
Teleconference - minutes
July 17, 2002
Teleconference - minutes
July 10, 2002
Teleconference - minutes
July 3, 2002
Teleconference - minutes
June 26, 2002
Teleconference - minutes
June 19, 2002
Teleconference - minutes
June 12, 2002
Teleconference - minutes
June 5, 2002
Teleconference - minutes
May 29, 2002
Teleconference - minutes
May 22, 2002
Teleconference - minutes
May 15, 2002
Teleconference - minutes
May 8, 2002
Teleconference - minutes
May 1, 2002
Teleconference - minutes
April 24, 2002
Teleconference - minutes
April 17, 2002
Teleconference - minutes
April 10, 2002
Teleconference - minutes
April 03, 2002
Teleconference - minutes
March 27, 2002
Teleconference - minutes
March 20, 2002
Teleconference - minutes
March 13, 2002

Teleconference - minutes

March 06, 2002

Teleconference - minutes

February 25-26, 2002

Face to face meeting, Mandelieu, France - minutes

February 20, 2002

Teleconference - minutes

February 13, 2002

Teleconference - minutes

February 06, 2002

Teleconference - minutes

January 30, 2002

Teleconference - minutes

January 23, 2002

Teleconference - minutes

January 16, 2002

Teleconference - minutes

January 9, 2002

Teleconference - minutes

December 19, 2001

Teleconference - minutes

December 12, 2001

Teleconference - minutes

December 5, 2001

Teleconference - minutes

November 27-28, 2001

Face to face meeting, Burlington, Mass, USA - minutes

November 21, 2001

Teleconference - minutes

November 14, 2001

Teleconference - minutes

November 7, 2001

Teleconference - minutes

October 31, 2001

Teleconference - minutes

October 24, 2001

Teleconference - minutes

October 17, 2001

Teleconference - minutes

October 10, 2001

Teleconference - minutes

October 3, 2001

Teleconference - minutes

September 26, 2001
Teleconference - minutes
September 19, 2001
Teleconference - minutes
September 11-13, 2001
Face to face meeting, San Jose, CA, USA - minutes
September 5, 2001
Teleconference - minutes
August 29, 2001
Teleconference - minutes
August 22, 2001
Teleconference - minutes
August 15, 2001
Teleconference - minutes
August 08, 2001
Teleconference - minutes
August 01, 2001
Teleconference - minutes
July 25, 2001
Teleconference - minutes
July 18, 2001
Teleconference - minutes
July 11, 2001
Teleconference - minutes
June 27, 2001
Teleconference - minutes
June 20, 2001
Teleconference - minutes
June 6-13, 2001
No teleconferences
June 5-7, 2001
Face to face meeting, Dinard, France - minutes
May 30, 2001
Teleconference - minutes
May 23, 2001
Teleconference - minutes
May 16, 2001
Teleconference - minutes
May 9, 2001
Teleconference - minutes
May 2, 2001
Teleconference - minutes
April 25, 2001
Teleconference - minutes

April 18, 2001
Teleconference - minutes
April 11, 2001
Teleconference - minutes
April 4, 2001
Teleconference - minutes
March 28, 2001
Teleconference - minutes
March 21, 2001
Teleconference - minutes
March 14, 2001
Teleconference - minutes
March 7, 2001
Teleconference - minutes
February 26-27, 2001
Face to face meeting, Cambridge, MA, USA - minutes
February 21, 2001
Teleconference - minutes
February 14, 2001
Teleconference - minutes
February 7, 2001
Teleconference - minutes
January 31, 2001
Teleconference - minutes
January 24, 2001
Teleconference - minutes
January 17, 2001
Teleconference - minutes
January 10, 2001
Teleconference - minutes
January 3, 2001
Teleconference - minutes
December 20, 2000
Teleconference - minutes
December 13-14, 2000
Face to face meeting, Redmond WA. USA - minutes
December 6, 2000
Teleconference - minutes
November 29, 2000
Teleconference - minutes
November 22, 2000
No teleconference
November 15, 2000
Teleconference - minutes

November 8, 2000
Teleconference - minutes
November 1, 2000
Teleconference - minutes
October 25, 2000
Teleconference - minutes
October 18, 2000
Teleconference - minutes
October 11-12, 2000
Face to face meeting, Raleigh NC. USA - minutes
September 13, 2000
Creation of the XML Protocol Working Group

Membership

See How to join a W3C Working Group if you want to become a member of this Working Group.

The working group currently consists of the following ***17 members*** representing ***10 organisations*** (excluding chair and W3C staff), and ***0 expert/observer***:

Name	Organization	Role	IPR Claims
David Fallside[intro]	IBM	chair	See
Tony Graham	Sun Microsystems	alternate	no
Martin Gudgin[intro]	Microsoft Corporation	principal	See
Marc Hadley	Sun Microsystems	principal	no
Gerd Hoelzing	SAP AG	alternate	no
John Ibbotson[intro]	IBM	principal	See
Anish Karmarkar	Oracle	principal	no
Suresh Kodichath	IONA Technologies	principal	See
Yves Lafon	W3C	team contact	no
Michael Mahan	Nokia	principal	See
Noah Mendelsohn	IBM	alternate	See
Jeff Mischkinsky	Oracle	alternate	no
Jean-Jacques Moreau	Canon	principal	no
Mark Nottingham	BEA Systems	alternate	no

David Orchard	BEA Systems	principal	no
Herve Ruellan	Canon	alternate	no
Jeff Schlimmer	Microsoft Corporation	alternate	See
Pete Wenzel	SeeBeyond	principal	See
Volker Wiechers[intro]	SAP AG	principal	no

Invited Experts/Observers:

none

The list of IPR statements for current and past WG members has been moved to a separate page

Previous members: Richard Martin (Active Data Exchange), Eric Fedok (Active Data Exchange), Shane Sesta (Active Data Exchange), Susan Yee (Active Data Exchange), Mark Nottingham (Akamai Technologies), Mark Jones (AT&T), Michah Lerner (AT&T), Jags Ramnaryan (BEA Systems), Dan Frantz (BEA Systems), Alex Ceponkus (Bowstreet), James Tauber (Bowstreet), Rekha Nagarajan (Calico Commerce), Mary Holstege (Calico Commerce), Krishna Sankar (Cisco Systems), Raj Nair (Cisco Systems), David Burdett (Commerce One), Murray Maloney (Commerce One), Jay Kasi (Commerce One), Yin-Leng Husband (Compaq), Kevin Perkins (Compaq), Mario Jeckle (DaimlerChrysler Research and Technology), Andreas Riegg (DaimlerChrysler Research and Technology), Mark Needleman (Data Research Associates), Yan Xu (DataChannel), Brian Eisenberg (DataChannel), Mike Dierken (DataChannel), Don Box (DevelopMentor), Martin Gudgin (DevelopMentor), Aaron Skonnard (DevelopMentor), Philippe Bedu (EDF (Electricite De France)), Olivier Boudeville (EDF (Electricite De France)), Jeffrey Kay (Engenia Software), Eric Jenkins (Engenia Software), Michael Freeman (Engenia Software), Bjoern Heckel (Epicentric), Dean Moses (Epicentric), Julian Kumar (Epicentric), Miles Chaston (Epicentric), Alan Kropp (Epicentric), Scott Golubock (Epicentric), Nilo Mitra (Ericsson), Jim d'Augustine (Excelon Corporation), Kazunori Iwasa (Fujitsu Limited), Masahiko Narita (Fujitsu Limited), Jim Hughes (Fujitsu Limited), Dick Brooks (Group 8760), David Ezell (Hewlett Packard), Stuart Williams (Hewlett Packard), Yin-Leng Husband (Hewlett Packard), Fransisco Cubera (IBM), Doug Davis (IBM), Mark Baker (Idokorro Mobile, Inc.), Charles Campbell (Informix Software), Soumitro Tagore (Informix Software), Bob Lojek (Intalio Inc.), Randy Hall (Intel), Highland Mary Mountain (Intel), Brad Lund (Intel), Mark Hale (Interwoven), Ron Daniel (Interwoven), Oisin Hurley (IONA Technologies), Eric Newcomer (IONA Technologies), Seumas Soltysik (IONA Technologies), Mike Greenberg (IONA Technologies), Roberto Castaldo (IWA / HWG), David Orchard (Jamcracker), Alex Milowski (Lexica), Ray Denenberg (Library of Congress), Rich Greenfield (Library of Congress), Noah Mendelsohn (Lotus Development), Glen Daniels (Macromedia), Simeon Simeonov (Macromedia), Jin Yu (MartSoft Corp.), Ryuji Inoue (Matsushita Electric Industrial Co., Ltd.), Steve Hole (MessagingDirect Ltd.), John-Paul Sicotte (MessagingDirect Ltd.), Paul Cotton (Microsoft Corporation), Henrik Nielsen (Microsoft Corporation), Don Box (Microsoft Corporation), Marwan Sabbouh (MITRE Corporation), Paul Denning (MITRE Corporation), Vilhelm Rosenqvist (NCR), Lew Shannon (NCR), Vidur Apparao (Netscape), Ray Whitmer (Netscape), Scott Isaacson (Novell, Inc.), Art Nevarez (Novell, Inc.), Winston Bumpus (Novell, Inc.), Henry Lowe (OMG), Fred Waskiewicz (OMG), David Clay (Oracle), Jim Trezzo (Oracle), Yasser alSafadi (Philips Research), Amr Yassin (Philips Research), David Cleary (Progress Software), Andrew Eisenberg (Progress Software), Peter Lecuyer (Progress Software), Daniela Florescu (Propel), Murali Janakiraman (Rogue Wave), Patrick Thompson (Rogue Wave), Michael Champion (Software AG), Dietmar Gaertner (Software AG), Colleen Evans (Sonic Software), David Chappell (Sonic Software), Ed Mooney (Sun Microsystems), Mark Baker (Sun Microsystems), Anne Thomas Manes (Sun Microsystems), Chris Ferris (Sun Microsystems), Jacek Kopecky (Systinet), Miroslav Simek (Systinet), Frank DeRose (TIBCO Software, Inc.), James Falek (TIBCO Software, Inc.), Don Mullen (TIBCO Software, Inc.), George Scott (Tradia Inc.), Erin Hoffmann (Tradia Inc.), Nick Smilonich (Unisys), Lynne Thompson (Unisys), Conleth O'Connell (Vignette), Richard Koo (Vitria Technology Inc.), Waqar Sadiq (Vitria Technology Inc.), Tony Lee (Vitria Technology Inc.), Hugo Haas (W3C), Carine Bournez (W3C), Randy Waldrop (WebMethods), Dave Cleary

(webMethods), Asir Vedamuthu (webMethods), Camilo Arbelaez (webMethods), Bill Anderson (Xerox), Tom Breuel (Xerox), Ugo Corda (Xerox), Matthew MacKenzie (XMLGlobal Technologies), David Webber (XMLGlobal Technologies), John Evdemon (XMLSolutions), Kevin Mitchell (XMLSolutions), Rich Salz (Zolera Systems), Frederick Hirsch (Zolera Systems)

Previous observers: none.

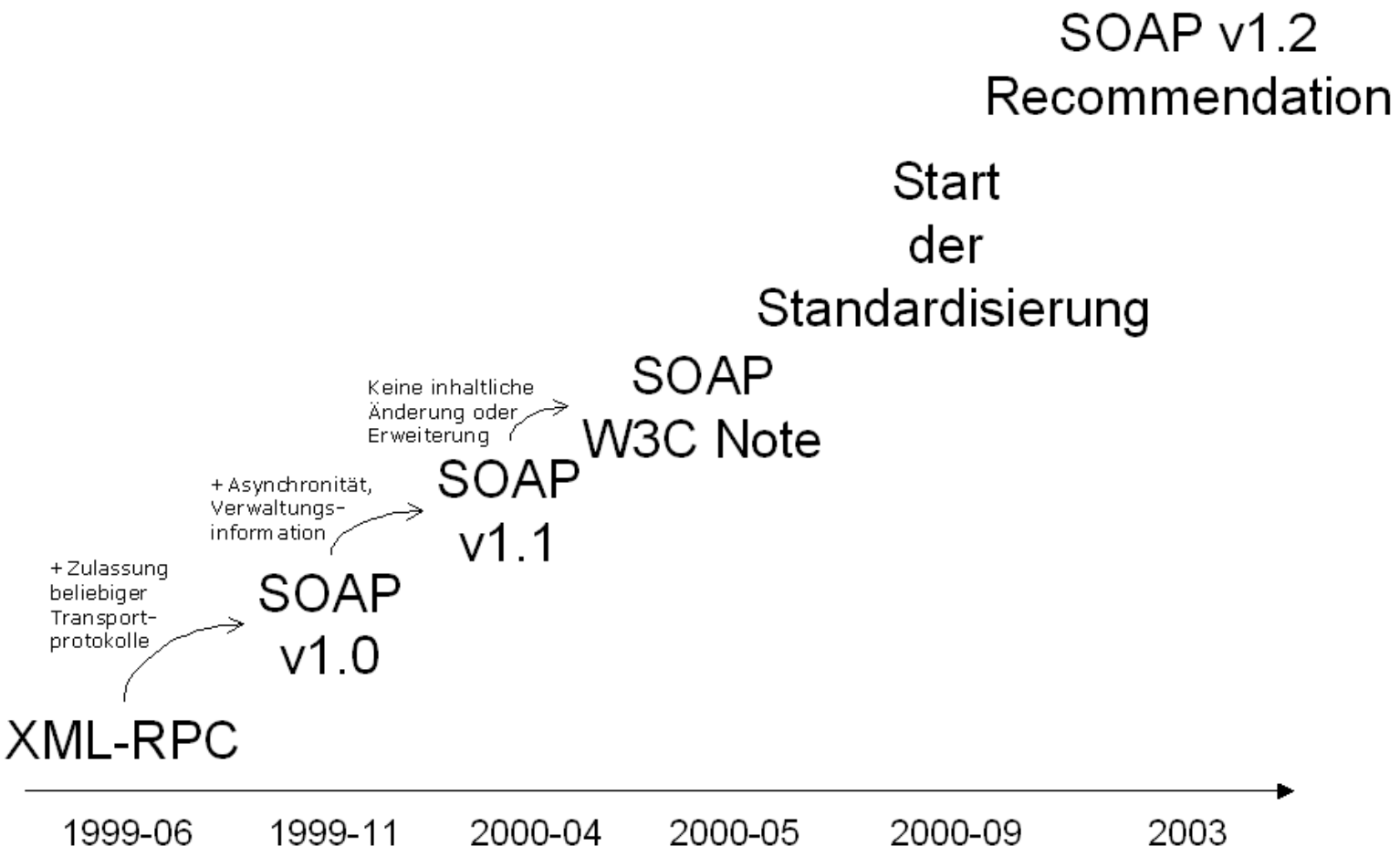
Related Resources

Technical

- Web Architecture from 50,000 feet
- XML Protocol Matrix

David Fallside, Yves Lafon

Revision: $Date: 2004/04/29 12:45:36 $


SOAP v1.2
Recommendation
Start
der
Standardisierung
SOAP
W3C Note
Keine inhaltliche
Änderung oder
Erweiterung
SOAP
v1.1
+ Asynchronität,
Verwaltungs-
information
SOAP
v1.0
+ Zulassung
beliebiger
Transport-
protokolle
XML-RPC
1999-06
1999-11
2000-04
2000-05
2000-09
2003

```
public class Math {
	public int add(int a, int b) {
		return a+b;
	}
}
```

```
import java.rmi.RemoteException;
import javax.xml.rpc.ParameterMode;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;

public class CallAdd {
	public static void main(String[] args) {
		Call call = new Call(new Service());

		call.setTargetEndpointAddress("http://10.0.0.1:8080/axis/Math.jws");
		call.setOperationName("add");
		call.addParameter("a", XMLType.XSD_INT, ParameterMode.IN);
		call.addParameter("b", XMLType.XSD_INT, ParameterMode.IN);
		call.setReturnType(XMLType.XSD_INT);

		Object[] param = new Object[2];
		param[0] = new Integer(args[0]);
		param[1] = new Integer(args[1]);

		try {
			Object result = call.invoke(param);
			System.out.println("result="+result);
		} catch (RemoteException re) {
			re.printStackTrace();
		}
	}
}
```

1 Dokumente und Daten...

Das Akronym *XML* ist derzeit in aller Munde...
Es hat Einzug in fast jede Produktankündigung jüngerer Zeit gefunden, und keine neuere IT-Entwicklung, die nicht *XML enabled* wäre.

Es scheint, daß sich anhand XML dieselbe übersteigerte (und nicht notwendigerweise immer mit wahrheitsgemäßen Versprechungen arbeitende) Marketing-Euphorie (engl. *hype*) vollzieht, die bereits für das Schlagwort *Objektorientierung* zu beobachten war.
Die fachlichen Einschätzungen des Themas XML reichen von *ASCII des 21 Jahrhunderts* (H. S. Thomson) bis hin zur (berechtigten) kritischen Hinterfragung des Neuheitswertes, insbesondere im Vergleich zu bekannten Lösungen wie troff, LaTeX oder das *Rich Text Format* (RTF).

Zur Illustration der Bandbreite der Bedeutungs- und Anwendungszuschreibungen die XML zuteil werden sind nachfolgend einige willkürlich ausgewählte Pressezitate versammelt:

- „XML schickt sich an in die Fußstapfen von HTML zu treten“
  Aus: c't 10/2000, p. 200
- „Das Datenformat [XML] erleichtert den Informationsaustausch zwischen vernetzten Computern“
  Aus: DER SPIEGEL, 2000-06-19
- „Das XML-Format, [...] das richtige Werkzeug zur Herstellung eigener Webinhalte“
  Aus: DER SPIEGEL, 2000-06-19
- „Alle Dokumente sind gleich“
  Aus: SZ, 1999-02-16
- „Nachfolger für ungeliebte Cookies [...] Enge Verbindung von Java mit XML, [...] Erweiterung des HTML-Standards“
  Aus: DER SPIEGEL, 1999-10-05
- „Ein digitales Esperanto für das Internet“
  Aus: Die Welt, 2000-10-07
- „Sortieren statt Stottern -- Programmiersprache HTML stößt an ihre Grenzen XML ist kommender Code im Netz“
  Aus: SZ, 2000-01-11
- „Die Extended Markup Language [sic!] für eCommerce“
  Aus: F.A.Z.
- „Sinnliche Suchmaschine [...] existierende Systeme wie [...] XML “
  Aus: DER SPIEGEL, 2000-06-07
- „XML: Abkürzung für Extensible Markup Language, eine neue Sprache für Seiten im World Wide Web. XML ist

deutlich flexibler als das bisherige HTML und bietet Programmierern mehr Funktionen. “
Aus: Walsroder Zeitung, 2001-03-19

- „Vom *zugänglichen* XML [...] im Web kaum noch was zu sehen [...]“
  Aus: c't 18/2001
- „Streit um Programmiersprache XML“
  Aus: F.A.Z., 2001-07-26
- „Netzwerktechnik versagt oft bei Angriffen via XML“
  Aus: Computerzeitung, 2002-08-26

Schon die Spannweite dieser Definitionen läßt die Bedeutung des Themas „XML“ erahnen. Hervorzuheben ist, daß sich offensichtlich nicht nur technisch-fokussierte Blätter des dreibuchstabigen Akronyms annehmen.
Allgemein scheint XML vorzugsweise im Kontext World Wide Web (manchmal auch schlicht zum „Internet“ verallgemeinert) Beachtung zu finden.

Das erste Kapitel dieser Vorlesung bietet einen Einstieg in die XML-Welt anhand der zugrundeliegenden technischen Basiskonzepte.
Hierzu wird zunächst die Historie strukturierter Auszeichnungssprachen (engl. *markup language*) beleuchtet, und die Vorläufer der Markup-Sprache XML vorgestellt.
Desweiteren werden die grundlegenden syntaktischen Konstrukte des XML Standards anhand verschiedener Beispiele eingeführt. Die notwendige Begriffsbildung erfolgt auf Basis des *XML Information Set*.
Ein weiteres Teilkapitel motiviert die Notwendigkeit der XML-Namensräume zur Unterscheidung verschiedener XML-Vokabulare und als Voraussetzung der XML-basierten Inhaltssyndikatisierung.
Daran schließt sich die Betrachtung der Grammatik eines XML-Dokuments an. Hierzu wird der klassischen Mechanismus der *Document Type Definitions* (DTD) eingeführt. Schwerpunkt liegt jedoch auf der Diskussion der *XML Schema Description Language* (XSD), durch welche die strukturellen und inhaltlichen Ausdrucksmöglichkeiten der DTDs entscheidend erweitert werden.
Zum Abschluß des Kapitels wird mit der *XML Hyper Text Markup Language* (XHTML) die vermutlich bekannteste Auszeichnungssprache, welche zur Grundlage des World Wide Webs wurde, unter dem Blickwinkel XML besprochen.

1.1 Einführung und Überblick

Im Grunde besitzt die Geschichte der eXtensible Markup Language zwei Anfänge. Einerseits stellt XML die evolutionäre Fortentwicklung existierender generischer Auszeichnungssprachen dar; andererseits sind die Hintergründe der Sprache XML so eng mit dem Aufkommen des World Wide Webs (WWW) verwoben, daß die Geschichte auch hier ihren Anfang nehmen könnte...

Der chronologischen Ordnung folgend sei zunächst die Entwicklung aus der Idee des *Hypertext* aufgerissen.
Die ersten Ideen zum Konzept des Hypertexts, als Plan zur Überwindung der Beschränkungen und Unzulänglichkeiten des klassischen textbasierten Publikationsmediums Papier, datieren zurück bis in die 1950er Jahre. Sie postulieren neben der nichtsequentiellen Organisation des Mediums auch zentrale Begriffe wie *Knoten*, *Link*, *Anker* und *Netz*. Ziel dieser Überlegungen war es, den auszudrückenden Inhalt von editorieller- und Präsentationsinformation wie Seitenzahlen, Fußnoten, Paginierung usw. zu trennen. Durch die nichtlineare Organisation soll es dem Leser freigestellt werden, auf welchen Pfaden er sich durch das Dokument bewegt.

Zur Realisierung dieser Bemühungen wird das Dokument mit weiteren Informationen angereichert, die jedoch für den Leser unsichtbar bleiben. Dieser Gedanke reicht zurück bis in die Anfänge des Buchdrucks. Dort sind formatierungsorientierte Auszeichnungssymbole, etwa für Fettdruck oder Unterstreichung, seit jeher bekannt. Vor dem Aufkommen der *what you see is what you get* Textverarbeitungssysteme waren diese bildlichen Symbole die einzige Möglichkeit zur Kommunikation präsentationsorientierter Information an den Schriftsetzer und Drucker.
Jedem Schüler ist bereits ein weiteres Beispiel einer editoriellen Auszeichnungssprache bekannt: Die graphischen Korrekturzeichen der Deutschlehrer. Auch sie liefern Informationen über den Inhalt, die nicht Bestandteil des Dokuments sind.

Voraussetzung für die angestrebte Flexibilisierung der Struktur eines Textes ist eine -- wie auch immer geartete -- technische Unterstützung. Seit den 60er Jahren wurden hierfür die aufkommenden elektronischen Rechenanlagen herangezogen. Eine der ersten Aktivitäten hierzu ist das von Ted Nelson initiierte (inzwischen legendäre) Xanadu-Projekt.

Zunächst erforderte die maschinelle Verarbeitung die Überarbeitung des Auszeichnungssymbolvorrates. Dies wurde notwendig, da eingesetzte Technik keine Unterstützung der alt-hergebrachten graphischen Auszeichungssymbole bot.
In einem ersten Entwicklungsschritt wurden daher die vormalig bildhaften Zeichen durch textuelle Pendants ersetzt und verallgemeinert. Beispielsweise: `Überschrift` zur inhaltlichen Kennzeichnung einer entsprechenden Textzeile.
Mit diesem Schritt erfolgte auch der Übergang zur formatierungsunabhängigen Auszeichnung, die bewußt auf die Beschreibung des späteren visuellen Aussehens der Information zugunsten einer neutralen deskriptiven Beschreibung der Semantik verzichtete.

In den 60er und 70er Jahren werden verschiedene Weiterentwicklungen der generischen Auszeichnungssprachen betrieben; u.a. bei der IBM durch das Team um Goldfarb, Mosher und Loire. Sie stellen 1969 unter dem Namen *Generalized Markup Language* einen Sprachvorschlag zusammen, der in der Folgezeit durch IBM kommerziell vermarktet wird.

Aus den GML-Aktivitäten bei IBM entwickelt sich die internationale Standardisierungsbewegung der *Standard GML* (SGML).
Durch sie wird eine Sprache festgelegt, welche die Definition eigener Sprachen erlaubt; daher auch der Begriff *Metasprache*. SGML bietet somit keinen feststehenden problemspezifischen Sprachumfang an, sondern eine Menge verschiedenster struktureller Konstrukte zur Formulierung von Dokumentgrammatiken.
In der Praxis wird der Einsatz einer mit Hilfe von SGML definierten Sprache oftmals plakativ zum *Einsatz von SGML*

verkürzt, obwohl diese Begrifflichkeit lediglich den Erstellungsprozeß der Grammatik bezeichnet.

Mittels SGML definiert Tim Berners-Lee Mitte der 80er Jahre eine eigene Sprache zur vereinfachten Formulierung von Dokumenten, die er *HyperText Markup Language* (HTML) nennt. Hauptbeweggrund seiner Aktivitäten ist der Versuch den Dokumentenaustausch am Europäischen Kernforschungszentrum CERN rechnergestützt zu vereinfachen.
Die Eingangs erwähnten zentralen Hypertextkonzepte finden sich bereits in seinem ersten Sprachvorschlag wieder. Zur technischen Realisierung der Verknüpfung zwischen den Dokumenten mittels Ankern und Links definiert er den *Uniform Resource Locator* (URL), eine global eindeutige Adresse für beliebige Inhalte.

Seine Aktivitäten in Genf bilden die Keimzelle des Web.

In der Folgezeit, insbesondere im Zuge der Kommerzialisierung des Word Wide Web, entstehen verschiedene Revisionen der ursprünglichen HTML. Einige der Erweiterungen werden durch die beiden großen Web Browser Hersteller Microsoft und Netscape proprietär vorgenommen, um ihre Position am Markt zu stärken.
In der Konsequenz entstehen während des oft apostrophierten *browser war* teilweise inkompatible HTML-Dialekte. (Man denke nur an die Tags: `marquee` (nur Microsoft Internet Explorer) oder `layer` (nur Netscape Navigator))
Darüberhinaus entwickelt sich HTML zunehmend von einer Präsentations-orientierten Auszeichnungssprache zu einer semantischen. Dies bedeutet: während HTML in der ersten Grundform zunächst überwiegend Elemente bot, durch die die Präsentation der Inhalte am Bildschirm festgelegt wurde (Beispiele: `b` für Fettdruck, `u` für Unterstreichungen oder `i` für Kursivschreibung), wurden später zunehmend semantische Elemente eingeführt. Durch sie wird die Bedeutung der ausgezeichneten Information ausgedrückt (Beispiele hierfür: `acronym` zur Kennzeichnung von Abkürzungen, `address` für Adressen oder `strong` zur besonderen Betonung einer Textpassage).

So wünschenswert die sukzessive Umgestaltung der HTML an die veränderten Bedürfnisse war, so aussichtslos waren die Bemühungen dennoch. Während bei den Präsentations-orientierten Elementen zunehmend Vollständigkeit hinsichtlich der Anwenderwünsche erzielt werden konnte, offenbaren sich die bisher erfolgten semantischen Erweiterungen als permanent inadäquat.
Letztlich war der Versuch, durch Standardisierung, semantische Erweiterungen in HTML einzubringen in doppelter Hinsicht zum Scheitern verurteilt:
1. birgt der Ansatz die Gefahr, die Elementmenge in unbekannte Größen zu erweitern
2. muß die Semantik jedes Tags definiert, abgestimmt und verabschiedet werden.

Aus diesen Gründen wurde seitens des W3C nach einer tragfähigeren Lösung gesucht. Unter Rückgriff auf die HTML-Wurzeln (als Anwendung der Metasprache SGML) wurde das Projekt *SGML for the Web* initiiert.
Der letztendlich verabschiedete Vorschlag zur *eXtensible Markup Language* (XML) bildet konzeptionell eine Untermenge der Sprachmöglichkeiten von SGML. Konsequenterweise ist jedes XML-Dokument auch ein gültiges SGML-Dokument.

Die Abweichung zu SGML wird besonders aus den Entwicklungszielen für XML deutlich:

1. Einfache Nutzung im Internet.
   In Abkehr von den Hauptnutzung SGMLs als offline Dokumentationsformat wird die Untermengenbildung *XML* für die primäre Nutzung im Internet vorgenommen.
2. Unterstützung eines breiten Anwendungsspektrums.
   Auch hier soll die Untermengenbildung das Einsatzspektrum über die Hauptnutzung SGMLs als Format der technischen Dokumentation hinaus befördern.
3. SGML Kompatibilität.
   XML bildet eine echte Untermenge des ISO-Standards SGML, durch diesen Schritt kann jedes XML-Dokument auch als gültiges SGML-Dokument interpretiert und durch die entsprechenden SGML-Werkzeuge verarbeitet werden.
4. Einfache Applikationsentwicklung.
   Die Untermengenbildung wird im Hinblick auf eine gegenüber SGML deutlich vereinfachte Entwicklung von XML verarbeitenden Applikationen vorgenommen.
5. Minimierung optionaler Sprachmerkmale -- Idealerweise gleich Null.
   Auch dieses Ziel ist im Hinblick auf eine vereinfachte Applikationsentwicklung, aber auch eine einfachere Benutzbarkeit durch Menschen auf dem Wege der Komplexitätsreduktion zu interpretieren.
6. Lesbarkeit.
   Das entstehende Textformat soll für Menschen und Maschinen gleichermaßen les- und verstehbar sein.
7. Kompakte Spezifikation.
   Die erstehende XML-Spezifikation sollte deutlich weniger Umfang aufweisen als der SGML-Vorgängerstandard. Letztlich konnte die reine Seitenzahl von über 600 Seiten für die SGML-Spezifikation auf ungefähr 30 Seiten für XML reduziert werden.
8. Formaler und präziser Sprachentwurf.
   Um die schnelle Akzeptanz seitens der Anwender zu forcieren erachteten die Mitglieder der XML-Arbeitsgruppe die schnelle Verfügbarkeit von XML-Werkzeugen für essentiell. Aus diesem Grunde sollte der XML-Sprachentwurf möglichst leicht und eindeutig in XML-Werkzeuge zu implementieren sein.
9. Leichte Dokumenterstellung.
   Die Erstellung von korrekten XML-Dokumenten sollte idealerweise so einfach sein, daß hierfür keine speziellen Werkzeuge benötigt werden.
10. Nicht notwendigerweise knappes Markup.
    Kompaktheit und Effizienz hinsichtlich des Volumens eines XML-Dokuments war zu keinem Zeitpunkt eines der Hauptentwicklungsziele. Auf der Basis des XML-Information Sets ist es jedoch möglich beliebig kompakte Binärformate identischer Mächtigkeit zur die in der XML-Spezifikation vorgestellten Textnotation zu definieren.

XML stellt jedoch keine echte semantische Auszeichnungssprache dar, da durch die Metasprache lediglich eine Möglichkeit zur Formulierung eigener Syntax gegeben ist. Die Bedeutung der Elemente bleibt jedoch unberücksichtigt, und kann mittels XML nicht ausgedrückt werden.

Tabelle 1: Einige chronologische Eckdaten

Jahr	Ereignis
1945	Vannevar Bush diskutiert in seinem Artikel As We May Think ein persönliches Informationssystem mit Kommunikationsmöglichkeiten und Zugriff auf Bücher, Tonaufnahmen, etc. unter dem Namen *Memex*.
1967	William Tunnicliffe (Chairman des Graphic Communications Association (GCA) Composition Committee) schlägt aus seinen Erfahrungen bei der wiederholten Erstellung von Telephonkatalogen (*yellow pages*) vor, häufig auftretende strukturelle Elemente zu standardisieren.
September 1967	William Tunnicliffe (Vorsitzender der Graphic Communication Association) spricht sich auf einer Konferenz des *Printing Office* der Regierung von Kanada für die Separierung von Inhalt und Format aus.
Ende der 1960er Jahre	Stanley Rice, ein New Yorker Schriftsetzer, schlägt *editorial structure tags* vor. Der CGA-Direktor Norman Scharpf initiiert das Projekt *GenCode*.
1969	Charles Goldfarb, Edward Mosher und Raymond Lorie entwickeln bei der IBM die *Generalized Markup Language* (GML). Anwendungshintergrund war ein Projekt zur Integration von Informationssystemen für Anwaltskanzleien.
1970	Goldfarb formuliert zwei Grundprinzipien generalisierter Auszeichungssprachen: 1) Auszeichnungssprachen beschreiben die Dokumentstruktur, nicht die physischen Charakteristika wie Präsentation 2) Die Struktur der Auszeichnungssprache soll so gewählt sein, daß sie sowohl von Menschen als auch Maschinen interpretiert werden kann
1978	ANSI ruft *Computer Languages for the Processing of Text*-Komitee ins Leben. Ziel ist die Weiterentwicklung der GML zu einem nationalen US-Standard.
1980	• ANSI veröffentlicht ersten Entwurf einer standardisierten GML (SGML). • Tim Berners-Lee tritt seine Arbeit am Europäischen Kernforschungszentrum CERN an. Dort entwickelt er in der Folgezeit die (niemals veröffentlichte) Hypertextanwendung *Enquire*.
1983	Der International Revenue Service (IRS) und das US Verteidigungsministerium (DoD) übernehmen den sechsten Entwurf zur SGML (auch bekannt als GCA 101-1983).
1984	Die SGML-Arbeitsgruppe nimmt unter Schirmherrschaft der International Standardization Organization (ISO) als ISO/IEC JTCI/SC18/WG8 ihre Arbeit auf. Goldfarb dient als *technical leader* der ISO-Gruppe, sowie dem umorganisierten ANSI-Komitee X3V1.8.
1985	Norm-Entwurf zu SGML veröffentlicht.
15. Oktober 1986	ISO verabschiedet SGML als ISO 8879:1986.
März 1989	Berners-Lee schlägt mit dem Dokument *Information Management: A Proposal* ein SGML-basiertes Hypertext-System zum Informationsaustausch vor.
1990	Am Weihnachtstag nimmt das World Wide Web seinen Betrieb mit zwei Maschinen am CERN auf. Die notwendigen Implementierungen von HTML, HTTP und URL erfolgten durch Berners-Lee. Die erste WWW-Verbindung wird zwischen Berners-Lees Workstation und Robert Cailliaus' NeXT-Rechner aufgebaut. Ein Screenshot des ersten Web-Browsers NeXTStep-Implementierung des Browsers
1991	Beginn der turnusmäßigen Überarbeitungsphase von ISO 8879.
3. November 1992	Erster Entwurf zu HTML
Juni 1993	Einreichung des ersten HTML Entwurfs bei IETF.
Oktober 1994	Gründung World Wide Web Consortium
14. November 1996	Erster Entwurf zu XML vorgestellt
14. Januar 1997	Verabschiedung der HTML v3.2
1998	W3C gibt die erste Version von XML als Recommendation frei.
2000	• W3C gibt XHTML v1.0 -- die Reformulierung von HTML v4.01 zu einer XML-Anwendung -- frei. • W3C verabschiedet XML 2nd edition; sie integriert u.a. die XML Namespaces und behebt einige editorielle Fehler.
2. Mai 2001	Das W3C verabschiedet den XML Schema-Standard. Er geht an vielen Stellen deutlich über die ererbten SGML-Möglichkeiten hinaus, und markiert den Übergang von Präsentations-orientierten Strukturen hin zu Datenstrukturen.

Zum Abschluß dieser Einführung seinen die zehn Punkte zusammengestellt und kommentiert, die durch das World Wide Web Consortium als plakative Kurzcharakterisierung von XML veröffentlicht wurden:

1. XML steht für strukturierte Daten.
   Diese Aussage betont die Rolle von XML als Sprache um Sprachen zu erzeugen. Nicht XML wird innerhalb verschiedenster Applikationen direkt verarbeitet, sondern XML basierte Formate. So steht nicht die XML selbst für all diese Anwendungsdomänen, sondern die jeweiligen problemspezifischen XML-basierten Sprachen. XML selbst dient lediglich der Strukturierung der verschiedensten darzustellenden Daten.
   Gleichzeitig rückt durch Aussage die Rolle der XML als Datenformat in den Vordergrund und läßt so die Weiterentwicklung gegenüber den präsentationsorientierten Vorläufern deutlich werden.
   Die Vorlesungskapitel Strukturelle Grundkonzepte, Dokument-Typ-Definitionen sowie XML Schemasprachen

vermitteln einen Eindruck dieses Wandels und dokumentieren die Grundlagen des gegenwärtigen datenorientierten Einsatzes der XML.

2. XML sieht ein wenig wie HTML aus.
   Diese Aussage soll offenkundig einerseits den bisherigen HTML-verwendenden Web-Autoren den Einstieg in die XML schmackhaft werden lassen. Dennoch führt sie ein wenig von der Grundidee XMLs als generischer Auszeichnungssprache für beliebigste Anwendungen weg, indem sie den Blick auf HTML focussiert.
   Die -- im Grunde der Verwandschaft zu SGML geschuldete -- offensichtliche syntaktische Ähnlichkeit zu HTML wird bereits bei der Betrachtung der strukturellen Grundkonzepte deutlich.
3. XML ist Text, aber nicht zum Lesen.
   XML-Dokumente können sicherlich im wörtlichen Sinne „gelesen" werden ... Die Aussage zielt jedoch auf den intendierten Einsatzzweck von XML: der Darstellung von Daten für den Austausch zwischen Maschinen.
   Unbenommen dessen kann XML selbstverständlich auch von Menschen gelesen und verstanden werden, wenngleich dies bei umfangreicheren XML-Dokumenten durchaus mühsam werden kann.
   Aufschluß über die textuelle Natur XMLs, insbesondere im Hinblick auf die Verwendung unterschiedlicher Alphabete, liefert das Kapitel strukturelle Grundkonzepte.
4. XML ist vom Design her ausführlich.
   Hiermit wird versucht dem häufig geäußerten Kritikpunkt der Platzzunahme XML-codierter Inhalte gegenüber klassischen Darstellungsweisen etwas pauschal entkräftend entgegenzutreten. Sicherlich geht das W3C in dieser Aussage nicht fehl, wenn die Entwicklung der Netzwerkbandbreiten, der CPU-Leistung und der Speicherkapazitäten berücksichtigt. Andererseits ist die Aufblähung der XML-formatierten Inhalte im Vergleich zu optimierten Binärformaten nicht von der Hand zu weisen, wird jedoch durch die mit der Verwendung von XML einhergehenden Vorteile mehr als ausgeglichen.
   Einen ersten Eindruck der Natur XML-codierter Inhalte liefert das Kapitel strukturelle Grundkonzepte. Dort finden sich auch Ansätze die bekannte XML-Syntax kompaktifiziert darzustellen ohne die Vorteile der generischen Auszeichnungssprache aufgeben zu müssen.
5. XML ist eine Familie von Techniken.
   Eine Aussage, die durch alle drei Kapitel der Vorlesung unterstrichen wird, die deutlich zeigen, daß XML nicht als isolierte Idee oder Technik anzusehen ist -- sondern erst im Zusammenspiel mit anderen XML-Standards und eingebettet in Applikationen und Infrastrukturen -- seine volle Wirkungsmächtigkeit entfalten kann.
6. XML ist neu, aber nicht so neu.
   Diese Bezugnahme soll nochmals unterstreichen, daß XML keineswegs den Anspruch erhebt eine vollkommen neue technische Errungenschaft zu sein, sondern vielfach bekanntes und erprobtes aus der Informatik wiederverwendet und im neuen Verwendungskontext weiterentwickelt.
   Diese Aussage wird durch die in den einzelnen Kapiteln dargebotenen Rückbezüge auf bereits bekannte Techniken und Lösungsformen untermauert.
7. XML überführt HTML in XHTML.
   Diese Aussage greift nochmals die Beziehung zwischen XML und HTML auf. Diesmal soll die Rolle von XML im Bezug auf die Weiterentwicklung von HTML zum XML-basierten Vokabular *XHTML* unterstrichen werden. So löst XML die Abhängigkeit zwischen SGML und HTML auf und reformuliert HTML auf der Basis von XML.
   Das Kapitel XHTML führt kurz in die Entwicklung der neuen HTML-Varianten auf Basis der XML ein und skizziert die vorgenommen Änderungen und zukünftige Erweiterungen dieser Hypertextsprache.
8. XML ist modular.
   Hierdurch wird unterstrichen, daß XML kein in sich geschlossenes monolithisches Gebilde darstellt, sondern einzelne Vertreter aus der Familie der XML-Sprachen wahlfrei zur Lösung konkreter Probleme herangezogen werden können. Ebenso wird die Sprachfamilie beständig an verschiedensten Stellen unabhängig voneinander weiterentwickelt, ohne einer zentralen Koordination zu bedürfen.
   Einen Eindruck dieser Modularität liefern bereits die Grundlagensprachen des ersten Kapitels. Eindrucksvoll verstärkt wird diese Aussage durch die im zweiten Kapitel eingeführten problemorientierten Sprachen.
9. XML ist die Basis für RDF und das Semantic Web.
   Grundidee des Semantic Web ist die Weiterentwicklung des sichtbaren XHTML-basierten Webs unter Nutzung seiner datenorientierten Ergänzung XML zu einem Netz von Sinnzusammenhängen. Die technische Voraussetzung hierzu bildet das im gleichnamigen Kapitel beschriebene Resource Description Framework welches eine XML-Sprache zur Darstellung von beschreibenden Daten zu beliebigen Quellen definiert.
10. XML ist lizenzfrei, plattform- und herstellerunabhängig, und gut unterstützt.
    XML ist eine durch das World Wide Web Consortium herausgegebene Spezifikation, die kostenfrei über das Web bezogen werden kann und durch Interessierte ohne weitere Lizenzkosten in eigenen kommerziellen Produkten verwendet werden. Durch den Standardisierungsprozeß innerhalb des World Wide Web Consortiums wird sichergestellt, daß keine Ausführungsplattform bevorzugt wird und gleichzeitig keine Nachteile für Andere entstehen. Dies wird durch die herstellerunabhängige Organisation des Gremiums versucht zu garantieren, in dem zwar Hersteller Mitglied werden können, die technischen Entscheidungen jedoch Arbeitsgruppen obliegen, die nicht durch eine Firma dominiert werden können.

Web-Referenzen 1: Vertiefende Informationen

- Artikel in der Online-Ausgabe des *Economist* über Ted Nelson -- *The Babbage of the web*
- COT1800 Public Networks, Lecture 8, Standard Generalised Markup Language
- Brief History of Document Markup
- XML, Element Types, DTDs, and All That
- Clark, J.: *Comparison of SGML and XML*

Web-Referenzen 2: Weiterführende Links

- Browser Timelines
- Browser Emulator

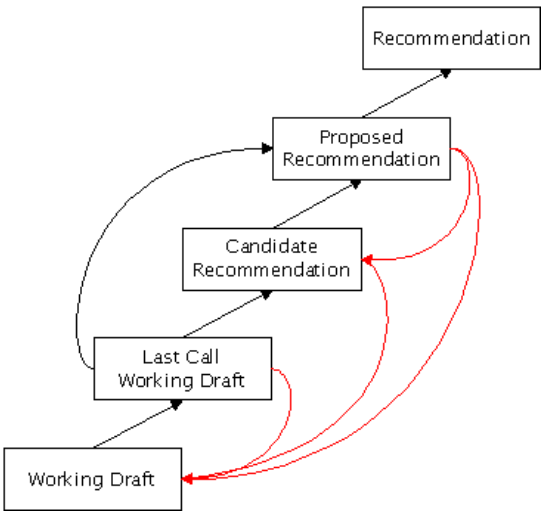
Die Rolle des World Wide Web Consortiums:
Das World Wide Web Consortium (W3C) stellt keine Normierungsorganisation im klassischen Sinne staatlicher Standardisierung wie die ISO, mit ihren nationalen Organisationen wie dem DIN, die UN, die ITU (früher CCITT) oder die IEC dar. Dies äußert sich schon darin, daß es keine normativen Dokumente verabschiedet, sondern lediglich

Einsatzempfehlungen *recommendations* (im Sinne des IETF RFC 2119) gibt. Konkret bedeutet dies: Das W3C ist de jure nicht in der Lage auf rechtlichem Wege einklagbare Normen zu setzen. Vielmehr spricht es Empfehlungen aus, von denen unter bestimmten Umständen durchaus abgewichen werden kann. Jedoch empfiehlt RFC 2119 zunächst die Auswirkungen zu untersuchen und die endgültige Entscheidung gründlich abzuwägen.

Organisatorisch ist das W3C seit seiner Gründung 1994 ein Projekt des Massachusetts Institute of Technology (Labaratory for Computer Science) in Zusammenarbeit mit CERN, der Kejo Universität in Japan und dem European Research Consortium for Informatics and Mathematics (ERCIM). Unterstützt wird das Projekt durch DARPA und die Europäische Kommission.
Das W3C versteht sich als offenes herstellerunabhängiges Gremium. Zugang zu den Ergebnissen erhält jeder Interessierte kostenfrei über die W3C-Web-Site.
Die Mitwirkung an neuen Standards ist auf die Mitglieder des W3C beschränkt. Dieser Status ist juristischen Personen vorbehalten, steht aber generell jeder Institution oder jedem Unternehmen offen. Durch die Mitgliedschaft wird ein Sitz im *Advisory Committee*, dem offiziellen Bindeglied zwischen Mitgliedern und W3C-Team, erworben.
Darüberhinaus kann jedes W3C-Mitglied an den laufenden Standardisierungsaktivitäten teilnehmen.

Das W3C veröffentlicht sechs verschiedene Typen von Dokumenten, wobei jedoch nur fünf formal zum Standardisierungsprozeß zu zählen sind.

- **Note**: Allgemeines datiertes Dokument zur Vorstellung einer Idee oder neuen Technik; es kann sich auch um einen formalen Kommentar handeln.
  Der Note-Status ist *nicht* Bestandteil des Standardisierungsprozesses. Insbesondere stellen Dokumente dieses Typs keine Absichtserklärung seitens des W3C über eine zukünftige Weiterentwicklung der Technik dar.
- **Working Draft (WD)**: Interimsstand einer W3C-Arbeitsgruppe zu einem Thema das durch das W3C weiter bearbeitet wird.
  WD-Dokumente werden an bestimmten Zeitpunkten (zumeist alle drei Monate nach Beginn der Arbeiten) erzeugt, sie stellen keinen abgestimmten und stabilen Arbeitsstand dar, sondern dokumentieren nur den aktuellen Diskussionsstand.
  Üblicherweise werden Dokumente dieses Status weiterentwickelt. (Gegenbeispiele finden sich unter working drafts no longer in development)
- **Last Call Working Draft**: Letzter Zustand eines Working Drafts. Er markiert die Erreichung der im Anforderungsdokument festgelegten Ziele.
- **Candidate Recommendation (CR)**: Direktor des W3C bestätigt die Erreichung der definierten Anforderungen, bzw. erklärt sich mit der Nichterreichung bestimmter Anforderungen einverstanden.
  Alle Anfragen und Kommentare an die Arbeitsgruppe wurden formalisiert bearbeitet, d.h. dokumentiert und beantwortet.
  Abhängigkeiten zu anderen Arbeitsgruppen wurden geklärt.
- **Proposed Recommendation (PR)**: Erweiterung des CR-Status. Bestandteile der PR sind umgesetzt. Idealerweise sollte die Arbeitsgruppe zwei interoperable Implementierungen jedes Spezifikationsbestandteils vorweisen können.
  Die Spezifikation wird dem Advisory Committee zur Begutachtung (*review*) übergeben.
- **Recommendation (REC)**: Direktor erklärt sich mit dem Grade der Unterstützung durch das Advisory Committee einverstanden und erklärt die Proposed Recommendation zum Recommendation und damit zum offiziellen W3C-Standard.



Web-Referenzen 3: Weiterführende Links

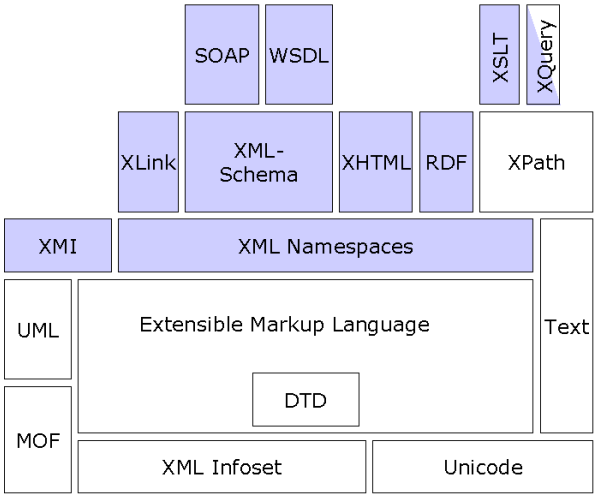
- Informationen über das W3C
- Liste der W3C-Mitglieder
- Deutsches W3C-Büro
- W3C Process Document; es definiert die verschiedenen W3C-Standardisierungsstatus

---

Definition 1: XML-Sprache

Eine Anwendung der Extensible Markup Language. Ein Vokabular, das aus Symbolen und der ihnen zugewiesenen Bedeutung (Semantik) gebildet wird, ergänzt um Regeln (grammatikalische Struktur und Gültigkeitsregeln für den Inhalt (z.B. Datentypen)) zur Kombination der Vokabularelemente.
Anwendungen einer so neu geschaffenen XML-Sprache *L* werden als XML-Dokumente, auch: *L*-Dokumente, bezeichnet.

Die XML-Sprachfamilie:



Die Graphik stellt den Zusammenhang einiger Basistechniken der XML sowie verschiedene bekannte XML-Sprachen dar.

Die strukturelle Basis der Extensible Markup Language bilden heute der XML Information Set, welcher die zentralen Begriffe der XML Strukturprimitive definiert, gemeinsam mit dem Codierungsstandard (ISO/IEC 10646) des Unicode Konsortiums zur plattformunabhängigen Darstellung beliebiger Zeichensätze.
XML selbst wurde ursprünglich als eine Untermenge der ISO-Norm der *Standard Generalized Markup Language* (SGML) definiert. Daher „erbt" XML auch den dort definierten Grammatikmechanismus, die sog. *Document Type Definitions* (DTD), welche eine zusätzliche Text-basierte Sprache zur Festlegung des Vokabulars einer XML-Sprache liefern.

Aufbauend auf dem grundlegenden XML-Standard des W3C führt die Graphik die *XML Namespaces* ein, welche zur Unterscheidung von gleichen Bezeichnernamen in verschiedenen Verarbeitungskontexten dienen.
Diese Erweiterung der XML gestattet es Anwendern ihre Element- und Attributnamen vollkommen frei festlegen zu können, ohne zukünftig, etwa bei der Syndikatisierung verschiedener Dokumente mit Namenskonflikten konfrontiert zu werden.

Durch die XML HyperText Markup Language (XHTML) wird der große und wichtige Bereich der präsentationsorientierten Auszeichnungssprachen vertreten. XHTML v1.0 stellte die Reformulierung der bekannten Web-Sprache HTML v4.01 ein, die als SGML-Sprache nicht mehr durch das W3C weiterentwickelt wird.

Am Rande sei noch auf die (auf der Abbildung nicht dargestellte) ISO-Norm *DSSSL* (sprich: *Dißl*) -- die *Document Style Semantics and Specification Language* -- hingewiesen. Sie erlaubt die Formulierung von Transformationsregeln, die beliebige SGML-Dokumente in ein präsentationsfähiges Format überführen. Hierzu ist ein generischer Prozessor notwendig, der als Eingabe das umzuformende SGML-Dokument und die Transformationsregeln akzeptiert.
Bekannte DSSSL-Implementierungen: James Clarks Jade, oder Seng.
Die mit DSSSL gesammelten Erfahrungen flossen in die Entwicklung der XML-Stylesheet und Transformationssprachen XSL-Formatting Objects und XSLT ein.

Ein zentraler Bestandteil des Hypertext-Gedankens ist die nichtlineare Navigierbarkeit. In Web-Dokumenten, die mit der Sprache HTML erstellt wurden, findet sich dies durch die bekannten Hyperlinks verwirklicht. Der darstellende Browser sorgt dafür, daß durch Mausklick auf den meistens farblich und durch Unterstreichung hervorgehobenen Link die hinterlegte Aktion ausgeführt wird; üblicherweise wird ein neues HTML-Dokument geladen.
Auch XML verfügt über einen solchen -- jedoch durch Verallgemeinerung ungleich mächtigeren -- Verknüpfungsmechanismus. Er beschränkt sich jedoch nicht nur auf visuelle Hyperlinks, sondern steht durch den W3C-Standard XLink für alle XML-Sprachen zur Verfügung. Dadurch erweitert sich das aus HTML bekannte System u. a. um mehrgliedrige Verweise, die auf eine Menge von Dokumenten verweisen sowie verschiedene Verhaltensmuster bei der Traversierung eines Links.

Während Hyperlinks zur direkten Referenzierung einzelner Elemente eines XML-Dokuments verwendet werden, lagert die Lokatorsprache XPath dies in eine eigene Sprache aus. Sie stellt den Kern einer Anfragesprache, zur Extraktion verschiedenster Informationen (Knotenmengen, Positionsangaben, Werte, etc.) aus einem XML-Dokument, dar.
Aufgrund des dadurch eröffneten Anwendungsgebietes finden sich Teile von XPath in der Transformationssprache XSLT wieder. Hier dienen sie zum Auffinden der zu transformierenden Dokumentstrukturen.
XSLT erlaubt die Generierung beliebiger Unicode-Ströme aus XML-Strukturen. Häufigstes Einsatzgebiet in der Praxis ist aber die Verwendung zur Umformung von XML-Dokumenten. Hierzu wird nicht die volle Mächtigkeit der Ausgabe beliebiger Texte genutzt, sondern es werden „lediglich" XML-Dokumente erzeugt.
Für das Anwendungsgebiet freier Dokumentstrukturen, wie z.B. RTF, Postscript oder LaTeX werden durch das W3C die Formatting Objects entwickelt. Konzeptionell baut dieser Standard auf XSLT auf, und reichert dieses um notwendige Primitive zur Erzeugung der erwähnten Formate an.

Ausgehend von mit XPath beschaffenen Lokalisierungsmöglichkeiten innerhalb eines XML-Dokuments oder einer Menge von XML-Dokumenten ist es konzeptionell nur ein kleiner Schritt eine Anfragesprache zu definieren. Dies wird durch XQuery vollzogen. Dieser Sprachstandard definiert eine SQL-artige algebraische Anfragesprache. Für beide Sprachen existiert eine textuelle nicht-XML-konforme kompakte Syntax, die ihre Einbettung in existierende XML-Vokabulare gestattet.

Zur Unterscheidung zwischen XML-codierten Sprachen und anderen Ansätzen hinterlegt die Graphik der Abbildung 2 alle im folgenden behandelten Standards zu denen eine XML-Sprache existiert farblich.

Auch das Resource Description Framework (RDF) baut auf den durch die Namensräum-Spezifikation etablierten Grundlagen auf. RDF definiert eine XML-Sprache, die ausgehend von klassischer Prädikatenlogik die Formulierung und maschinelle Auswertung von Aussagen über Web Ressourcen (etwa Text- oder Bilddaten) gestattet.

Die Erschließung neuer Anwendungsfelder läßt die Unzulänglichkeiten der Dokument-Typ-Definition als Möglichkeit zur Formulierung von XML-Vokabularen offenkundig werden, unterstützt der DTD-Mechanismus doch weder ein hinreichend entwickeltes Typsystem noch gestattet er die angemessene Formulierung von Strukturen wie sie in XML-Dokumenten benötigt werden.
Aus diesem Grunde entwickelt die XML-Sprache XML Schema die Primitive der DTD in einem eigenständigen XML-Vokabular fort, welches neben erweiterten Strukturierungsmöglichkeiten auch ein umfangreiches Typsystem zur Verfügung stellt.

Als eine Anwendung der Schemasprache stellt die Abbildung die beiden Sprachen WSDL und SOAP vor, die im Umfeld der *Web Services* große Bedeutung erlangt haben.
Diese XML-Sprachen dienen der Beschreibung von Aufrufschnittstellen und -charakteristika von Funktionalitäten, die über Internetprotokolle zur Verfügung gestellt werden. Die Darstellung des eigentlichen Aufrufs, des Namens der aufzurufenden Funktion sowie der benötigten Übergabeparameter, kann ebenfalls XML-codiert erfolgen. Den bekanntesten Ansatz hierzu bildet das SOAP-Protokoll welches einen betriebsystem- und programmiersprachenunabhängigen Nachrichtenmechanismus definiert.

1.2 Strukturelle Grundkonzepte

Die grundlegende XML-Syntax ist in der namensgebenden W3C-Recommendation der Extensible Markup Language definiert. Die Semantik der Metasprache wird hingegen durch den W3C-Standard des XML Information Set festgelegt. Diese Spezifikationen beinhalten die grundlegenden Definitionen hinsichtlich Terminologie und Beziehung der verschiedenen möglichen Elemente eines XML-Dokuments. Im vorliegenden Teilkapitel werden beide Sprachaspekte grundlegend eingeführt und ein erstes Verständnis der XML vermittelt. Dabei wird in Form von Ausblicken auf nachfolgende Abschnitte der Bogen zu Grammatikdefinitionssprachen und weiterführenden Konzepten wie Namensräumen gespannt.
Zum leichteren Verständnis sind die aus der offiziellen Spezifikationen entnommenen formalen Grammatikdefinitionen der EBNF-Notation durch vereinfachte graphische Strukturdarstellungen ergänzt.

Definition 2: XML Dokument
Ein XML-Dokument ist ein Datenstrom (der nicht zwingend als Datei vorliegen muß), welcher den Strukturierungsprinzipien der eXtensible Markup Language genügt.

Definition 3: XML Information Set
Die Spezifikation des XML Information Sets definiert die Semantik der Metasprache XML, d.h. ihre zentralen Begriffe.
Gleichzeitig setzt es diese Begriffe in Beziehung und definiert so syntaxunabhängig die Struktur eines XML-Dokumentes.

Ausgehend von der Allgemeinheit der Aussage aus Definition 1 folgt, daß der Infoset neben seinem theoretischen Wert als Semantikdefinition zur XML auch zur Formulierung der Datenstrukturen, welche innerhalb eines XML-Prozessors vorliegen müssen, um beliebige XML-Dokumente verarbeiten zu können, herangezogen werden kann. Daher läßt sich ein XML-Prozessor definieren als:

Definition 4: XML-Prozessor
Ein XML-Prozessor ist eine maschinelle Komponente (typischerweise: Software), die zum Lesen, Speichern und Verarbeiten eines XML-Dokuments eingesetzt wird.
Er erlaubt Zugriff auf den Inhalt und die Struktur des XML-Dokuments.

Die XML-Spezifikation faßt den XML-Prozessorbegriff etwas enger und beschränkt ihn lediglich auf Software-Module, die XML-Dokumente lesend verarbeiten. Konzeptionell spricht jedoch nichts gegen eine Umsetzung in Hardware, beispielsweise im Kontext eingebetter Systeme etc. (In XML-Spezifikation nachschlagen)
Ferner nimmt die XML-Spezifikation an, ein Prozessor operiere nicht eigenständig, sondern im integrierten Zusammenspiel mit einer Applikation.

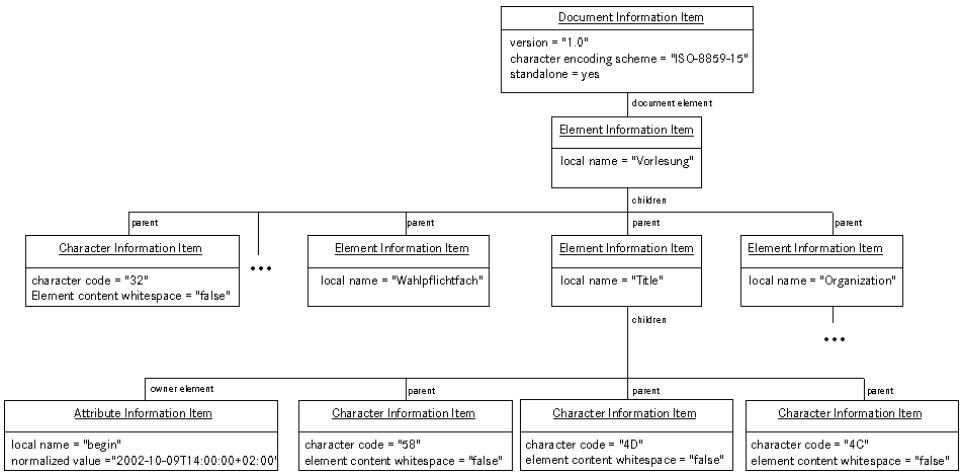
Beispiel 1: Ein erstes XML-Dokument

```
(1)<?xml version="1.0" encoding="ISO-8859-15" standalone="yes"?>
(2)<Vorlesung>
(3)   <Wahlfach/>
(4)   SS2003
(5)   <Titel beginn="2003-03-17T14:00:00+01:00">XML</Titel>
(6)   <Hochschule>Fachhochschule Furtwangen</Hochschule>
(7)   <Praktikum>Freiwilliger Übungsbetrieb</Praktikum>
(8)</Vorlesung>
```

Download des Beispiels

Das Beispiel zeigt ein erstes einfaches XML-Dokument, welches bereits die häufigst verwendeten Sprachelemente der XML versammelt.
Jedem XML-Dokument entspricht genau ein Information Set, der alle Informationselemente des Dokuments in Form einer Baumstruktur beinhaltet. Die nachfolgende Abbildung zeigt den Information Set des Beispiels in der Notation

eines UML-Klassendiagramms. Dabei sind die einzelnen Knoten des Information Sets als Objekte (Klassensymbole mit unterstrichenem Klassennamen) und die Eigenschaften der Knoten als Attributwerte dargestellt.



Document Information Item

Jedes Information Set besteht genau aus einem *Document Information Item*. Dieses stellt den äußeren Rahmen des XML-Dokuments dar. Es beinhaltet dokumentbezogene Informationen, wie die verwendete XML-Version und das gewählte Codierungsschema innerhalb des Unicode-Systems.
Das Document Information Item enthält daher u.a. die Informationen des XML-Dokumentprologs in der erste Zeile jedes Dokuments. Das durch die öffnende Winkelklammer und ein Fragezeichen eingeleitete Konstrukt ist in der ersten Zeile des Beispiels 1 dargestellt. Innerhalb des Prologs findet sich die Zeichenkette `xml`, sowie die Bezeichner `version` und `encoding`. Beiden ist ein durch doppelte Hochkommata umschlossener Wert nachgestellt, `1.0` für `version`, bzw. `ISO-8859-15` für `encoding`.
Beendet wird der Prolog wiederum durch ein Fragezeichen und die schließende Winkelklammer. Wird auf die Angabe des optionalen Prologs im Dokument verzichtet, so sind die daraus ableitbaren Angaben im Document Information Item nicht gesetzt.

Als weitere Eigenschaften verfügt jedes Document Information Item über eine geordnete Liste von Kindknoten. Darin ist genau ein Element Information Item enthalten, welches den Startknoten des XML-Dokuments verkörpert. Wegen seiner hervorgehobenen Bedeutung als Wurzel des Dokumentbaumes wird dieser Knoten auch als `Document Element` bezeichnet.
Zusätzlich kann die Liste Elemente vom Typ Processing Instruction Information Item enthalten. Sie dienen der Darstellung von Verarbeitungsanweisungen, die durch den XML-Prozessor interpretiert werden.
Im Kopfbereich vor `Document Element` plazierte XML-Kommentare werden durch Comment Information Items innerhalb der `children`-Liste dargestellt.
Verfügt das XML-Dokument über eine Dokumenttypdeklaration, so findet sich auch ein Document Type Declaration Information Item in der Liste.
Deklarierte *Notationen* werden in einer ungeordneten Menge als Notation Information Item dargestellt.

Zusammengefaßt enthält das Document Information Item folgende Informationen:

- **Kindknoten**: In der Reihenfolge des Auftretens im Dokument geordnete Liste. Sie enthält mindestens ein `Element Information Item`, welches in der Rolle des `Document Items` fungiert. Ferner je ein Element des Typs `Processing Instruction Information Item` für jede Processing Instruction die außerhalb des Wurzelements definiert ist und jeweils ein `Comment Information Item` zu jedem definierten Kommentar.
- **Document Element**: Ein Element des Typs `Element Information Item`, das auf den Wurzelknoten des Dokuments verweist.
- **Notations**: Die in der DTD definierten Notationen als ungeordnete Menge.
- **Unparsed Entities**: Die in der DTD definierten ungeprüften Entitäten als ungeordnete Menge.
- **Basis URI**: Lokation, falls bekannt, des XML-Dokuments in Form eines *Uniform Resource Identifiers* (URI) gemäß IETF RFC 2396.
- **Character Encoding Scheme**: Der Name der gewählten Codetabelle aus dem Unicode-Standard.
- **Standalone**: Legt -- als Boole'scher Wert (Zugelassene Belegungen: *yes*, *no*) -- fest, ob die im Dokument gespeicherten Daten durch eine sie verarbeitende Applikation interpretiert und somit ergänzt oder verändert werden. Häufigste Form dieses Interpretationsvorganges ist die Präsenz einer expliziten Grammatik in Form einer *Document Type Definition*, welche Gültigkeitsregeln für eine Familie von Dokumenten formuliert.
  Konsequenterweise bedeutet daher die Belegung mit *no*, daß die gespeicherten Daten entweder durch die Applikation interpretiert werden, dies ist beispielsweise bei der Auflösung von Entitäten der Fall, oder durch eine *DOCTYPE*-Deklaration ein Verweis auf die externe Dokumentgrammatik erfolgt. Die Angabe von *standalone* ist optional. Fehlt sie und ist gleichzeitig eine DOCTYPE-Deklaration im Dokument gegeben, so wird *standalone="no"* angenommen.
  Im Beispiel ist die standalone-Deklaration auf *yes* gesetzt, d.h. es existiert explizit keine Dokumentgrammatik. (In XML-Spezifikation nachschlagen)
- **Version**: Die eingesetzte XML-Version. Dieser Wert wird aus dem Dokumentprolog übernommen.
- **All Declarations Processed**: Boole'scher Wert der angibt ob der verarbeitende XML-Prozessor die evtl. verfügbare DTD vollständig abgearbeitet hat.

Wie auch im Beispieldokument, bildet die erste Zeile den sog. *Prolog* eines jeden XML-Dokuments (In XML-Spezifikation nachschlagen) . Die Angabe der Version ist zwingend und derzeit auf die Konstante `1.0` fixiert. Die aktuelle XML-Spezifikation sieht als gültige Belegung der Versionsangabe ausschließlich die Zeichenkette `1.0` vor. Zukünftigen Weiterentwicklungen ist es jedoch freigestellt auch andere Revisionskennungen zu vergeben.
`encoding` leitet das zweite Namen-Wert-Paar ein. Die Deklaration ist innerhalb des Prologs optional, und kann daher

auch unterbleiben. Die Zeichenkette der Encodingdeklaration benennt das Codierungsschema, welches für das so gekennzeichnete Dokument verwendet wurde. Es definiert den Satz der innerhalb des Dokumentes zugelassenen Zeichen fest.
Gemäß Produktion 22 der XML-Syntaxdefinition ist der gesamte Prolog optional.

Die Encoding-Deklaration hat folgendes Aussehen (In XML-Spezifikation nachschlagen) :

```
[80] EncodingDecl ::= S 'encoding' Eq ('"' EncName '"' | "'" EncName "'" )
[81] EncName      ::= [A-Za-z] ([A-Za-z0-9._] | '-')*
[3]  S            ::= (#x20 | #x9 | #xD | #xA)+
[25] Eq           ::= S? '=' S?
```

Die Festlegung der Produktion 80, sowie die der Produktion 23, stellt heraus, daß sich die Encodingdeklaration nicht auf die Prologzeile selbst auswirkt. Hier sind die beiden Zeichenketten `xml` und `encoding` in der Codierung UTF-8 oder UTF-16 Vorschrift.

Als Belegungen des `Encoding Namens` (`EncName`) sind beliebige Zeichensätze zugelassen. Der XML-Standard empfiehlt jedoch lediglich auf die durch die *Internet Assigned Numbers Authority* verwalteten zurückzugreifen (Dokument: Official Names for Character Sets) (In XML-Spezifikation nachschlagen) .
Die häufigsten praktisch eingesetzten Deklarationen sind die der ISO-8859 (*extended ASCII*)-Familie, sowie die der Unicode- und ISO-10646-Standards.
Die verschiedenen Abschnitte der ISO-8859 Familie werden als `ISO-8851-`*n* ausgedrückt, wobei *n* die Nummer des Abschnittes des zugehörigen ISO-Dokuments referenziert. Ferner können die durch JIS X-0208-1997 normierten asiatischen Zeichensätze als `ISO-2022-JP`, `Shift_JIS` und `EUC-JP` dargestellt werden.

```
<?xml version="1.0" encoding="Shift_JIS" standalone="yes"?>
<kougi>
  <title begin="二千一年三月二十一日三時三十分＋九時">選挙義務講義ＸＭＬ</title>
  <organization>アウグスブルグ大学コンピュータ・サイエンス過分、工学と家政と形成の大学</organization>
</kougi>
```

Unicode stellt einen Industriestandard (entwickelt u.a. durch Apple, HP, IBM, Microsoft und SUN) zur Darstellung verschiedenster Alphabete und graphischer Zeichen dar. Sein zunächst durch 16-Bit codierter Zeichenvorrat bot Raum für 65536 unterschiedliche Symbole.
Die seit 1991 laufenden Unicodebemühungen münden in die ISO-Norm zur Erweiterung des klassischen ASCII-Codes (ISO 646) als ISO-10646 *Universal Multiple-Octet Coded Character Set (UCS)*. Seit 1996 sind beide Standards synchronisiert und werden abgestimmt vorangetrieben.
UCS definiert zwei aufeinander aufbauende Codierungen: UCS-2 (16 Bit Umfang) und UCS-4 (32 Bit). Der bisherige Unicode-Standard ist voll kompatibel zu UCS-2 und durch diesen darstellbar.

Tabelle 2: Verschiedene Codierungen des Zeichens "A"

Codierung	Bitbreite	Binärdarstellung	Größe der Beispieldatei in Byte (ohne Berücksichtigung des XML-Prologs)	Bemerkung zum Meßwert
UTF-7	>= 7	100 0001	263	(encoding="UTF-7")
Extended ASCII, Latin-1 (ISO-8859-1)	8	0100 0001	258	(encoding="ISO-8859-1")
UTF-8	>= 8	0100 0001	259	(encoding="UTF-8") keine Byte Order Mark
UCS-2, Unicode	16	0000 0000 0100 0001	516	(encoding="UCS-2") keine Byte Order Mark
UTF-16 (big endian)	>= 16	0000 0000 0100 0001	516	(encoding="UTF-16") keine Byte Order Mark
UCS-4	32	0000 0000 0000 0000 0000 0000 0100 0001	1032	(encoding="UTF-8") keine Byte Order Mark
UTF-32	>= 32	0000 0000 0000 0000 0000 0000 0100 0001	1032	(encoding="UTF-32") keine Byte Order Mark

Die Zeilenumbrüche wurden in allen Fällen durch die Kombination von Wagenrücklauf und Zeilenvorschub ausgedrückt.

Die Tabelle stellt einige Codierungen zur Darstellung des Zeichens *A* zusammen.
Auffallend ist der große Platzbedarf der UCS-2 und -4 Codierungen. Insbesondere bei den „klassischen“ ASCII-Symbolen werden hier (u.U. sehr viele) führende Nullbits erzeugt, die in der Konsequenz zu einer deutlichen Vergrößerung der Beispieldatei führen.
Daher wurde mit dem *UCS Transformation Format* (UTF) eine kompaktere Darstellung zum jeweiligen UCS-Set eingeführt. UTF-8 verwendet standardmäßig die ersten acht Bit zur Darstellung der bekannten ASCII-Zeichen

Anmerkung: Inzwischen existiert auch eine „UTF-32“ genannte 32-Bit Ausprägung, diese ist jedoch identisch zu UCS-4, mit Ausnahme daß durch UTF-32 „nur“ 2^{21}-Zeichen dargestellt werden können.
Die Dateigröße ist daher für das betrachtete Beispiel in dieser Darstellungsweise unverändert zu der des UCS-4-Encodings.

Der Größenunterschied zwischen der UTF-7 codierten Datei und der Latin-1 encodierten erklärt sich aus der Darstellung des Umlautes sowie des +-Zeichens, die beide nicht nicht im klassischen 7-Bit ASCII-Code enthalten ist.

So wird *Ü* im Wort *Übungsbetrieb* des Beispieldokumentes durch die die Bytefolge `2B 41 4E 77 2D` dargestellt, während alle übrigen Zeichen durch ein einzelnes Byte ausgedrückt werden können.
UTF-8 ist in der Lage sämtliche Standard-ASCII-Zeichen durch jeweils genau ein Byte auszudrücken, wiederum für den Umlaut muß auf die 16-Bit-Darstellung des UCS-2 zurückgegriffen werden. Daher erhöht sich hier die Dateigröße um ein Byte.
Erwartungsgemäß beträgt der Umfang des UCS-2 codierten Dokuments exakt das Doppelte des 8-Bit Äquivalents der Latin-1-Darstellung.
Dasselbe gilt für die UTF-16-Variante, die für das vorliegende Beispiel unterschiedslos zu UCS-4 verläuft, da keinerlei Zeichen aus UCS-4 im Dokument auftreten.

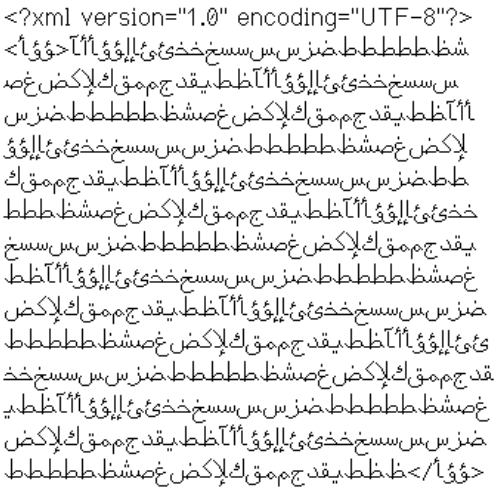
Die nachfolgende Tabelle stellt beispielhaft die Anwendung der UTF-8-Codierung zusammen:

Tabelle 3: UTF-8 Codierung

Unicode-Bereich	Bitbelegung
U-00000000 - U-0000007F:	0xxxxxxx
U-00000080 - U-000007FF:	110xxxxx 10xxxxxx
U-00000800 - U-0000FFFF:	1110xxxx 10xxxxxx 10xxxxxx
U-00010000 - U-001FFFFF:	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
U-00200000 - U-03FFFFFF:	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
U-04000000 - U-7FFFFFFF:	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Diese Mimik zeigt den Nachteil des UTF-*n*-Encodings deutlich: Die Darstellung nicht *n*-Bit darstellbarer Zeichen benötigt u.U. mehr Bitstellen als im Standard UCS-Code.
So wird beispielsweise das Zeichen mit der größtmöglichen Position (`7FFFFFFF`) in UTF durch sechs Byte encodiert, während UCS dieselbe Information mit den verfügbaren 32-Bit ausdrücken kann. Andererseits „verschwendet" die UCS-Darstellung für die niederwertigen Zeichen Bitstellen durch die führenden Nullen.

In der Praxis gilt es daher für das zu wählende Encoding einen möglichst guten Kompromiß zu finden: Im allgemeinen stellt das UTF-8-Encoding einen solchen dar, soweit überwiegend ASCII-Zeichen, und nur vereinzelt Sonderzeichen (hierzu zählen auch die deutschen Umlaute) eingesetzt werden.
Bei überwiegender Verwendung nicht in acht-Bit ASCII darstellbarer Zeichen (z.B. arabischer, chinesischer, etc.) erhöht die dann aufwendigere UTF-8-Codierung die Datenmenge.
So umfaßt die UTF-16-Darstellung des unten abgebildeten Beispieldokuments, welche in diesem Anwendungsfall identisch zu UCS-2 ist, 966 Bytes, während UTF-8 1299 Byte benötigt.



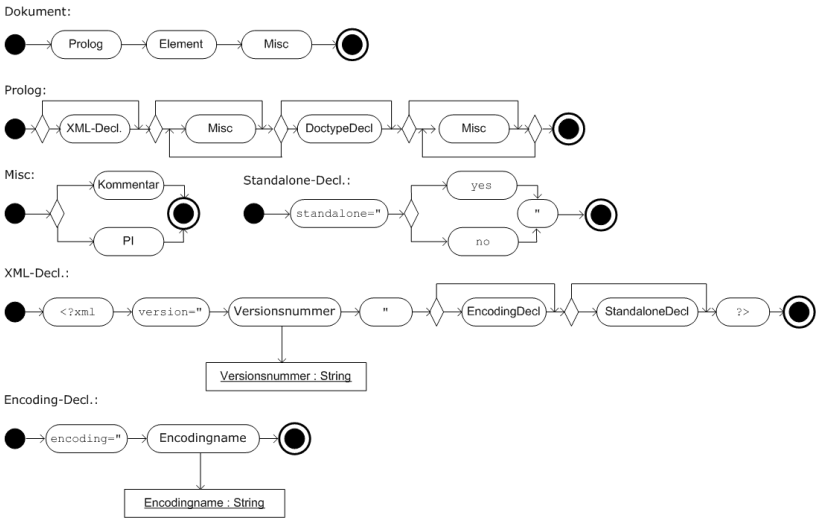
Achtung: Bereits durch die Unterstützung der beiden ISO-Zeichendarstellungen UTF-8 und UTF-16 ist die Konformität zum XML-Standard erfüllt! XML-Prozessorimplementierungen wird nicht abverlangt darüberhinausgehend weitere Darstellungen umzusetzen. (In XML-Spezifikation nachschlagen)

Wie bereits eingangs angemerkt, erklärt die XML-Spezifikation die Encodingdeklaration sowie den gesamten Prolog-Ausdruck als optionales Element (In XML-Spezifikation nachschlagen) .
Als Konsequenz geht dabei (auch) die Angabe des gewählten Encodings verloren.
Daher fordert der Anhang F der XML-Spezifikation *Autodetection of Character Encodings* bei einem von UTF-8 oder -16 abweichendem Codierungsschema die zwingende Angabe der XML-Deklaration (`<?xml ...`) (In XML-Spezifikation nachschlagen) .
Hintergrund dieser Maßnahme ist der Versuch anhand der damit bekannten fünf Zeichen das zugrundeliegende Encoding zu ermitteln.
Diese fünf Zeichen können als stabil angenommen werden, da Produktion 23 und 80 diese explizit von einem von UTF-8 oder -16 abweichenden Encoding ausnehmen.

Für Dokumente im deutschen Sprachraum, d.h. XML-Ströme die häuptsächlich aus den um die deutschen Umlaute ergänzten Standard-ASCII-Zeichen bestehen, hat es sich in der Vergangenheit eingebürgert den Zeichensatz latin-1 (`ISO-8859-1`) zu verwenden, um die Mehrbytedarstellung der Umlaute und weiterer Sonderzeichen in der UTF-Codierung zu umgehen.
Jedoch enthält der latin-1-Zeichensatz nicht das unter Unicode-Zeichennummer 20AC abgelegte Eurosymbol (_) welches zur Abkürzung des Währungsbegriffes der europäischen Gemeinschaftswährung verwendet wird.
Dieses Symbol wurde in die unter Nummer 15 veröffentlichte aktualisierte Fassung der Zeichensatzfamilie 8859 aufgenommen. Daher sollte bei der Erstellung von XML-Dokumenten generell darauf geachtet werden entweder `ISO-`

8859-15 als Codierung zu wählen oder auf die ohnehin ungleich flexiblere UTF-Codierung zurückzugreifen.

Die Darstellung der Abbildung 6 faßt die syntaktischen Elemente abgekürzt zusammen:



Web-Referenzen 4: Weiterführende Links
•Payer, M.: *UNICODE, ISO/IEC 10646, UCS, UTF*
•Kuhn, M.: *UTF-8 and Unicode FAQ*
•SC Unipad ein kostenfreier Unicode Editor

Element Information Item

Jedes XML-Dokument enthält mindestens ein *Element*, das `Document Element`.
Seine, wie auch die Grenzen aller anderen Elemente, werden durch die *Start-* und *Ende-Marke* (engl. *Tag*) markiert. Für den Sonderfall eines *leeren Elements* bildet die Start- auch zugleich die Ende-Marke. Als eine Konsequenz können diese Elemente keine weiteren Kindknoten besitzen.

Die XML-Spezifikation legt den Aufbau des Start-Tags wie folgt fest (In XML-Spezifikation nachschlagen) :

```
[40] STag      ::= '<' Name (S Attribute)* S? '>'
[41] Attribute ::= Name Eq AttValue
```

Mittels der Tag-Namen werden die Typen eines Dokumentes definiert. Sie werden später, in Verbindung mit einem Grammatikmechanismus wie DTD oder XML-Schema, zur Gültigkeitsprüfung herangezogen.
Der Aufbau der Elementnamen ist ähnlich zu den aus den Programmiersprachen bekannten Regeln. Am Beginn muß ein Buchstabe, ein Unterstrich oder der Doppelpunkt stehen. Darauf können nahezu beliebige Zeichen folgen, die über ihre Unicoderepräsentation genau definiert sind.
Leerzeichen und sog. *white spaces* (vgl. Produktion 3 der XML-Spezifikation) wie Tabulatoren und Zeilenvorschübe sind nicht zugelassen. Desweiteren darf ein Elementname weder Auszeichnungssymbole, wie die öffnenden und schließenden Winkelklammern, enthalten, noch mit der Zeichenkette *XML* beginnen. Die Zeichenfolge *XML* ist -- in allen Schreibweisen -- für die Standardisierung reserviert und wird ausschließlich in W3C-Dokumenten verwendet.
Durch den Namespace Standard (siehe Abschnitt 1.3) wird dem Doppelpunkt, als Trennsymbol zwischen Namensraumkürzel und Elementnamen, eine besondere semantische Bedeutung zugeschrieben. Daher sollte -- obwohl er spezifikationsgemäß ein erlaubtes Zeichen darstellt -- von seiner Verwendung in Elementnamen abgesehen werden.

Oftmals wird -- insbesondere in der Praxis -- die existierende und notwendige Unterscheidung zwischen *Tag* und *Element* nicht getroffen.
Die Tags oder Marken drücken beschreibende Information über ein Element aus. Der durch den Tag ausgedrückte Elementname liefert somit lediglich deskriptive Information über die Natur des Elements. Hierzu können Worte einer natürlichen Sprache verwendet werden, jedoch auch beliebige andere identifizierende Zeichenketten. Üblicherweise sind jedoch sprechende Tags anzutreffen.

Über den Tag-Namen hinaus kann ein Startelement auch noch *Attribute* enthalten (Vgl. Produktion 41). Diese sind jedoch nicht vom Typ *Element* und werden daher im Abschnitt *Attribute Information Item* betrachtet.

Der Aufbau eines Elementnamens wird durch die Produktionen 4ff definiert (In XML-Spezifikation nachschlagen) :

```
[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
[5] Name     ::= (Letter | '_' | ':') (NameChar)*
[6] Names    ::= Name (S Name)*
[7] Nmtoken  ::= (NameChar)+
[8] Nmtokens ::= Nmtoken (S Nmtoken)*
```

Im Beispiel sind `Vorlesung`, `Titel` und `Hochschule` („normale“) Elemente, während ein leeres Element darstellt.
Die Abbildung zeigt, daß auf der semantischen Ebene des Information Sets die syntaktische Unterscheidung zwischen Elementknoten mit Kindelementen und leeren Elementen des XML-Dokuments keine Berücksichtigung findet.

Eine Sonderstellung unter den Elementen eines Dokuments nimmt der ausgezeichnete Wurzelknoten ein, er wird auch durch das *Document Information Item* referenziert. Unterhalb dieses Knotens spannt sich der Dokumentbaum auf. Hierfür enthält jedes Element Information Item eine geordnete Menge (*children*) weiterer Elementknoten.
Die durch den Elementnamen verwirklichte Typisierung spiegelt sich im Information Set durch das Attribut *local name* wieder.

Darüberhinaus enthält jedes *Element Information Item* durch die Eigenschaft *namespace name* die Identifikation des Namensraumes, in dem dieses Element plaziert ist.
Das Namensraumkürzel, welches zur Identifikation eines Elements herangezogen wird, findet sich in der Eigenschaft *prefix*.
Der *local name* entspricht dem -- um Namensraumkürzel und trennenden Doppelpunkt gekürzten -- wiedergegebenen Elementnamen des XML-Dokuments.
Zusätzlich wird jeder Namensraum, der syntaktisch an die Attributdefinition angelehnt ist, in ein Element der ungeordneten Menge *namespace attributes* abgebildet, welche (nochmals) die Namensräume eines Elements beinhaltet.

Beispiel 2: Element mit deklariertem Namensraum

```
(1)...
(2)     <myNS:aParent xmlns:myNS="example.com">
(3)             <myNS:aElement/>
(4)     </myNS:aParent>
(5)...
```

Das Beispiel zeigt das leere Element `aElement` innerhalb des Elements `aParent`. Durch das Elternelement wird der Namensraum `example.com` deklariert und dem Kürzel `myNS` zugewiesen.
Gemäß den Prinzipien der Namensräume steht der auf dem Elternknoten deklarierte Namensraum auch in allen Kindknoten zur Verfügung. Daher enthält die Eigenschaft *in-scope namespaces* des Elements `aElement` auch die Namensräume der übergeordneten Elemente.
Das resultierende *Element Information Item* des Knotens `aElement` ergibt sich daher als (der Ausschnitt enthält nur die für das Beispiel relevanten Elemente):

```
local name    = aElement
namespace URI = example.com
prefix        = myNS
```

Nähere Ausführungen zur Bedeutung von Namensräumen und ihrer Verwendung finden sich im Abschnitt *Namensräume*.

Verweise auf die im Dokumentbaum nachfolgenden Knoten eines Elements werden in einer geordneten Liste *children* gesammelt. Ihre Inhalte sind sind vom Typ *Element Information Item*, *Unexpanded Entity Reference Information Item*, *Character Information Item* und *Comment Information Item*.
Anhand der beiden Informationstypen *Element Information Item* und *Character Information Item* zeigen sich bereits die beiden Strukturierungsformen eines XML-Dokuments. Einerseits die durch die starke Verwendung von Elementen- und Attributen gekennzeichnete strukturierte Darstellung, andererseits die durch „eingestreuten“ Freitext entstehende charakteristische semistrukturierte Variante.
In beiden Fällen werden die textartigen Inhalte durch *Character Information Items* repräsentiert.
Das Beispiel zeigt die verschiedenen Auftretensformen exemplarisch. Der Inhalt der Elemente `title` und `organization` ist rein Zeichenketten-artig; jedoch mischt `vorlesung` strukturierten Inhalt (in Form der genannten Elemente) und unstrukturierte Information -- repräsentiert durch den Text `2002/03`.
Die XML-Spezifikation prägt für Zeichenketten-artige Inhalte, die optional durch *eingestreute* Elemente angereichert werden, den Begriff *mixed Content*.

children enthält jedoch keine Verweise auf die Attribute eines Elements. Diese sind durch die separate ungeordnete Menge *attributes* repräsentiert. Die Diskussion der als *Attribute Information Item* bezeichneten Mengenelemente findet sich im folgenden.

Die in der Abbildung dargestellte Beziehung *parent* verbindet jedes Element mit seinem übergeordneten. Als Ziele dieser Referenz sind ausschließlich Ausprägungen von Document Information Item oder Element Information Item zugelassen.
Diese Festlegung untermauert nochmals die strikte Baumstruktur eines XML-Dokuments. Andernfalls müßte *parent* als Menge definiert werden.

Attribute Information Item

Das betrachtete Beispiel enthält, neben den Elementen, auch ein XML-Attribut.
Syntaktisch werden Attribute innerhalb eines Start-Tags plaziert und durch Namen-Wert-Paare ausgedrückt (In XML-Spezifikation nachschlagen) .

Der Information Set enthält folgende Eigenschaften zu jedem Attribut:

- **namespace name**: Namensraum des Attributs, falls definiert.
- **Lokaler Name**: Der um das eventuell definierte Namensraumkürzel bereinigte Attributname.
- **Präfix**: Namensraumkürzel des Namensraumes, innerhalb dessen das Attribut plaziert ist.
- **Normalisierter Wert**: Normalisierter Attributinhalt. Der Normalisierungsvorgang ist in Abschnitt 3.3.3 der XML-Spezifikation beschrieben (In XML-Spezifikation nachschlagen) .
  Unter anderem eliminiert er Zeilenumbrüche innerhalb des Attributinhalts und löst Entitätsreferenzen auf.

- **specified**: Boole'scher Wert, der angibt, ob das Attribut im XML-Dokument auftrat oder aufgrund einer Vorgabebelegung durch die DTD erzeugt wurde.
  Zur Ermittlung dieser Eigenschaft des Attribute Information Items ist die Definition und Referenzierung einer expliziten Grammatik notwendig.
- **Attributtyp**: Typ des Attributs. Zugelassene Belegungen sind: `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NMTOKEN`, `NMTOKENS`, `NOTATION`, `CDATA`, und `ENUMERATION`.
  Zur Ermittlung dieser Eigenschaft ist der Zugriff auf die DTD des Dokumentes notwendig. Ist dies nicht möglich, so ist der *attribute type* mit keinem Wert belegt.
- **Referenzen**: Handelt es sich bei dem Attribut um ein Referenzattribut (d.h. es ist als `IDREF(S)`, `ENTITY`, `ENTITIES` oder `NOTATION` typisiert), so enthält diese Eigenschaft eine Verweisliste auf alle Auftreten des Attributwertes.
- **Eigentümerelement**: Bildet die Entsprechung zur *parent*-Eigenschaft des *Element Information Item*. Als solches enthält die Eigenschaft einen Verweis auf das Element, welches das Attribut beherbergt.

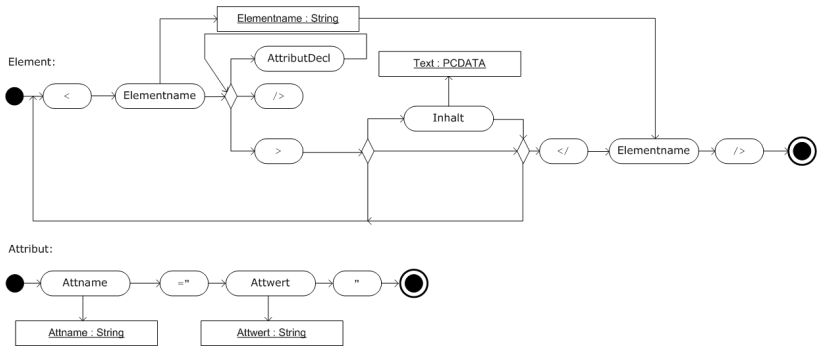
Im Vergleich zum *Element Information Item* erlaubt das Attribut keine weitere Unterstrukturierung (im XML-Sinne); insbesondere fehlen mengenwertige Eigenschaften zur Aufnahme der dann notwendigen Verweise. Stattdessen wird der gesamte Inhalt durch die Eigenschaft *normalized value* dargestellt.
Daher dürfen innerhalb von Attributen keine (Meta-)Symbole wie die öffnende Winkelklammer auftreten, die als Starttags (miß-)interpretiert werden könnten (In XML-Spezifikation nachschlagen) .

Auch die Form des Auftretens von Attributen innerhalb des definierenden Elements unterscheidet sich von der der Subelemente innerhalb eines Elements. Während Kindelemente durch die geordnete Liste *children* dargestellt werden, können Attribute (formalisiert in der ungeordneten Menge *attributes*) in beliebiger Reihenfolge angegeben werden, ohne die Dokumentsemantik zu verändern. Mehr noch, die Listenkonstruktion erlaubt das unterscheidbare mehrfache Auftreten desselben Elements. Diese Mimik ist für allgemeine Mengen, und damit für Attribute, nicht möglich.

Element vs. Attribut
Der Vergleich der Eigenschaften von Element und Attribut zeigt bereits, daß sich nicht weiter strukturierte Elemente auch durch Attribute darstellen ließen. Dies wirft innerhalb der Betrachtung der Syntax eines XML-Dokuments bereits die Frage nach der Organisation, und damit dem Entwurf, eines solchen auf.
Die bestehende XML-Spezifikation bleibt jedoch eine Anwendungs- oder Einsatzempfehlung zu dieser Fragestellung schuldig.
Aufgrund der inhärenten Einschränkungen der Attributprimitive bietet sich ihr Einsatz nur in einigen Sonderfällen an. Beispielsweise zur Darstellung deskriptiver Information über das enthaltende Element, die nicht Bestandteil der im XML-Dokument dargestellten Information ist. Hierbei kann es sich um Informationen höherer Ordnung, sog. Metainformation handeln.

Generell bieten sich Elemente immer dann an, wenn eine weitere Unterstrukturierung des Inhaltes gewünscht oder vielleicht zukünftig notwendig ist. Die Darstellungsform als Attribut würde in diesem Fall eine strukturelle Umorganisation des XML-Vokabulars erfordern, da die Spezifikation keine Unterstrukturierungsmöglichkeit für Attribute vorsieht.
Darüberhinaus gestatten Attribute keine Wiederverwendung in verschiedenen Bedeutungskontexten, da sie syntaktisch an das umgebende Element gebunden sind. Diese Einschränkung wird zwar durch die Einführung des Standards *XML Schema* weitgehend gemildert, jedoch nicht die zuvor genannte Mächtigkeitseinschränkung.
Zusätzlich stellen Attribute die einzige Möglichkeit zur Typisierung des Inhaltes dar solange DTDs verwendet werden. Dieser Punkt dürfte jedoch durch den wachsenden Praxiseinsatz der XML Schemata immer mehr an Bedeutung verlieren.

Die Darstellung der Abbildung 7 faßt die syntaktischen Elemente abgekürzt zusammen:



Character Information Item

Die Betrachtung der Attribut- und Elementknotentypen im Information Set zeigt bereits die zwei grundlegenden Arten der Informationsdarstellung eines XML-Dokumentbaumes.
Die Eigenschaft *normalized value* des *Attribute Information Item*s kapselt den im XML-Dokument angegebenen Inhalt direkt im Informationsknoten. Der Datentyp der Eigenschaft ist für alle Dokumenttypen fixiert angebbar, da keine weitere Unterstukturierung von Attributen erfolgen kann.
Entgegensetzt hierzu verläuft die Argumentationslinie für Elemente. Ihr Inhaltsmodell kann eine freie Mischung aus Zeichenketten-Daten und weiteren Elementen aufweisen. Die Länge der Zeichenketten ist hierbei nicht näher festgelegt. Daher können diese im minimalen Falle nur aus einem einzelnen Zeichen bestehen. (In XML-Spezifikation nachschlagen) .
Innerhalb des Information Sets eines Dokuments werden alle Zeichen im Rumpf eines Elements als Ausprägungen des *Character Information Item*s dargestellt.

Jedes Character Information Item stellt das im Dokument gegebene Zeichen gemäß ISO 10646-Codierung in der

Eigenschaft *character code* dar. Die Werte können hierbei jedoch nur in den durch die Spezifikation vorgegebenen Grenzen variieren (In XML-Spezifikation nachschlagen) . Darüberhinaus genügt bereits die Unterstützung der UTF-8 und -16-Darstellung zur Erfüllung der Spezifikationsanforderungen an konforme Prozessoren.
Handelt es sich bei dem betrachteten Zeichen um einen white-space (Leerzeichen, Tabulator, Zeilenvorschub, Wagenrücklauf), der in der DTD als „erhaltenswert" spezifiziert wurde, so ist die Boole'sche Eigenschaft *element content whitespace* auf `true` gesetzt; andernfalls konstant auf `false`. Häufig werden white-spaces zur besseren visuellen Strukturierung des XML-Dokumentes eingesetzt. So enthält das Beispieldokument jeweils nach der schließenden Marke einen Zeilenvorschub. Unter Datengesichtspunkten handelt es sich hierbei jedoch um keine verwertbare Information. Die Angabe der Berücksichtigung bzw. Vernachlässigung im XML-Dokument existierender white-spaces kann in der DTD gesetzt werden. Ist keine solche Deklaration gesetzt oder existiert keine explizite Grammatik, so hat die Eigenschaft *element content whitespace* keinen Inhaltswert.
Der als `parent`-Eigenschaft realisierte Verweis auf das beherbergende Elternelement bildet den Abschluß der Eigenschaften des *Character Information Items*.
Im betrachteten Beispiel sind unterhalb der Elemente `organization` und `title` *Character Information Element*-Ausprägungen plaziert. Die Darstellung zeigt diese als Objekte (Unterhalb des `organization`-Knotens wurde aus Übersichtlichkeitsgründen auf die Darstellung verzichtet).

Eine Sonderrolle kommt den Zeichen zu, die auch als Metasymbole der Auszeichnungssprache dienen. Sie dürfen daher nicht in XML-Dokumenten auftreten.
Bei diesen Zeichen handelt es sich um die beiden Winkelklammern, die einfachen und doppelten Anführungszeichen sowie das *Kaufmanns-Und*. Um eine Fehlinterpretation zu vermeiden existieren hierfür vordefinierte Textersetzungsmuster.
Jeder spezifikationskonforme XML-Prozessor berücksichtigt diese Symbole und gibt sie in der korrekten Darstellung an die Applikation weiter; damit sind diese Fluchtsymbole (engl. *escape characters*) aus Applikationssicht vollkommen transparent.

Tabelle 4: Vordefinierte Textersetzungsmuster

Entitätsreferenz	Ausgedrücktes Zeichen
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Web-Referenzen 5: Weiterführendes ... Die in XHTML v1.0 vordefinierten Entitäten
Latin-1 Entities
Special Entities
Symbole

Comment Information Item

Zur Dokumentation steht innerhalb jedes XML-Dokuments die von SGML ererbte Kommentierungssyntax zur Verfügung.
Die Spezifikation erlaubt die Anbringung von Kommentaren an zwei Stellen im XML-Dokument:

- Nach dem Prolog. (In XML-Spezifikation nachschlagen)
- An jeder beliebigen Stelle des Inhalts, außerhalb von Markup-Symbolen. (In XML-Spezifikation nachschlagen)

Nicht erlaubt sind demnach Kommentare in Tags, d.h. innerhalb geöffneter Winkelklammern.
Dergleichen gilt für Kommentare selbst, was geschachtelte Kommentare verbietet.

Ferner können Kommentare desselben Stils auch in DTDs und im internal Subset verwendet werden.

Produktion 15 der XML-Spezifikation legt die Struktur wie folgt fest:

```
[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

Als Konsequenz sind innerhalb von Kommentaren alle Zeichen, auch Metasprachensymbole, zugelassen. Somit ist das beliebige „auskommentieren" von Dokumentteilen möglich.
Als zentrale Einschränkung dürfen (aus SGML-Kompatibilitätsgründen) keine zwei aufeinanderfolgenden Trennstriche (*hyphen-minus*, ISO 10646 #x2D) innerhalb eines Kommentars auftreten, da diese fehlerhafterweise als Beginn des Kommentarendes interpretiert würden.

Der gesamte Inhalt eines Kommentars wird als uninterpretierte Zeichenkette in der Eigenschaft *content* des *Comment Information Item*s abgelegt.
Zusätzlich verweist jeder Kommentar über die bekannte *parent*-Eigenschaft auf seinen Elternknoten. Wie bereits durch die beiden Einsatzformen angedeutet, kann es sich hierbei ausschließlich um ein *Document Information Item* oder ein *Element Information Item* handeln.

Beispiel 3: Verschiedene Kommentarstrukturen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<Root>
(3)     <!-- this is a comment -->
(4)     <ElementA>
(5)             <ElementB>
(6)                     <!--
(7)                     <ElementC/>
(8)                     <ElementD att1="..."/>
(9)                     -->
(10)            </ElementB>
(11)    </ElementA>
(12)</Root>
```

Das Beispiel zeigt verschiedene Einsätze von Kommentaren. Zunächst eine einzeilige Anmerkung, die nur verschiedene Zeichen versammelt. Im Anschluß einen mehrzeiligen Kommentar, der auch XML-Strukturen beinhaltet. Ein prozessierender Zugriff auf den Kommentarinhalt ist jedoch nicht vorgesehen, und wird durch gängige Parser und APIs zumeist nicht unterstützt.

Processing Instruction Information Item

Im Gegensatz zu den prinzipiell in beliebigem Freitext formulierbaren Kommentaren, die üblicherweise zur Kommunikation mit einem menschlichen Leser des XML-Dokuments dienen, zielt die *Processing Instruction* und das zugehörige Element des Information Sets auf Kommentare, welche einen maschinellen Verarbeiter des XML-Dokuments, den XML-Prozessor, betreffen.

Im Grunde genommen läuft die Anreicherung eines XML-Dokuments mit Verarbeitungsinformation der Idee einer deskriptiven Auszeichnungssprache entgegen ...
Jedoch wurde für die XML beschlossen, nicht zuletzt aus Kompatibilitätsgründen zu SGML, dieses Sprachmerkmal beizubehalten. Eine mögliche weitere Erklärung könnte das syntaktische Aussehen der XML-Deklaration innerhalb des des Dokumentprologs sein. Ihre in Produktion 23ff festgelegte Struktur stellt eine Anwendung der Processing Instruction dar, auch wenn dies innerhalb der Spezifikation nicht explizit formuliert wird.

Die Syntax einer *Processing Instruction* lautet:

```
[16] PI       ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

Eine Processing Instruction wird demnach immer durch eine öffnende Winkelklammer und ein folgendes Fragezeichen eingeleitet. Daran schließt sich die Benennung der Applikation an, für die diese Instruktion eingefügt wurde. Optional können weitere Zeichen -- ausgenommen der Kombination aus Fragezeichen und schließender Winkelklammer -- folgen.
Das adressierte System kann beliebig identifiziert werden, jedoch ist die Zeichenkette *XML* in allen Variationen ausgeschlossen.
Unbedachterweise verbietet die Spezifikation jedoch nicht die Bildung von Namen, die *XML* als Präfix nutzen ...
Jedoch sollte von der Nutzung solcher Konstruktionen abgesehen werden, da sie zur Verwirrung der (menschlichen) Leser beitragen.

Wie Kommentare auch können Processing Instructions an beliebiger Stelle innerhalb des XML-Dokuments auftreten: Vor Beginn des Wurzelelements sowie im Rumpf jedes Elements. Nicht gestattet ist ihre Angabe in Elementnamen und Attributen.
Ergänzend sei angemerkt, daß die Angabe von Processing Instructions auch innerhalb der *Document Type Definition* erfolgen kann. (siehe *Document Type Definition Information Item*).

Beispiel 4: Verschiedene Processing Instructions

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<?mySystem value="42"?>
(3)<root>
(4)   <?System2?>
(5)   <elementA>
(6)     <?System3 a="1" anotherValue?>
(7)  </elementA>
(8)</root>
```

Download des Beispiels

Übung 1: Processing Instructions
Begründen Sie mit Hilfe der XML-Spezifikation warum Processing Instructions nicht innerhalb von Elementen und Attributen zugelassen sind.
Hinweis: Es gibt mehr als eine Begründung!

Das *Processing Instruction Information Item* enthält die angesprochene Zielapplikation als Namen innerhalb der Eigenschaft *target*.
Der weitere Inhalt der Deklaration wird uninterpretiert als Zeichenkette in die Eigenschaft *content* übernommen.
Neben einem Verweis auf die Basis-URI der Processing Instruction wird durch *parent* das Elternelement -- entweder

ein Knoten des Typs *Document Information Item* oder *Element Information Item* -- referenziert.

Zur Formalisierung der Identifikation der Zielapplikation empfiehlt die XML-Spezifikation die Verwendung des Sprachmittels *Notation*.

Notation Information Item

Notationen können nicht in XML-Dokumenten auftreten, sondern sind DTDs vorbehalten.
Dennoch sollen sie hier wegen ihrer Berücksichtigung im Information Set und der Verbindung zu den *Processing Instructions* kurz eingeführt werden.

Durch Notations können Dateninhalten externe Quellen (etwa: URLs oder Applikationen) zugeordnet werden. Der Grundgedanke liegt in der Erweiterung der Ausdrucksmächtigkeit eines XML-Dokuments. Parser berücksichtigen solch typisierte Inhalte jedoch nicht. Die Behandlung muß (proprietär) in der datenverarbeitenden Applikation erfolgen.
Die Syntax der Notation lautet:

```
[82] NotationDecl ::= '<!NOTATION' S Name S (ExternalID | PublicID) S? '>'
[83] PublicID     ::= 'PUBLIC' S PubidLiteral
[75] ExternalID   ::= 'SYSTEM' S SystemLiteral
                    | 'PUBLIC' S PubidLiteral S SystemLiteral
```

Die einfachst denkbare Anwendung von Notations zielt auf Informationen, deren Inhalte nicht durch den XML-Prozessor auf syntaktische Korrektheit geprüft werden. Dies sog. *unparsed entities* werden aus einem eineindeutigen Namen und einer freien Zeichenkette gebildet.
Die Zeichenkette bezeichnet primär die als URI (gemäß RFC 2396 und 2732) codierte Identifikation des verarbeitenden Systems. Im Falle standardisierter Inhalte kann auf einen *public identifier* zurückgegriffen werden. Üblich ist die Verwendung von *system identifier*n, die auf eine bekannte URI oder ein installiertes Anwendungsprogramm verweisen.

Beispiel 5: NOTATIONS

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<!DOCTYPE someContent [
(3)<!NOTATION isoDate SYSTEM "http://www.iso.ch/markete/8601.pdf">
(4)<!NOTATION gif PUBLIC
(5)    "-//Compuserve Information Services//NOTATION Graphics
Interchange
(6)        Format//EN">
(7)<!NOTATION hex SYSTEM "hexEditor.exe">
(8)<!ELEMENT someContent (#PCDATA)>
(9)<!ATTLIST someContent format NOTATION (hex) #IMPLIED> ]>
(10)<someContent format="hex">4C 65 74 20 61 6C 6C 20 68 6F 70 65 20 61
(11)62 61 6E 64 6F 6E 2C 20 79 65 20 77 68 6F 20 6D 61 79 20 65 6E 74
(12)65 72 20 68 65 72 65 21</someContent>
```

Download des Beispiels

Hinweis: Die Deklarationen in DTD-Syntax -- erkennbar am einleitenden Ausrufezeichen (!) -- sind *kein XML*! Sie müssen daher nicht den syntaktischen Konventionen der Extensible Markup Language genügen!

Im Beispiel wird nach der *DOCTYPE*-Deklaration DTD-Syntax verwendet, um das Inhaltsmodell des nachfolgenden XML-Dokuments festzulegen. Die genaue Bedeutung der DTD-Konstrukte wird im Abschnitt 1.4 behandelt.
Das Beispiel beinhaltet drei Notationselemente: `isoDate`, `gif` und hex. `isoDate` referenziert die internationale Norm 8601 als PDF-Dokument.
Die durch hex definierte *NOTATION* wird im XML-Dokument verwendet.
Diese Anwendung offenbart auch einen weiteren Aspekt dieses Konstrukts; die Nachbildung von Datentyp-ähnlichen Strukturen, die jedoch durch den Anwendungsprogrammierer umgesetzt werden müssen.
Im Vergleich der beiden *SYSTEM*-Referenzen zeigt sich die Allgemeinheit des URI-Mechanismus als Schwäche. Da anhand der URI nicht ermittelbar ist, ob es sich bei der identifizierten Lokation um ein Dokument, ein System oder ein ausführbares Programm handelt, läßt sich keine verbindliche Semantik zur Behandlung von *NOTATION*-Elementen angeben.
Die mit `gif` benannte *Notation* definiert einen *Public*-Identifier, der nicht auf eine URI verweist, sondern eine weltweit eindeutige Benennung darstellt. Die Abbildung auf eine Systemlokation zur Behandlung der Notation kann durch die zusätzliche Angabe eines System-Identifiers erfolgen. Liegt eine solche nicht vor, muß der XML-Prozessor für die Abbildung Sorge tragen.
Ein weiteres Beispiel für die Verwendung des *NOTATION*-Konstrukts findet sich im Zusammenspiel mit ungeprüften Entitäten.

Jedes *NOTATION*-Element wird im Information Set durch ein *Notation Information Item* repräsentiert. Es enthält neben dem Namen (Eigenschaft *name*) die beiden möglichen Identifierinhalte in den Eigenschaften *system identifier* bzw. *public identifier*. Beide Eigenschaften sind Zeichenketten-artig; lediglich die URI-spezifische Inhaltsnormalisierung wird (wie in der XML-Spezifikation beschrieben) durchgeführt.

Document Type Declaration Information Item

Dokumente, zu denen eine -- in der Sprache DTD formulierte -- explizite Grammatik existiert, verfügen über eine *DOCTYPE*-Deklaration, welche den durch URI identifizierten Ablageort der DTD enthält.

Die Charakteristika dieser Information sind im *Document Type Declaration Information Item* zusammengefaßt. Es enthält alle Informationen, die im XML-Dokument über die DTD vorliegen. Daher bietet es keinen Zugriff auf die durch die DTD festgelegte Grammatik!
Im Einzelnen bietet der Information Set Knotentyp:

- **system identifier**: Verweis auf die externe DTD (*external subset*), falls vorhanden.
- **public identifier**: Verweis auf eine externe DTD, die durch einen *public identifier* identifiziert wird.
- **children**: Ungeordnete Menge, die Verweise auf *Processing Instruction Information Item*s enthält, die innerhalb der DTD definiert wurden.
- **parent**: Der Verweis auf das übergeordnete *Document Information Item*

Das Beispiel zeigt zwei *DOCTYPE*-Deklarationen. Zunächst eine SYSTEM-Deklaration, welche die Datei `abc.dtd` im aktuellen Dateisystemkatalog über eine URI referenziert.
Die zweite Deklaration zeigt die HTML v4.01-Standarddeklaration. Für sie existiert der dargestellte PUBLIC-Identifer. Zusätzlich wurde ein SYSTEM-Identifier (er folgt direkt ohne Schlüsselwort auf den PUBLIC-Identifer) angegeben, der, per URI, auf das DTD-Dokument beim W3C verweist.

Beispiel 6: Verschiedene DOCTYPE-Deklarationen

```
(1)<!DOCTYPE someContent SYSTEM "abc.dtd">
(2)<!DOCTYPE HTML PUBLIC
(3)     "-//W3C//DTD HTML 4.01//EN"
(4)     "http://www.w3.org/TR/html4/strict.
dtd">
```

Unexpanded Entity Reference Information Item

Im Abschnitt über das *Character Information Item* wurden bereits die Vordefinierten Textersetzungsmuster eingeführt, um die als Metasymbole dienenden Zeichen auch im Informationsbereich von XML-Dokumenten verwenden zu können.
Das für die fünf vordefinierten Symbole gewählte Verfahren steht auch dem Anwender zur Definition eigener Ersetzungsmuster offen. Die Definition kann ausschließlich in der Document Type Definition, oder dem DTD-Abschnitt eines Dokuments, erfolgen.
Diese Muster werden als *Entitäten* (engl. *entities*) bezeichnet.

Die Syntax ist wie folgt festgelegt (In XML-Spezifikation nachschlagen)

```
[71] GEDecl      ::= '<!ENTITY' S  Name  S  EntityDef S? '>'
[4]  NameChar    ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar | Extender
[73] EntityDef   ::= EntityValue | (ExternalID  NDataDecl?)
[9]  EntityValue ::= '"' ([^%&"] | PEReference | Reference)* '"'
                   | "'" ([^%&'] | PEReference | Reference)* "'"
[76] NDataDecl   ::= S  'NDATA' S  Name
```

Aus der Syntax ergeben sich zwei Klassen von Entitäten: Die durch eine URI und evtl. öffentlichen Namen identifizierten *externen Entitäten*, und die *internen Entitäten*, deren Inhaltsdefinition direkt rechts neben dem Schlüsselwort `ENTITY` gegen ist.
Die Verwendung von Entitäten kann an jeder Stelle des Informationsbereichs eines XML-Dokuments gestattet werden. Daher können Element- und Attributnamen keine Entitätsreferenzen enthalten.
Der Einsatz erfolgt, wie für die vordefinierten entities gezeigt, durch namentliche Referenzierung mit einem vorgestellten Kaufmanns-Und und einem abschließenden Semikolon.

Referenzierungssyntax: & *entityName* ;

Beispiel 7: Einige Entitätsdefinitionen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<!DOCTYPE someContent [
(3)     <!ELEMENT someContent ANY>
(4)     <!ENTITY entA "xyz">
(5)     <!ENTITY entB SYSTEM "http://www.jeckle.de/vorlesung/xml/script.html">
(6)     <!ENTITY entC PUBLIC "-//FHA//Symbol//DE" "file://symbols">
(7)]>
(8)<someContent>
(9)&entA;
(10)&entB;
(11)&entC;
(12)</someContent>
```

Download des Beispiels

Das Beispiel zeigt die drei möglichen Ausprägungen der Entitätsdefinitionen. `entA` deklariert eine interne Entität, deren Auftreten durch die Zeichenfolge abc ersetzt wird.
Durch `entB` und `entC` werden externe Quellen angesprochen. Die verwendete Syntax ist dabei zu der bei `NOTATION`s eingeführten identisch. Während `entB` ausschließlich eine lokal bekannte Referenz darstellt, bedient sich `entC` des *PUBLIC*-Identifiers um darüberhinaus auch eine öffentlich bekannte Quelle anzusprechen.
Im Dokument werden alle drei Entitäten über den selben Syntaxmechanismus eingebunden.

Innerhalb des Information Sets werden nicht-expandierte Entitäten durch das Element *Unexpanded Entity Reference Information Item* repräsentiert (für alle expandierten ist ihre Herkunft vollkommen transparent und für die Applikation nicht von Interesse).
Es definiert folgende Eigenschaften:

- **name**: Name der Entität.
- **system identifier**: Falls vorhanden, der angegebene *SYSTEM*-Identifier.
  (Im Beispiel: `http://www.jeckle.de/vorlesung/xml/script.html` bzw. `file://symbols`)
- **public identifier**: Der öffentliche Identifier, der zuvor entsprechend Abschnitt 4.2.2 der XML-Spezifikation bearbeitet wurde.
  (Im Beispiel: `-//FHA//Symbol//DE`)
- **parent**: Das *Element Information Item*, welches die Entität enthält.

Unparsed Entity Information Item

Mit der XML-Syntaxproduktion 76 wird als zusätzlicher Entitätstyp die explizit nicht zu prüfende Entität (engl. *unparsed entity*) eingeführt. Sie kann beispielsweise dazu verwendet werden, um Binärdaten wie Bilder, Videos, o.ä. aus einem XML-Dokument zu referenzieren.

Beispiel 8: Eine ungeprüfte Entität

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<!DOCTYPE someContent [
(3)     <!ELEMENT someContent ANY>
(4)     <!NOTATION gif PUBLIC
(5)    "-//Compuserve Information Services//NOTATION Graphics
Interchange
(6)        Format//EN">
(7)     <!ENTITY entA SYSTEM "xyz.gif" NDATA gif>
(8)]>
(9)<someContent>
(10)&entA;
(11)</someContent>
```

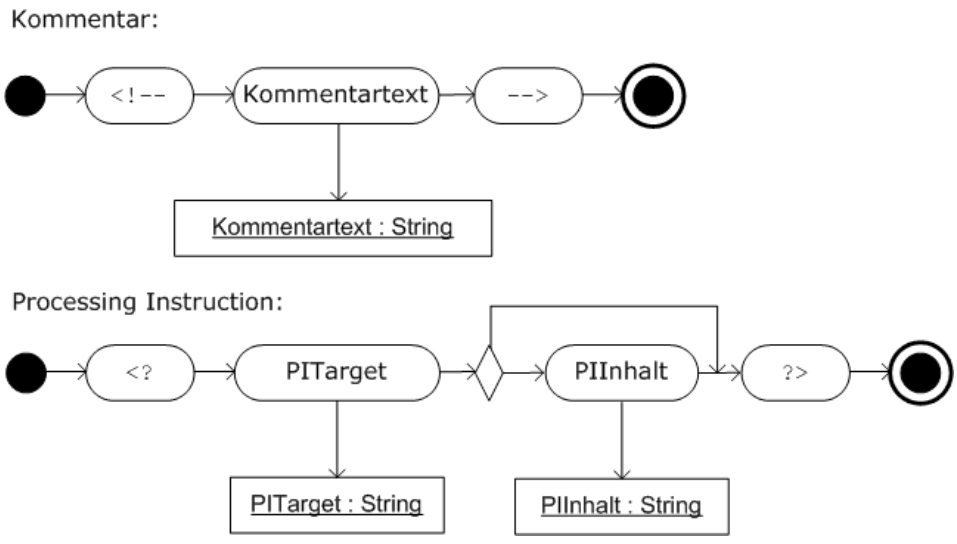
Download des Beispiels

Das Beispiel definiert zunächst die *NOTATION* für das Graphikformat *GIF* über dessen *PUBLIC*-Identifier. Diese wird innerhalb der Entität `entA` zur Definition der ungeprüften Inhaltsreferenz auf die Datei `xyz.gif` herangezogen.

Seitens des Information Sets stellt das *Unparsed Entity Information Item* eine Abwandlung der im vorhergehenden betrachteten Entitätsreferenz dar.
Im Einzelnen definiert der Information Set die Eigenschaften:

- **name**: Name der Entität.
- **system identifier**: Falls vorhanden, der angegebene *SYSTEM*-Identifier.
  (Im Beispiel: `xyz.gif`)
- **public identifier**: Der öffentliche Identifier, der zuvor entsprechend Abschnitt 4.2.2 der XML-Spezifikation bearbeitet wurde.
  (Im Beispiel: `-//Compuserve Information Services//NOTATION Graphics Interchange Format//EN`)
- **notation**: Referenz auf das *Notation Information Item*, welches auf die Inhaltsdefinition der Entität verweist.

Die Darstellung der Abbildung 8 faßt die syntaktischen Elemente abgekürzt zusammen:



Namespace Deklaration Information Item

Jedem im XML-Dokument definierten Namensraum ist ein *Namespace Deklaration Information Item* zugeordnet. Es enthält die notwendigen syntaktischen Details zur Identifikation des Namensraumes:

- **prefix**: Das gewählte Präfix des Namensraumes, bzw. leer falls es sich um den Vorgabenamensraum handelt.
- **namespace name**: Der Name des Namensraumes, an den das Präfix gebunden ist.

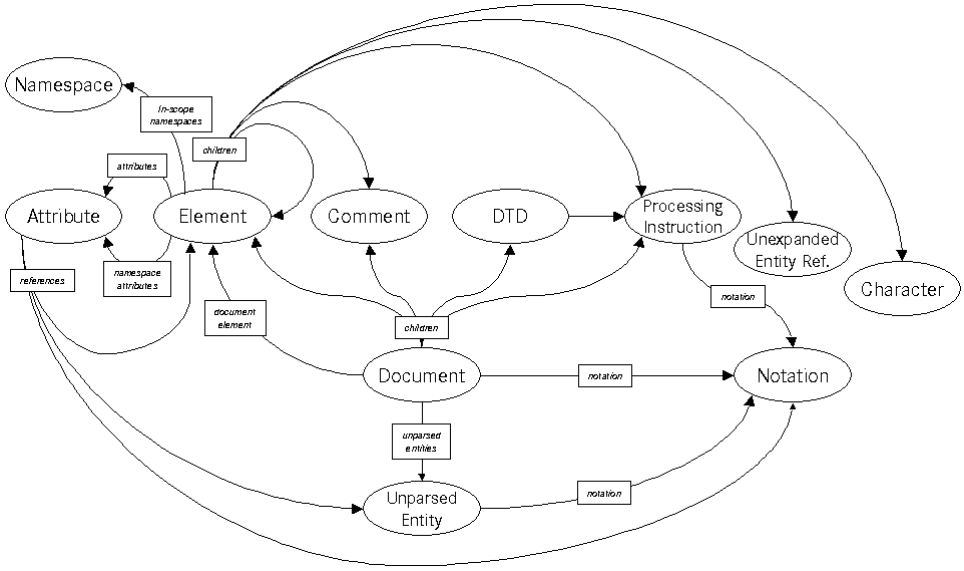
Beispiel 9: Beispiel eines Dokuments mit Namensräumen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<root>
(3)     <elementA>...</elementA>
(4)     <elementB xmlns="http://www.fh-furtwangen.de">...</elementB>
(5)     <elementC xmlns:abc="http://www.xyz.com">
(6)             ...
(7)             <abc:elementD/>
(8)     </elementC>
(9)</root>
```

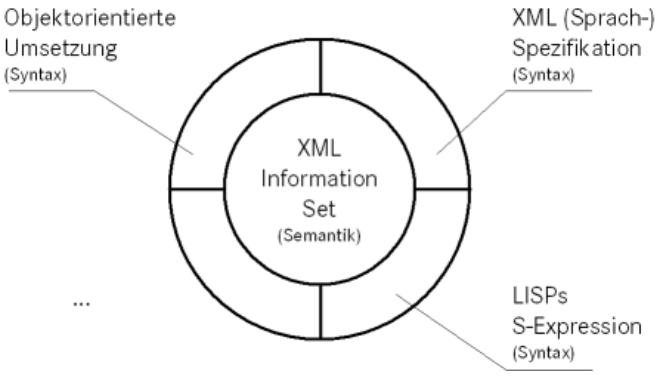
Für das Beispiel lauten die Namensräume wie folgt:

Elementname	Namensraum
root	(Das Element befindet sich im leeren Namensraum)
elementA	(Das Element befindet sich im leeren Namensraum)
elementB	http://www.fh-furtwangen.de
elementC	(Das Element befindet sich im leeren Namensraum)
elementD	http://www.xyz.com

Eine ausführliche Betrachtung zur Verwendung von Namensräumen findet sich im entsprechenden Abschnitt.



Die Graphik der Abbildung 9 stellt alle diskutierten Elemente des Information Sets in der Übersicht mit ihren Beziehungen dar. Zur Veranschaulichung wurde eine einfache Graphenstruktur gewählt, die alle Informationseinheiten als Knoten (darstellt als Ellipsen) und alle zugelassenen Beziehungen als gerichtete Kanten zwischen diesen enthält. Zusätzlich ist an die Kanten die Art der Beziehung angetragen.
Den Ausgangspunkt der baumartigen Struktur eines XML-Dokuments bildet die im Zentrum abgebildete Primitive `Document Information Item`, die alle weiteren Inhalte eines Dokuments über die `children`-Kante als Kindknoten enthält. Ferner fällt in dieser Darstellung besonders auf, daß lediglich `Element Information Items` über weitere Kindknoten verfügen und so die charakteristische XML-Struktur herausbilden. Alle übrigen Primitive dienen überwiegend als Blattknoten des Baumes.



Die Graphik der Abbildung 10 setzt die durch den Infoset-Standard definierte Semantik und die darauf aufsetzenden Syntaxen in Beziehung. Der XML-Basisstandard definiert hierbei nur eine von mehreren möglichen Syntaxen zur Darstellung von Infoset-Ausprägungen. Ebenso denkbar wäre der Einsatz anderer Darstellungen gleicher Mächtigkeit wie beispielsweise der S-Expression aus LISP oder objektorientierte Umsetzungen.

Auf Basis der Definitionen des Information Sets läßt sich ein beliebiges XML-Dokument, welches den Strukturierungsprinzipien des Infosets folgt, als *wohlgeformt* (*well-formed*) charakterisieren.

Definition 5: Wohlgeformtes XML-Dokument

Ein textartiges Objekt, dessen Inhalt folgenden Anforderungen genügt:

- Das XML-Dokument nutzt eine DTD, oder enthält die Deklaration `standalone="yes"`
- Zu jedem Start-Tag existiert genau ein Ende-Tag.
  Bei leeren Elementen können diese zu einem Tag zusammenfallen.
- Korrekte Elementschachtelung, d.h. Elemente überlappen einander nicht.
- Genau ein Wurzelelement.
- Alle Attributwerte sind in einfachen oder doppelten Anführungszeichen.
- Kein Start-Tag (oder Tag der ein leeres Element einleitet) enthält zwei oder mehr Attribute desselben Namens.
- Keine Kommentare oder Processing Instructions innerhalb von Tags.
- Kommentare beginnen und enden mit genau zwei Bindestrichen.
- Die Sonderzeichen < und & treten nicht innerhalb von Elementinhalten oder Attributwerten auf.

siehe XML-Spezifikation

Der Textstrom des Beispiels 10 zeigt ein nicht-wohlgeformtes XML-Dokument, welches gegen eine Reihe der in Definition 5 verstößt:

Beispiel 10: Ein nicht wohl-geformtes XML-Dokument

```
(1)<?xml version="1.0"?>
(2)<root>
(3)     <elementA att=a oder b>
(4)             <elementB> iff a<b ==> ...
(5)     </elementA>
(6)     <elementC att1="42" att1="3.14">
(7)             <elementD <?do-something?> >
(8)     </elementC>
(9)             </elementD>
(10)    <!---- dies ist nicht erlaubt ---->
(11)</root>
```

Download des Beispiels

So findet sich in Zeile 3 ein nicht in die erforderlichen Anführungszeichen eingeschlossener Attributwert.
Der textuelle Elementinhalte des in Zeile 4 geöffneten Elements `elementB` enthält ein öffnendes Winkelklammersymbol, welches um Fehler während des Einlesevorganges zu vermeiden durch die alternative Zeichensequenz `<` hätte ersetzt werden müssen. Darüberhinaus fehlt das korrekte schließende Tag zum Öffnenden.
Innerhalb des Elements `elementC` der Zeile 6 wird zweifach ein identisch benanntes Attribut definiert.
Im öffnenden Tag des in Zeile 7 definierten Elements `elementD` findet sich eine -- dort nicht zugelassene -- Processing Instruction.
Überdies überlappen sich die Elementgrenzen der Elemente `elementC` und `elementD` und zusätzlich wird der in Zeile 10 plazierte Kommentar nicht durch die erforderlichen genau zwei Bindestriche eingegrenzt.

1.3 Namensräume

Die XML-Namensräume wurden schon verschiedentlich erwähnt. Sie bilden die wichtigste, und offensichtlichste Weiterentwicklung der XML-Urspezifikation seit ihrer Veröffentlichung.
Trotz ihrer engen Beziehung zum XML-Kernstandard bildet die Recommendation *Namespaces in XML* eine eigenständige Spezifikation. Aufgrund der engen syntaktischen Beziehung zum XML-Standard und der großen praktischen Bedeutung, sowie des Einflusses auf die weitere Entwicklung verschiedenster Sekundärstandards und XML-Sprachen, werden die Namensräume explizit in der Neuauflage des XML-Standards berücksichtigt. Einen Beleg hierfür bildet die Anmerkung zu Abschnitt 2.3 *Common Syntactic Constructs*. Dort wird von der -- laut Syntaxproduktion 5 erlaubten -- Verwendung des Doppelpunktes in Elementnamen abgeraten. Dies geschieht, um Mehrdeutigkeiten, oder schlichtweg der Verwirrung des Anwenders, vorzubeugen, da es sich beim Doppelpunkt um ein Symbol besonderer Bedeutung innerhalb der Namensraumdeklarationen handelt.

Warum Namensräume?

Die breite Entwicklung immer neuer XML-Sprachen führt zwangsläufig zu Mehrfachentwicklungen für ähnliche oder identische Problemstellungen. Technisch betrachtet äußerst sich dies -- bei natürlichsprachlicher Benennung der Elemente -- durch die Verwendung identischer Bezeichner in verschiedenen XML-Sprachen. Hierbei bilden die verschiedenen Sprachen Anwendungskontexte, innerhalb derer die Bezeichner, durch Einbezug der Anwendungssemantik, eindeutig sind; andernfalls kann unterstellt werden, daß bereits durch die Sprachentwicklung andere Benennungskonventionen gewählt worden wären.
In der Konsequenz der Verfügbarkeit verschiedenster XML-Sprachen für beliebige Anwendungsbereiche entsteht der (berechtigte) Wunsch existierende Sprachfragmente in eigene Sprachen zu integrieren, um so zeitraubenden und vielfach fehleranfälligen Mehrfachentwicklungen vorzubeugen. Jedoch tritt bei diesem Integrationsszenario die u. U. kontextabhängige Elementeindeutigkeit zu Tage.

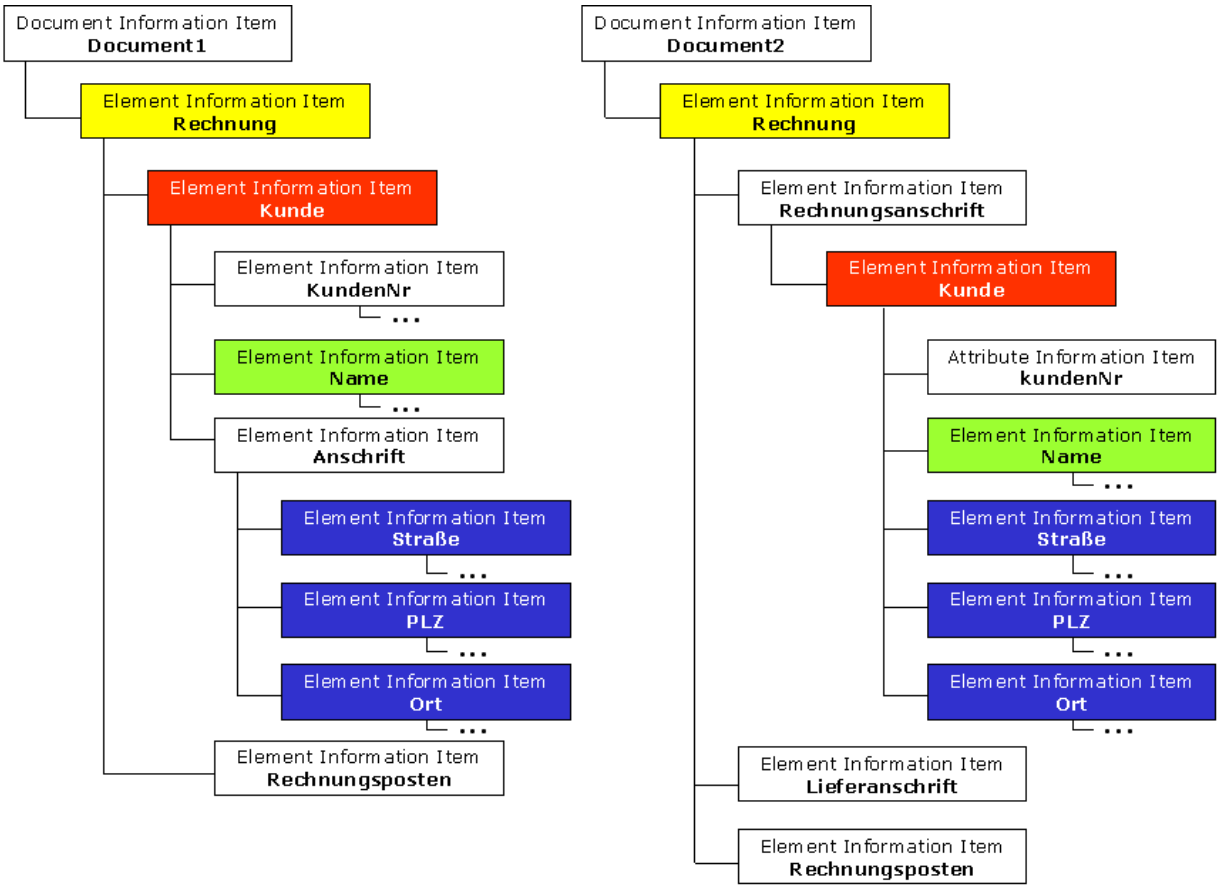
Das Beispiel zeigt zwei Dokumente identischen Informationsumfanges, die lediglich strukturell differieren.

Beispiel 11: Ein Rechnungsdokument

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<Rechnung>
(3)   <Kunde>
(4)      <KundenNr>4711</KundenNr>
(5)      <Name>Max Mustermann</Name>
(6)      <Anschrift>
(7)         <Straße>Musterplatz 1</Straße>
(8)         <PLZ>12345</PLZ>
(9)         <Ort>Musterstadt</Ort>
(10)      </Anschrift>
(11)   </Kunde>
(12)   <Rechnungsposten>
(13)    ...
(14)    </Rechnungsposten>
(15)</Rechnung>
```

Beispiel 12: Eine alternative Rechnungsstruktur

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<Rechnung>
(3)     <Rechnungsanschrift>
(4)             <Kunde kundenNr="4711">
(5)                     <Name>Max Mustermann</Name>
(6)                     <Straße>Musterplatz 1</Straße>
(7)                     <PLZ>12345</PLZ>
(8)                     <Ort>Musterstadt</Ort>
(9)             </Kunde>
(10)    </Rechnungsanschrift>
(11)    <Lieferanschrift>
(12)   ...
(13)   </Lieferanschrift>
(14)    <Rechnungsposten>
(15)    ...
(16)    </Rechnungsposten>
(17)</Rechnung>
```



Die beiden Bäume mit *Information Set*-Ausprägungen zeigen die Struktur der Beispieldokumente. Dabei sind Knoten die den selben Inhalt repräsentieren mit identischen Farben unterlegt, unabhängig davon um welchen Knotentyp es sich handelt. Die *Character Information Item* Knoten wurden aus Übersichtlichkeitsgründen weggelassen und durch Punkte angedeutet, sie sind jedoch für die vorliegende Betrachtung nicht von Interesse.

Einige der Elemente und Attribute werden in beiden Dokumenten mit gleichen Inhalten verwendet; z.B. `Name`, `Ort` oder `PLZ`. Dies äußert sich in identischen Teilbäumen unterhalb der Information Set-Knoten welche diese XML-Elemente repräsentieren. Hieraus läßt sich ableiten, daß die beiden vorgestellten Sprachen an den genannten Stellen keine strukturelle Differenz aufweisen.
Dagegen unterscheiden sich die Kindknoten der Elemente `Rechnung` und `Kunde` hinsichtlich ihrer Struktureigenschaften. So folgt im ersten Beispieldokument auf das `Rechnung`-Element direkt der `Kunde`, während im zweiten XML-Dokument zunächst ein Element mit dem Namen `Rechnungsanschrift` erwartet wird.
Dergleichen gilt für die Kindelemente des `Kunden`. Im zweiten Beispieldokument wird die diesem Element untergeordnete Kundennummer durch ein Attribut (`kundenNr`) dargestellt. Dagegen codiert das erste Beispiel diese Information direkt in den Elementinhalt.

Solange die beiden Dokumente in unterschiedlichen Anwendungswelten (Unternehmen o. ä.) verwendet werden, ist der gewählte Ansatz nicht problematisch. Bedenklich wird er jedoch in mindestens zweierlei Hinsicht:
Zunächst bei der „Mischung“ der beiden Dokumente. Dieser Wunsch tritt bei praktischen Problemstellungen häufig auf, wenn es um die Übernahme von XML-codierten Daten in ein anderes XML-Dokument geht. In der Konsequenz folgt das entstehende Zieldokument nicht mehr den Strukturierungsregeln eines der Ausgangsdokumente; mithin entsteht eine neue Dokumentstruktur, deren Regeln nicht explizit dokumentiert sind.
Eine weitaus größere Herausforderung stellt die Zusammenfassung und Veröffentlichung von XML-Strukturen in sog. Schemabibliotheken oder Datenbanken dar. Hier werden zwar die Dokumente nicht vereinigt, jedoch offenbart sich die gleiche Anwendungsdomäne (z.B. Rechnungsverwaltung, Stücklisten, Produktstrukturen) als problematisch, da sie die XML-Strukturen in direkte Konkurrenz treten läßt. In Zeiten immer stärker werdenden ökonomischen Flexibilisierungsdruckes erweist sich dies als äußerst kontraproduktiv, im Hinblick auf eine angestrebte Standardisierung. Die offene Konkurrenz verschiedener *Dialekte* innerhalb einer Domäne verzögert damit oft die Entscheidung zum Einsatz eines Sprachformates.

Einen anderen interessanten Anwendungsfall stellt der ausdrückliche Wunsch nach der Einbettung fremder Sprachelemente dar. Diese Form der Wiederverwendung knüpft an das durch öffentlich verfügbare XML-Formate eröffnete Anwendungsfeld an. Da nicht in jedem Fall ein alle Anforderungen erfüllendes existierendes XML-Format ermittelt werden kann, jedoch verschiedene vorhandene Formatteile des gewünschten Umfanges abdecken, entsteht der Wunsch nach einer selektiven Weiterverwendung. Ein bekanntes Beispiel bilden Freitexte in beliebigen XML-Sprachen, welche auf Teile des (X)HTML-Sprachumfanges zurückgreifen. Gleichzeitig ist damit die Semantik der Elemente durch den zugehörigen W3C-Standard festgelegt. XHTML selbst stellt ein interessantes Anwendungsbeispiel für die gemeinsame Verwendung verschiedener XML-Sprachen in einem Dokument dar. So können Web-Seiten neben den bekannten Textstrukturen (XHTML) auch mathematische Symbole und Formeln (in der XML-Sprache MathML) und Vektorgraphiken (in der XML-Sprache SVG) enthalten.
Als Nebeneffekt der Wiederverwendung existierender XML-Sprachen verringern sich mögliche Fehlerquellen, was in der Konsequenz zur Erhöhung der Qualität der entstehenden Sprachen führt.

Zusammenfassend lassen sich die (Hinter-)Gründe der Namensraumeinführung wie folgt darstellen:

- Wiederverwendung bestehender (fremder) XML-Strukturen in eigenen Dokumenten.
- Wunsch nach breiteren Standards.
- Verringerung des Designaufwandes.
- Nutzung bereits gesammelter Designerfahrung.
- Zusammenführung verschiedener XML-codierter Inhalte (*heterogeneous content syndication*).

Definition 6: Namensräume

XML-Namensräume stellen eine XML-basierte Syntax zur Verfügung um Element- und Attributnamen eines Vokabulars eindeutig zu identifizieren und so Bedeutungsüberschneidungen durch gleichbenannte Elemente- oder Attribute in zu unterscheidenden Vokabularen auszuschließen. XML-Namensräume bilden damit die notwendige Voraussetzung zur freien dezentralen Entwicklung eigener Vokabulare ohne die Möglichkeit einer späteren Syndikatisierung zu verlieren.

Konzept der Namensräume:
Die Recommendation *Namespaces in XML* definiert die Syntax und Semantik der Namensräume. Ihr Konzept wurde rund ein Jahr nach Verabschiedung der ersten XML-Version eingeführt. Daher wurde der Kompatibilität mit bereits existierenden XML-Dokumenten große Priorität eingeräumt.

Grundidee der Namensräume ist es, die Element- und Attributnamen dergestalt zu erweitern, daß (auch nach Vereinigung beliebiger Dokumente wieder) eineindeutige Bezeichner entstehen. Dies könnte durch anwenderdefinierte Erweiterungen geschehen, sie trügen jedoch wiederum die Gefahr in sich, daß sie unbeabsichtigt mehrfach benutzt würden.
Daher scheidet der unkoordinierte Einsatz solcher Namenserweiterungen aus. Jegliche Koordination bedingt jedoch inhärent eine zentrale Vergabestelle zur Registrierung der vergebenen Namen, die über die Eindeutigkeit wacht und Mehrfachnutzungen unterbindet.
Die Einführung einer solchen Stelle hätte jedoch einen unüberschaubaren Verwaltungsaufwand bedeutet, den das W3C nicht zu leisten im Stande wäre. Man nehme nur als Vergleich das Vergabeverfahren von Einträgen des Internet Domain Name Systems (DNS), welches bereits dezentral durch die einzelnen nationalen Domain-Registrars gehandhabt wird. Der dort anzutreffende Aufwand hätte sich für XML-Namensräume potenziert, legt man pro Domainadresse mehrere Namensräume zugrunde.

Ziel des W3C war es, durch die Namensräume einen gleichermaßen mächtigen als auch leicht zu handhabenden und zu administrierenden Identifikationsmechanismus zu etablieren. Offenkundig wird diesem Anspruch nur ein (überwiegend) dezentraler, aber dennoch die Eineindeutigkeit garantierender, Ansatz gerecht.
Diesen Anforderungen genügt das aus IETF RFC 2396 bekannte Namensschema der *Uniform Resource Identification* (URI) (später aktualisiert in IETF RFC 2732). Es kombiniert zentrale und dezentrale Elemente in der Handhabung, und ermöglicht so -- trotz Existenz und Pflege einer zentralen Registratur -- größtmögliche Flexibilität in der Anwendung. Der bekannteste Einsatz von URI-Namen ist der im World-Wide-Web allgegenwärtige *Uniform Ressource*

Locator (URL) (IETF RFC 1738); einer Untermenge der URI.
Die zentrale Komponente findet sich im Domainnamen verwirklicht. Er ist entweder durch die IP-Adresse (konkret: IPv4-Adresse; im Falle des RFC 2732: der IPv6-Adresse) oder deren literaler Repräsentation gegeben. Unterhalb der Domainebene kann durch deren Verwalter eine beliebige Strukturierung vorgenommen werden. Die verschiedenen Ebenen werden dabei durch ISO-10646/ASCII #x2F „*/*“ voneinander abgetrennt.
Wie auch bereits bei URLs notwendig, ist das Schema (*URI scheme*) (z.B. `http`) zwingend mitanzugeben.

Trotz der Möglichkeit XML-Namensräume durch URLs zu identifizieren handelt es sich dabei nicht die Bezeichnung einer Internetquelle. Die verwendete Zeichenkette dient ausschließlich Benennung der im Namensraum versammelten XML Element Information Items und Attribute Information Items.
Die Auflösung des Namensraumbezeichners durch einen XML-Prozessor ist nicht vorgesehen.

Nachfolgend ist die in definierte Syntax einer URI wiedergegeben. Sie wurde behutsam an die in der XML-Spezifikation verwendete BNF-Notation (In XML-Spezifikation nachschlagen) angepaßt, ohne jedoch die Produktionen in ihrer Struktur zu verändern.

```
[URI1]  URI-reference ::= (absoluteURI | relativeURI)? ("#" fragment)?
[URI2]  absoluteURI   ::= scheme ":" ( hier_part | opaque_part )
[URI3]  relativeURI   ::= ( net_path | abs_path | rel_path ) [ "?" query ]
[URI4]  hier_part     ::=  ( net_path | abs_path ) ("?" query)?
[URI5]  opaque_part   ::= uric_no_slash uric?
[URI6]  uric_no_slash ::= unreserved | escaped | ";" | "?" | ":" | "@" |
                          "&" | "=" | "+" | "$" | ","
[URI7]  net_path      ::= "//" authority abs_path?
[URI8]  abs_path      ::= "/" path_segments
[URI9]  rel_path      ::= rel_segment abs_path?
[URI10] rel_segment   ::= (unreserved | escaped |
                          ";" | "@" | "&" | "=" | "+" | "$" | "," )+
[URI11] scheme        ::= alpha (alpha | digit | "+" | "-" | "." )*
[URI12] authority     ::= server | reg_name
[URI13] reg_name      ::= ( unreserved | escaped | "$" | "," |
                          ";" | ":" | "@" | "&" | "=" | "+" )+
[URI14] server        ::= ((userinfo "@")? hostport)?
[URI15] userinfo      ::= ( unreserved | escaped |
                          ";" | ":" | "&" | "=" | "+" | "$" | "," )*
[URI16] hostport      ::= host (":" port)?
[URI17] host          ::= hostname | IPv4address
[URI18] hostname      ::= ( domainlabel "." )* toplabel (".")?
[URI19] domainlabel   ::= alphanum | alphanum *( alphanum | "-" ) alphanum
[URI20] toplabel      ::= alpha | alpha (alphanum | "-" )* alphanum
[URI21] IPv4address   ::= digit+ "." digit+ "." digit+ "." digit+
[URI22] port          ::= digit*
[URI23] path          ::= (abs_path | opaque_part)?
[URI24] path_segments ::= segment ("/" segment)*
[URI25] segment       ::= pchar* (";" param)*
[URI26] param         ::= pchar*
[URI27] pchar         ::= unreserved | escaped |
                          ":" | "@" | "&" | "=" | "+" | "$" | ","
[URI28] query         ::= uric*
[URI29] fragment      ::= uric*
[URI30] uric          ::= reserved | unreserved | escaped
[URI31] reserved      ::= ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
                          "$" | ","
[URI32] unreserved    ::= alphanum | mark
[URI33] escaped       ::= "%" hex hex
[URI34] hex           ::= digit | "A" | "B" | "C" | "D" | "E" | "F" |
                          "a" | "b" | "c" | "d" | "e" | "f"
[URI35] digit         ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                          "8" | "9"
[URI36] uric_no_slash ::= unreserved | escaped | ";" | "?" | ":" | "@" |
                          "&" | "=" | "+" | "$" | ","
```

Die Produktionen `alphanum`, `lowalpha` sowie `upalpha` zur Konstruktion der alphanumerischen Namen wurden aus Übersichtlichkeitsgründen weggelassen.

Neben einigen anderen gängigen URI-Varianten stellt das nachfolgende Beispiel einige der möglichen syntaktisch korrekten URIs zusammen, die für die späteren Betrachtungen von Interesse sind.

Beispiel 13: Gültige URIs

```
(1)http://www.wi.fh-furtwangen.de
(2)http://meinrechner.wi.fh-augsburg.de
(3)mailto:mario@jeckle.de
(4)ftp://ftp.shareware.com
(5)http://www.jeckle.de/xml/vorlesung/script.htm#Namespaces
(6)#EinfuehrungUndUeberblick
(7)urn:oasis:names:specification:docbook:dtd:xml:4.1.2
(8)urn:oid:1.3.6.1.2.1.27
(9)org.omg/standards/UML
```

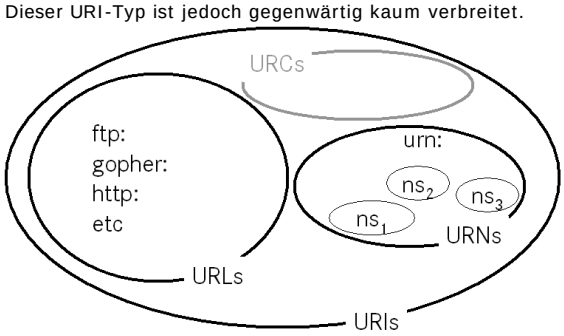
Exkurs: URIs, URLs, URNs ...

Vielfach wird in der Praxis die Abgrenzung der im Internet gebräuchlichen Adressierungs- und Identifikationsmechanismen nicht trennscharf vollzogen.
Darüberhinaus trat im Laufe der Entwicklung eine merkliche Bedeutungsverschiebung insbesondere zwischen der *Uniform Resource Identifikation* und den als WWW-Adressen genutzten *Uniform Resource Locators* ein.

Gegenwärtig wird die Begriffsabgrenzung wie in Abbildung 12 schematisch dargestellt vollzogen:

- *Uniform Resource Identification* (URI) dient als abstrakter Oberbegriff eineindeutig identifizierbarer Web-Ressourcen. Konzeptionell sind URIs:
  - über Zeit und Raum eindeutig
  - für Menschen leicht zu merkend
  - mit keinerlei Registrierungskosten verbunden
  - unabhängig von der tatsächlichen Lokalisation der so identifizierten Ressource

  Der URI-Raum zerfällt in die disjunkten Bezeichnerschemata URL, URN und URC.
- *Uniform Resource Location* (URL) bezeichnet den physischen Aufenthaltsort einer Ressource, etwa den Ablageort einer HTML-Seite.
  Beispiele:
  - `http://www.jeckle.de/vorlesung/xml/script.html`
  - `http://www.wi.fh-furtwangen.de/`
  - `mailto:mario@jeckle.de`
  - `ftp://example.org/aDirectory/aFile`
  - `news:comp.infosystems.www`
  - `tel:+1-816-555-1212`
  - `ldap://ldap.example.org/c=GB?objectClass?one`
  - `urn:oasis:SAML:1.0`
- *Uniform Resource Name* (URN) bezeichnen den eineindeutigen Namen einer beliebigen Resource. Für die URN existiert kein Auflösungsmechanismus durch den die physische Lokation ermittelt werden könnte. URNs dienen daher ausschließlich der eindeutigen Benennung!
  Syntaktisch folgt auf das definierte Kürzel urn eine Zeichenkette welche eine Weiterklassifikation der Ressource gestattet. Hierdurch wird eine weitere Partitionierung des URN-Raumes erzielt. Diese *namespace ID* genannten Zeichenketten unterliegen einem globalen Registrierungszwang um ihre Eindeutigkeit zu gewährleisten. Diese Unterstrukturierungen sind in der Abbildung als ns_1 bis ns_3 benannt.
  Beispiele:
  - `urn:oasis:names:specification:docbook:dtd:xml:4.1.2`
  - `urn:oid:1.3.6.1.2.1.27`
- *Uniform Resource Citation* (URC) schlägt die Brücke zwischen Lokationsbezeichnung und reiner Benennungskonvention. Eine URC verweist in eine Metadatenstruktur welche die physischen Ressourcen-Aufenthaltsorte katalogisiert.
  Dieser URI-Typ ist jedoch gegenwärtig kaum verbreitet.



Web-Referenzen 6: Weiterführende Links
- URIs, URLs, and URNs: Clarifications and Recommendations
- The Anatomy of an URL

Verwendung von Namensräumen:
Am naheliegendsten wäre nach der Zielsetzung der Verwendung von URIs zur eindeutigen Benennung von XML-Element- und Attributnamen, die URI direkt vor dem XML-Bezeichner zu plazieren, evtl. separiert durch ein Trennsymbol wie den Doppelpunkt „:“.
Hieraus entstünden dann, auf jeden Fall eindeutige, Element- und Attributnamen wie beispielsweise für das erste

Beispieldokument dieses Kapitels (die URI `http://www.example.com/sales` werde zur Identifizierung verwendet):

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)     <http://www.example.com/sales:Rechnung>
(3)             <http://www.example.com/sales:Kunde>
(4)                     <http://www.example.com/sales:KundenNr>4711</http://www.example.com/sales:
KundenNr>
(5)                     <http://www.example.com/sales:Name>Max Mustermann</http://www.example.com/
sales:Name>
(6)                     <http://www.example.com/sales:Anschrift>
(7)                             <http://www.example.com/sales:Straße>Musterplatz 1</http://www.
example.com/sales:Straße>
(8)                             <http://www.example.com/sales:PLZ>12345</http://www.example.com/sales:
PLZ>
(9)                             <http://www.example.com/sales:Ort>Musterstadt</http://www.example.com/
sales:Ort>
(10)                    </http://www.example.com/sales:Anschrift>
(11)            </http://www.example.com/sales:Kunde>
(12)    <http://www.example.com/sales:Rechnungsposten>
(13)            ...
(14)    </http://www.example.com/sales:Rechnungsposten>
(15)</http://www.example.com/sales:Rechnung>
```

Bei entsprechender Nachbearbeitung des zweiten Beispieldokumentes mit einem anderen URI-identifizierten Namensraum, entstehen eindeutige Element- und Attributnamen, die nicht mehr kollidieren.

Jedoch verstößt diese Lösung gegen die in Produktion 5 der XML-Spezifikation formulierte syntaktische Einschränkung. Sie erlaubt das in URIs elementare Pfadtrennersymbol („/“) (aus den URI-Produktionen 8, 24 und 31) nicht in XML-Namen (#x2F findet sich nicht in den in Produktion 85 aufgeführten Unicode-Blöcken).
Die Integration der Namensräume auf diesem Weg hätte daher eine Modifikation der XML-Spezifikation nach sich gezogen. Diese erweiternde Aufweichung der zugelassenen Namen für Elemente und Attribute hätte jedoch mit der Kompatibilität zu SGML gebrochen, und somit eine der Grundforderungen der XML-Entwicklung verletzt.
Darüberhinaus ist die Spezifikation vollständiger URIs für Menschen „unhandlich“ und reduziert die Lesbarkeit der entstehenden XML-Dokumente.

Als Ausweg und pragmatischer Kompromiß zwischen eineindeutigen Namenspräfixen und Lesbarkeit wurde daher ein zweistufiges Verfahren eingeführt. Es erlaubt die Zuordnung von URIs zu Präfixen. Dieser Vorgang wird als „Bindung“ bezeichnet.
Diese Präfixes können Attributen oder Elementen vorangestellt werden, um sie in bestimmte Namensräume zu übernehmen.
Für die Präfixe gelten dieselben Bildungsgesetze wie für die Element- und Attributnamen. Im Einzelnen legt die *Namespace Recommendation* fest: (im XML-Namespace-Dokument nachschlagen)

```
[NS7] Präfix     ::= NCName
[NS4] NCName     ::= (Letter | '_') (NCNameChar)*
[NS5] NCNameChar ::= Letter | Digit | '.' | '-' | '_'
                   | CombiningChar
                   | Extender
```

Anmerkung: Die rechten Seiten der Produktionen beziehen sich entweder auf die dargestellten Definitionen des Namespace-Standards oder auf Syntaxregeln der XML-Recommendation.

Die Bindung einer URI an ein -- gemäß Produktion NS7 frei wählbares -- Präfix geschieht durch das reservierte Attribut `xmlns`.
Die Syntax hierfür wird mit

```
[NS2] PräfixedAttName ::= 'xmlns:' NCName
```

angegeben.

Nach der Bindung der URI an das Präfix kann dieses jedem Element oder Attribut vorangestellt werden, um es in den Namensraum zu übernehmen.
Hierdurch verändert sich die Produktion Name aus der XML-Spezifikation zum *qualifizierten Namen*, der durch die Voranstellung des Präfixes entsteht. Der rechts vom trennenden Doppelpunkt folgende Elementname stellt den lokalen Namen (innerhalb des Namensraumes dar). Dieser lokale Name darf *keinen* Doppelpunkt mehr enthalten; insofern schränkt Produktion NS8 in Verbindung mit NS4 die Festlegung der Produktion 5 der XML-Spezifikation ein.

```
[NS6] QName     ::= (Präfix ':')? LocalPart
[NS8] LocalPart ::= NCName
```

Während der Verarbeitung eines XML-Dokuments, das Namensräume nutzt, ersetzt ein XML-Prozessor jedes Auftreten eines deklarierten Präfixes transparent durch die gebundene URI.
Prozessoren, welche die Namensraum-Spezifikation unterstützen, werden als *namespace aware* bezeichnet. Alle anderen Prozessoren treffen die durch NS6 eingeführte Unterscheidung zwischen *Präfix* und *LocalPart* eines qualifizierten Namens nicht und betrachten die Kombination aus Präfix und Element- bzw. Attributnamen als Bezeichner. Die Präfix-URI-Bindung durch das `xmlns:...`-Attribut wird hierbei als gewöhnliches XML-Attribut betrachtet und führt daher zu keinen Validierungsfehlern. (Die Einschränkung der Produktion 5, ein Name dürfe nicht

mit der Zeichenfolge (('X'|'x') ('M'|'m') ('L'|'l')) beginnen, stellt in der XML-Spezifikation lediglich einen Hinweis dar.)

Semantisch bildet die durch xmlns eingeleitete Deklaration ein *Pseudoattribut*, da es für die maschinelle Verarbeitung vorbehalten und mit festelegter Bedeutung ausgestattet ist, welche durch den XML-Dokumentautor nicht verändert werden kann.
Zusätzlich werden Namensraumdeklarationen durch Programmiersprachenschnittstellen nicht den gewöhnlichen Attributen gleichgestellt betrachtet, sondern nehmen, wie auch im Information Set, dort eine Sonderstellung ein.

Anmerkung: Auf Webseiten und in Mailinglisten finden sich manchmal Formulierungen der Struktur {*namespaceName*} *elementName* (z.B. {http://www.w3.org/2001/XMLSchema}element oder {http://www.w3.org/1999/XSL/Transform}template).
Hierbei handelt es sich um eine zwar geläufige, aber *nicht spezifikationskonforme Schreibweise*!
Sie dient lediglich dazu, das prinzipiell beliebig wählbare Präfix einzusparen und den gewählten Namensraum hervorzuheben.
Strukturen dieses Stils sind jedoch keine gültigen XML-Dokumente!

Angewendet auf das betrachtete Beispiel läßt sich die URI http://www.example.com/sales an das Präfix myNS1 binden. Diese Bindung steht im definierenden Element (local name: rechnung) und allen untergeordneten zur Verfügung.

Beispiel 14: Dokument mit W3C-konformen Namensräumen

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<myNS1:Rechnung xmlns:myNS1="http://www.xyz.com/sales">
(3)   <myNS1:Kunde>
(4)      <myNS1:KundenNr>4711</myNS1:KundenNr>
(5)      <myNS1:Name>Max Mustermann</myNS1:Name>
(6)      <myNS1:Anschrift>
(7)         <myNS1:Straße>Musterplatz 1</myNS1:Straße>
(8)         <myNS1:PLZ>12345</myNS1:PLZ>
(9)         <myNS1:Ort>Musterstadt</myNS1:Ort>
(10)      </myNS1:Anschrift>
(11)   </myNS1:Kunde>
(12)   <myNS1:Rechnungsposten>
(13)    <!--.....-->
(14)    </myNS1:Rechnungsposten>
(15)</myNS1:Rechnung>
```

Download des Beispiels

Hinweis: Für das Attribut xmlns kann keine Namensraumdeklaration angegeben werden; es ist spezifikationsgemäß an keinen Namensraum gebunden.

Die Deklaration des Namensraumes mit der Präfixbindung kann auf beliebige hierarchisch höhergeordnete Elemente ausgelagert werden. In der Praxis hat es sich aus Übersichtlichkeitsgründen durchgesetzt, alle in einem XML-Dokument benutzten Namensräume mit ihren Präfixen zu Beginn des Dokuments im Wurzelelement zu definieren. Das nachfolgende Beispiel zeigt dies anhand eines XHTML-Dokuments, das neben Elementen der Hypertextsprache auch mathematische Formeln und Vektorgraphiken enthält.

Beispiel 15: Ein XHTML-Dokument mit MathML- und SVG-Inhalten

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml"
(3)            xmlns:mml="http://www.w3.org/TR/REC-MathML"
(4)            xmlns:svg="http://www.w3.org/2000/svg">
(5)     <xhtml:head>
(6)              <xhtml:title>XHTML Dokument, mit MathML- und SVG-Inhalten</xhtml:title>
(7)     </xhtml:head>
(8)     <xhtml:body>
(9)              <xhtml:h1>Eine Û¢erschrift</xhtml:h1>
(10)             <mml:math>
(11)             <mml:mrow>
(12)                     <mml:mi>x</mml:mi>
(13)                     <mml:mo>=</mml:mo>
(14)                     <mml:mfrac>
(15)                             <mml:mrow>
(16)                                     <mml:mrow>
(17)                                             <mml:mo>-</mml:mo>
(18)                                             <mml:mi>b</mml:mi>
(19)                                     </mml:mrow>
(20)                                     <mml:mo>&PlusMinus;</mml:mo>
(21)                                     <mml:msqrt>
(22)                                             <mml:mrow>
(23)                                                     <mml:msup>
(24)                                                             <mml:mi>b</mml:mi>
(25)                                                             <mml:mn>2</mml:mn>
(26)                                                     </mml:msup>
(27)                                                     <mml:mo>-</mml:mo>
(28)                                                     <mml:mrow>
(29)                                                             <mml:mn>4</mml:mn>
(30)                                                             <mml:mo>&InvisibleTimes;</mml:mo>
```

```
(31)                                                    <mml:mi>a</mml:mi>
(32)                                                    <mml:mo>&InvisibleTimes;</mml:mo>
(33)                                                    <mml:mi>c</mml:mi>
(34)                                                </mml:mrow>
(35)                                            </mml:mrow>
(36)                                    </mml:msqrt>
(37)                            </mml:mrow>
(38)                            <mml:mrow>
(39)                                    <mml:mn>2</mml:mn>
(40)                                    <mml:mo>&InvisibleTimes;</mml:mo>
(41)                                    <mml:mi>a</mml:mi>
(42)                            </mml:mrow>
(43)                    </mml:mfrac>
(44)            </mml:mrow>
(45)            </mml:math>
(46)            <svg:svg width="4cm" height="8cm">
(47)                    <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm"/>
(48)            </svg:svg>
(49)    </xhtml:body>
(50)</xhtml:html>
```

Download des Beispiels

Definition 7: Namensraumidentifikation
Jeder XML-Namensraum wird durch eine gültige URI identifziert. Diese URI dient ausschließlich der Benennung, daher muß sie nicht auf eine gültige Ressource verweisen.

Überschreiben des Vorgabe-Namensraums:
Aus den Beispielen ist leicht ersichtlich, daß die explizite Angabe des definierten Präfixes für jedes Element eines Namensraumes platzraubend und für die Zuordnung aller Elemente eines Teilbaumes zum selben Namensraum redundant und -- wegen des zusätzlichen Spezifikationsaufwandes -- unpraktikabel ist. Die mehrmalige explizite redundante (identische) Angabe des identifizierenden Präfixes bildet zusätzlich noch eine potentielle Fehlerquelle hinsichtlich Übertragungsfehlern und reiner Tippfehler bei manuell erstellten XML-Dokumenten.

Eine einfache Kompaktifizierungsvariante greift auf die aus den Programmiersprachen geläufigen Regeln für Namensräume zurück. Dort beinhaltet ein explizit geöffneter Block alle enthaltenen Elemente bis zum Blockendesymbol und faßt sie so zu einem Gültigkeitsbereich zusammen.
Dieses Prinzip läßt sich leicht auch auf XML-Dokumente, die immer eine streng hierarchische Baumstruktur aufweisen, anwenden.

Hierzu wird das `xmlns`-Attribut leicht modifiziert eingesetzt. Wird es *ohne nachfolgendes Präfix* und unter Weglassung des separierenden Doppelpunktes verwendet, so definiert es einen *Vorgabenamensraum* (*default namespace*). Dieser umfaßt neben dem Element, welches das Attribut beinhaltet, auch alle Kindelemente. Eine Ausnahme hiervon bilden untergeordnete Elemente, die explizit durch Präfix oder Redefinition des Vorgabenamensraumes einem anderen Namespace zugeordnet werden.

Das nachfolgende Beispiel zeigt dies für das bereits mit Namenräumen versehene Rechnungsdokument

Die syntaktische Definitionsform der Namensraumüberschreibung als XML-(Pseudo-)Attribut stellt hierbei sicher, daß für ein Element keine mehrmalige Überschreibung des Vorgabenamensraumes vorgenommen werden kann, da in diesem Falle das Attribut `xmlns` mehrfach im selben Elementkontext auftreten müßte, was der XML-Basisspezifikation widerspräche.

Beispiel 16: Rechnungsdokument mit überschriebenem Vorgabenamensraum

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<Rechnung xmlns="http://www.xyz.com/sales">
(3)   <Kunde>
(4)      <KundenNr>4711</KundenNr>
(5)      <name>Max Mustermann</Name>
(6)      <Anschrift>
(7)         <Straße>Musterplatz 1</Straße>
(8)         <PLZ>12345</PLZ>
(9)         <Ort>Musterstadt</Ort>
(10)      </Anschrift>
(11)   </Kunde>
(12)   <Rechnungsposten>
(13)            <!--...-->
(14)    </Rechnungsposten>
(15)</Rechnung>
```

Download des Beispiels

Durch die Definition des Vorgabenamensraumes für das Element `rechnung` und all dessen Kindelemente wird derselbe Effekt erreicht wie durch die Präfixangabe im vorangegangenen Beispiel.
Diese Schreibweise stellt lediglich eine Abkürzung der expliziten Qualifizierung jedes einzelnen XML-Namens dar.
Insbesondere führt die mehrmalige Redefinition des Vorgabenamensraumes *nicht* zu kaskadierten Namensräumen.
Jeder Namensraum ist von allen umgebenden unabhängig definiert.

So kann das Dokument des XHTML-Beispiels auch dahingehend verändert werden, daß die Namensräume erst an der Stelle im Dokument deklariert werden, an der sie auch benötigt werden.

Beispiel 17: Ein XHTML-Dokument mit MathML- und SVG-Inhalten, unter Verwendung überschriebener Vorgabenamensräume

```
(1)<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
(2)<html xmlns="http://www.w3.org/1999/xhtml">
(3)     <head>
(4)             <title>XHTML Dokument, mit MathML- und SVG-Inhalten</title>
(5)     </head>
(6)     <body>
(7)             <h1>Eine Überschrift</h1>
(8)             <math xmlns="http://www.w3.org/1998/Math/MathML">
(9)             <mrow>
(10)                    <mi>x</mi>
(11)                    <mo>=</mo>
(12)                    <mfrac>
(13)                            <mrow>
(14)                                    <mrow>
(15)                                            <mo>-</mo>
(16)                                            <mi>b</mi>
(17)                                    </mrow>
(18)                                    <mo>+-</mo>
(19)                                    <msqrt>
(20)                                            <mrow>
(21)                                                    <msup>
(22)                                                            <mi>b</mi>
(23)                                                            <mn>2</mn>
(24)                                                    </msup>
(25)                                                    <mo>-</mo>
(26)                                                    <mrow>
(27)                                                            <mn>4</mn>
(28)                                                            <mo> </mo>
(29)                                                            <mi>a</mi>
(30)                                                            <mo> </mo>
(31)                                                            <mi>c</mi>
(32)                                                    </mrow>
(33)                                            </mrow>
(34)                                    </msqrt>
(35)                            </mrow>
(36)                            <mrow>
(37)                                    <mn>2</mn>
(38)                                    <mo> </mo>
(39)                                    <mi>a</mi>
(40)                            </mrow>
(41)                    </mfrac>
(42)            </mrow>
(43)            </math>
(44)            <svg xmlns="http://www.w3.org/2000/svg" xmlns:svg="http://www.w3.org/2000/svg" svg:
width="4cm" svg:height="8cm">
(45)                    <ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm"/>
(46)            </svg>
(47)    </body>
(48)</html>
```

Download des Beispiels

Die Namensraumpräfixe können durch den Anwender frei vergeben werden. Sie dienen lediglich der abkürzenden Schreibweise und sind für die Namensraumauflösung unerheblich.
Daher werden zwei Elemente oder Attribute als gleich betrachtet, wenn sie lexikalisch in Namen und Namensraumidentifier übereinstimmen. Hierbei ist es unerheblich, ob der Namensraum explizit durch Präfixangabe oder durch Überschreiben des Vorgabenamensraumes definiert wurde.
Die Elemente der XML-Dokumente aus den Beispielen 18 und 19 befinden sich alle ausnahmslos im Namensraum `http://www.example.com`.

Beispiel 18: Namensraumpräfixe 1

```
(1)<abc:ElementA xmlns:abc="http://www.example.com"
(2)                             xmlns:xyz="http://www.example.com">
(3)     <ElementB xmlns="http://www.example.com">
(4)             <ElementC/>
(5)     </ElementB>
(6)     <xyz:ElementB>
(7)             <abc:ElementC/>
(8)     </xyz:ElementB>
(9)</abc:ElementA>
```

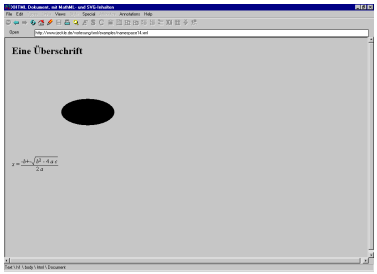
Download des Beispiels

Beispiel 19: Namensraumpräfixe 2

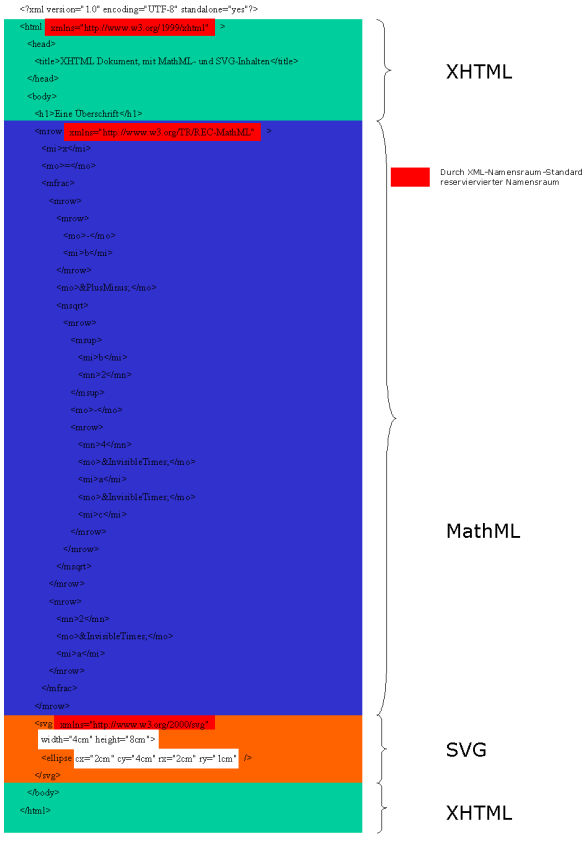
```
(1)<ElementA xmlns="http://www.example.com"
(2)             xmlns:myNamespace="http://www.example.com">
(3)   <foo:ElementB xmlns:foo="http://www.example.com">
(4)      <myNamespace:ElementC/>
(5)   </foo:ElementB>
(6)   <ElementB xmlns="http://www.example.com">
(7)      <myNamespace:ElementC/>
(8)   </ElementB>
(9)</ElementA>
```

Download des Beispiels

Die Abbildung zeigt das Beispieldokument in der Darstellung des W3C-Browsers Amaya.



Im Beispieldokument wird der Vorgabenamensraum dreimal, entsprechend der verschiedenen verwendeten XML-Sprachen, neu gesetzt. So wird auf `html` und alle direkt untergeordneten Elemente der URI-identifizierte Namensraum `http://www.w3.org/1999/xhtml` angewendet. head, `title` und body sowie dessen Kindelemente finden sich demnach, da sie keinen eigenen Namensraum definieren, ebenfalls im so definierten Vorgabenamensraum.
`mrow` als hierarchisch tieferstehendes Element redefiniert den Namensraum zu `http://www.w3.org/TR/REC-MathML`. Daher werden das Element `mrow` sowie all dessen Kindelemente (im Beispiel: `ellipse`) auch diesem zugeordnet.
Die Attribute `width`, `height`, `cx` , ... verfügen über kein explizites Namensraumpräfix und sind daher dem leeren Namensraum zugeordnet.
Auf den MathML-Namensraum folgend wird der Vorgabenamensraum zu `http://www.w3.org/2000/svg` redefiniert. Auch hier gelten dieselben Regeln, d.h. der überschriebene Vorgabenamensraum erstreckt sich auf alle Kindelemente.
Mit dem schließenden Tag svg endet auch dessen Namensraum. Alle folgenden Elemente befinden sich wieder im umgebenden Namensraum, der zu Beginn des Dokuments mit `http://www.w3.org/1999/xhtml` festgelegt wurde.
Die nachfolgende Graphik stellt die Namensräume nochmals farblich hervorgehoben dar.
Ein weiteres Beispiel findet sich in der Namespace-Recommendation.



Der XML-Namensraumstandard des W3C sieht die beiden im Vorhergehenden diskutierten Varianten exklusiv zueinander vor. D.h. für ein Element, welchem bereits durch Präfixangabe eine Namensraumzuordnung gegeben wurde, kann nicht zusätzlich der Vorgabenamensraum überschrieben werden. Deklarationen der Form `<xyz:abc xmlns="..." ...>` sind widersprüchlich; und daher illegal. (in der XML-Namespace Recommendation nachschlagen)

Das abschließende Beispiel 20 zeigt die Verwendung zweier Vokabulare (SVG und MathML), die beide ein mit set benanntes Element definieren.
Durch die Deklaration der jeweiligen Namensräume unterscheiden sich die qualifizierten Namen, die dem (gleichnamigen) Elementnamen die Namensraum-URI voranstellen.

Beispiel 20: Namensräume im realen Einsatz

```
(1)<?xml version="1.0"?>
(2)<document>
(3)     <svg xmlns="http://www.w3.org/2000/svg">
(4)             <g transform="translate(100,100)">
(5)                     <text id="TextElement" x="0" y="0" style="font-family:Verdana; font-
size:35.27; visibility:hidden">
(6)                     It's alive!
(7)                             <set attributeName="visibility" attributeType="CSS" to="visible"
begin="3s" dur="6s" fill="freeze"/>
(8)                     </text>
(9)             </g>
(10)    </svg>
(11)
(12)    <math xmlns="http://www.w3.org/1998/Math/MathML">
(13)            <set>
(14)                    <ci> b </ci>
(15)                    <ci> a </ci>
(16)                    <ci> c </ci>
(17)            </set>
(18)    </math>
(19)</document>
```

Download des Beispiels

Präzedenz des explizit zugeordneten Namensraumes:
Eine explizit durch Präfixzuordnung vorgenommene Namensraumfestlegung besitzt Präzedenz gegenüber dem evtl. überschriebenen Vorgabenamensraum.
Findet daher für ein Element sowohl die Überschreibung des Vorgabenamensraumes, als auch gleichzeitig die Namensraumfestlegung durch explizite Präfixzuordnung statt, so wird das Element demjenigen Namensraum zugeordnet, der durch die URI identifiziert wird, an den das Präfix gebunden ist.
Dies gilt insbesondere auch dann, wenn ein und dasselbe Element sowohl über ein Präfix, als auch eine Überschreibung des Vorgabenamensraumes verfügen.
Das XML-Dokument aus 21 illustriert dies beispielhaft. So wird `ElementA` -- durch Überschreibung des Vorgabenamensraumes -- dem Namensraum `urn:namspaces:Namespace1` zugeordnet und diese Festlegung auch an das Kindelement `ElementB` weitergegeben.
Das Kindelement `ElementC` hingegen überschreibt die Vorgabe des Elternelements durch explizite Präfixangabe und ist daher dem durch `urn:namespace:Namespace2` identifizierten Namensraum zugeordnet.
Für `ElementD` findet sich sowohl eine Namensraumdefinition, welche durch Überschreiben des Vorgabenamensraumes zu `urn:namespace:Namespace3` stattfindet, als auch eine Präfix-gebundene Definition an den Namensraum `urn:namespace:Namespace2`. Gemäß der Präzedenz der expliziten Festlegung durch Präfix wird `ElementD` jedoch ausschließlich dem Namensraum zugeordnet, an den das angegebene Präfix `ns1` gebunden ist. Im Beispiel ist dies die URI `urn:namespace:Namespace2`.

Beispiel 21: Präzedenzregel

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<ElementA xmlns="urn:namspaces:Namespace1"
(3)                     xmlns:ns1="urn:namespace:Namespace2"
(4)                     xmlns:ns2="urn:namespace:Namespace3">
(5)     <ElementB/>
(6)     <ns1:ElementC/>
(7)     <ns1:ElementD xmlns="urn:namespace:Namespace3"/>
(8)</ElementA>
```

Download des Beispiels

Aufheben der Namensraumzuweisung:
Durch Überschreibung des Vorgabenamensraumes mit der Zeichenkette leeren Inhalts -- formal der Zuweisung der leeren URI als Namensraumidentifikator -- kann eine bestehende Namensraumdefinition aufgehoben werden. Als Resultat entsteht eine Situation identisch zu einem Dokument ohne festgelegte Namensräume, d.h. die Elemente finden sich im leeren Namensraum.

Beispiel 22: Aufheben von Namensraumdeklarationen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<Adressen>
(3)     <table xmlns="http://www.w3.org/TR/REC-html40">
(4)             <tr>
(5)                     <td>Name</td>
(6)                     <td>Adresse</td>
(7)             </tr>
(8)             <tr>
(9)                     <td>
(10)                            <Vorname xmlns="">Max</Vorname>
(11)                    <Nachname xmlns="">Mustermann</Vorname>
```

```
(12)                    </td>
(13)                    <td>
(14)                            <Straße xmlns="">Musterstr. 1</Straße>
(15)                            <PLZ xmlns="">12345</PLZ>
(16)                            <Ort xmlns="">Musterstadt</Ort>
(17)                    </td>
(18)            </tr>
(19)    </table>
(20)</Adressen>
```

Das Beispiel 22 zeigt die notwendigen Deklarationen zur Aufhebung der Vorgabenamensraumdefinition.
So wird zwar für das Element `table` und alle seine Kindelemente der Vorgabenamensraum auf `http://www.w3.org/TR/REC-html40` gesetzt, dies jedoch für die Kindelemente `Vorname`, `Nachname`, `Straße`, `PLZ` und `Ort` durch die Festlegung `xmlns=""` explizit für das jeweilige Element aufgehoben.

Die Aufhebung von definierten Namensräumen kann ausschließlich durch die Überschreibung des Vorgabenamensraum erfolgen. Eine Bindung der leeren URI an ein Präfix zur späteren Verwendung ist nicht zugelassen.

Namensräume für Attribute:
Abweichend von der Mimik für Elemente, dort wirkt sich ein überschriebener Vorgabenamensraum auch immer auf die Kindelemente aus, wird eine Namensraumdeklaration auf Elementebene nicht auf Attribute propagiert.
Diese Festlegung der Spezifikation mag insbesondere unter Kenntnis der Baumstruktur der Infosets, welche Attribute und Elemente gleichermaßen als Kindknoten der beherbergenden Elementinformationseinheit darstellt, verwundern.
Eine mögliche Begründung dieser Asymmetrie mag in der besonderen Rolle der Attribute zur Informationsdarstellung liegen. So wird teilweise damit argumentiert, daß Attribute üblicherweise unabhängig vom aktuell umgebenden Element sein sollten und daher nur zur Darstellung von Daten herangezogen werden sollten, die nicht über einen direkten Bezug zum sie umgebenden Element verfügen.
In der Konsequenz müssen Attribute immer explizit mit einem Namensraumpräfix versehen werden, um sie einem Namensraum zuzuordnen.
Beispiel 23 zeigt die Anwendung der Namensräume auf Attribute. So befinden sich weder das Attribute `att1` des Elements `ElementB`, noch dasjenige von `ElementD` in einem Namensraum. Das mit dem Wert `XYZ` versehene Attribut `att2` des Elements `ElementC` wird hingegen -- aufgrund des explizit angegebenen Präfixes -- dem Namensraum `http://www.example.com/NS2` zugeordnet.
Ferner illustriert `ElementC` die Rolle der Namensräume als Bestandteil des identifzierenden Namens von Elementen und Attributen. Aufgrund der Interpretation des Namensraumes als Benennungsbestandteil darf das `att2` benannte Attribut mehrfach auftreten, da die Zuhilfenahme des Namensraumes die eindeutige Identifikation gestattet.

Beispiel 23: Namensräume für Attribute

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<Wurzelelement>
(3)     <ElementA xmlns:NS1="http://www.example.com/NS1" xmlns:NS2="http://www.example.com/NS2">
(4)             <ns2:ElementB att1="...">
(5)                     <ElementD att1="..." xmlns="http://www.example.com/NS3">
(6)                             <ElementC att2="ABC" NS2:att2="XYZ"/>
(7)                     </ElementD>
(8)             </ns2:ElementB>
(9)     </ElementA>
(10)</Wurzelelement>
```

Definition 8: Namensraumvererbung

Namensräume, die durch Überschreiben des Vorgabenamensraumes zugewiesen werden wirken sich ausschließlich auf Elemente und deren direkte oder transitive Kindelemente aus, sofern diese den Namensraum nicht wieder verändern.
Namensräume, die durch explizite Präfixangabe zugewiesen werden, wirken sich ausschließlich auf dasjenige Element aus vor dessen Name das Präfix plaziert ist.
Namensräume für Attribute werden ausnahmslos durch explizite Präfixangabe festgelegt und gelten ausschließlich für das Attribut selbst.

Ausgehend von der Vererbungsregel für Namensräume, sowie der Präzedenz expliziter Präfixangaben lassen sich daher folgende Auswertungsregeln definieren:

Ein Element befindet sich in demjenigem Namensraum ...

1. ... an den das vorangestellte Präfix gebunden ist.
   Verfügt das Element über kein Namensraumpräfix, so befindet es sich in demjenigen Namensraum ...
2. ... der auf diesem Element durch Überschreibung des Vorgabenamensraumes definiert wurde.
   Findet für dieses Element keine Überschreibung des Vorgabenamensraumes statt, so befindet es sich in demjenigen Namensraum ...
3. ... der für das Elternelement gilt, sofern er dort Vorgabenamensraum ist.
   Man beachte: Das *gilt* im vorangehenden Satz umschließt sich nicht nur die Überschreibung des Vorgabenamensraumes im direkten Elternelement, sondern auch eine dort geltende Namensraumüberschreibung die in dessen Elternelement oder dessen Elternelement ... stattfand.
   Findet in keinem der Elternelemente eine Überschreibung des Vorgabenamensraumes statt, so befindet sich das Element in demjenigen Namensraum ...
4. ... der leer ist (d.h. im leeren Namensraum).

Ein Attribut befindet sich in demjenigem Namensraum, der durch explizite Präfixangabe festelegt wurde.

Internationale URIs und Namensraumidentifikatoren:
Die Berücksichtigung von Zeichen, die in XML v1.1 zugelassenen, deren Nutzung in den klassischen URIs nach RFC 2396 bzw. RFC 2732 jedoch untersagt ist, führt zur Einführung des neuen Begriffes des *Internationalized Resource Identifier*s (IRI). Diese Neuschöpfung stellt im Kern eine URI-Fassung dar innerhalb der Leerzeichen sowie diverse Sonderzeichen zulassen sind. Diese internationalisierten Identifikatoren werden durch einen im Spezifikationsentwurf festgelegten Algorithmus in syntaktisch korrekte URIs umgewandelt.
Beispiel 24 zeigt gültige IRIs und jeweils dahinter in Klammern angegeben die daraus resultierende URI-Darstellung.

Beispiel 24:

```
(1)http://www.{iri-}example.com (http://www.%7Biri-%7Dexample.com)
(2)mailto:marc léon@example.org (mailto:marc%20l%E9on@example.org)
```

Kompatibilität zu älteren Dokumenten:
Elemente, für die weder ein expliziter Namensraum durch Präfix definiert ist, noch ein Namensraum von einem Elternelement übernommen werden kann, sind einem leeren Namensraum zugeordnet; konzeptionell entspricht dies einem NULL-Präfix.
Somit befinden sich alle Elemente, die keinem Namensraum angehören, automatisch in einem gemeinsamen Namensraum, der an keine URI gebunden ist.

Zusammenfassend gelten somit folgende Prinzipien:

- Jede beliebige URI kann an eigendefinierte Präfixe gebunden werden.
  Insbesondere kann dieselbe URI an verschiedene Präfixe gebunden werden, und dasselbe Präfix in verschiedenen Kontexten für verschiedene URIs stehen.
- Jedes Element übernimmt den überschriebenen Vorgabenamensraum seines Elternelements, sofern es keinen eigenen definiert.
- Elemente oder Attribute ohne Namensraumzuordnung (die auch keine von ihrem hierarchisch höherstehenden Element übernehmen) befinden sich im Standardnamensraum.
- Attribute ohne Namensraumprefix befinden sich im leeren Namensraum.

Web-Referenzen 7: Weiterführende Links

•XML-Namespace Recommendation
•Namespace Recommendation in deutscher Übersetzung
•Namespace Tutorial @ Zvon.org
•Tim Bray: Namespaces by Example
•Hintergrundartikel: Namespaces in XML Adopted by W3C
•(Tutorial) Simon St. Laurent: Namespaces in XML
•Roland Bourret: XML Namespaces FAQ

1.4 Dokument-Typ-Definitionen

Die Dokument-Typ-Definition (DTD) stellt eine an die EBNF angelehnte reguläre Sprache zur Formulierung von XML-Grammatiken zur Verfügung.
Ursprünglich wurde diese textuelle Notation zur Spezifikation von XML-Strukturen mit SGML eingeführt. Zur Verwendung in XML wurde der Sprachumfang lediglich reduziert, und um einige -- im neuen Anwendungskontext nicht benötigte -- Konstrukte bereinigt.
Inzwischen haben die sehr stark Dokumenten-orientierten DTDs gegenüber den XML-Schemasprachen an Bedeutung verloren. Mittelfristig ist, wegen der deutlich gesteigerten Ausdrucksmächtigkeit, mit dem vollständigen Übergang zu Schemasprachen zu rechnen.
Nachfolgend sollen jedoch die Grundzüge des DTD-Mechanismus vorgestellt werden, um seiner historischen Bedeutung gerecht zu werden. Ebenso soll hierdurch ein Grundverständnis für die existierenden Sprachdefinitionen geweckt werden.

Dokumenttypdeklaration:
Prinzipiell stellt die DTD eine eigenständige, vom XML-Dokument losgelöste, Informationseinheit dar. Ihre Rolle entspricht der der *Klasse* in der objektorientierten Programmierung, welche schablonenartig die Struktur und die inhaltlichen Charakteristika beliebig vieler konkreter Ausprägungen festlegt. In dieser Analogie agiert das XML-Dokument wie ein *Objekt* einer spezifischen Klasse.
Daher ergibt sie die folgende Definition als Erweiterung der Wohlgeformtheit fast intuitiv:

Definition 9: Gültiges XML-Dokument
Ein XML-Dokument heißt *gültig* (*valid*), wenn es über eine Dokument-Typ-Definition verfügt, und konform zu dieser aufgebaut ist.
(In XML-Spezifikation nachschlagen)

Implizit folgt hieraus, daß die Wohlgeformtheit eine schwächere Stufe der Gültigkeit, und damit Voraussetzung hierfür, darstellt.
Entsprechend der Unterscheidung in *well-formed* und *valid* ergeben sich zwei Klassen von XML-Prozessoren: nicht-validierende und validierende. Beide Typen prüfen die Wohlgeformtheit des eingehenden Textstromes. Zusätzlich testet ein validierender Prozessor die Konformität des XML-Dokuments zu seiner DTD.
(In XML-Spezifikation nachschlagen)

Die am Markt verfügbaren XML-Editierwerkzeuge beinhalten zumeist einen validierenden Parser, der die Konformitätsprüfung zu einer gegebenen DTD erlaubt. Ferner bieten viele Hersteller, oder Open-Source-Initiativen, eigenständige Parser-Module zur Validierung von XML-Dokumenten an.

Üblicherweise liegt die DTD in Form einer separaten (Text-)Datei vor, welche durch das XML-Dokument referenziert wird. Zu diesem Zwecke kann im Prolog des XML-Dokuments durch das Schlüsselwort DOCTYPE die physische Lokation der DTD angegeben werden.
(In XML-Spezifikation nachschlagen)

Die Syntax einer DOCTYPE-Deklaration lautet (In XML-Spezifikation nachschlagen) :

```
[28]  doctypedecl ::= '<!DOCTYPE' S Name (SExternalID)?
                      S? ('[' (markupdecl | DeclSep)* ']' S?)? '>'
[28a] DeclSep     ::= PEReference | S
[29]  markupdecl  ::= elementdecl | AttlistDecl | EntityDecl | NotationDecl
                      | PI | Comment
```

Die von SGML übernommene Syntax erinnert äußerlich leicht an die im XML-Dokument verwendete, weist jedoch mehr Ähnlichkeit zum Aussehen der Kommentare in XML auf.
So wird auch die DOCTYPE-Deklaration durch eine öffnende Winkelklammer und ein Ausrufezeichen eingeleitet. Daran schließt sich -- im Falle einer externen DTD -- die SYSTEM- oder PUBLIC-Deklaration an. XML-Dokument-interne Grammatikdefinitionen werden in eckigen Klammern innerhalb der DOCTYPE-Deklaration eingefügt.

Anwendungsbeispiele finden sich im Abschnitt zum *Document Type Definition Information Item* des Information Sets.

Alternativ zur externen Ablage der DTD kann diese auch ganz oder teilweise in das XML-Dokument eingebettet werden. Dieser Anteil der DTD innerhalb des XML-Dokuments wird als *internal subset* bezeichnet. Konsequenterweise wird der extern des Dokuments abgelegte DTD-Anteil daher mit dem Begriff *external subset* belegt.
Beispiele zur Definition interner Subsets finden sich in den Betrachtungen der XML-Information Set Items zu Notationen und Entitäten.
Anmerkung: Existieren in einem internen und externen Subset Festlegungen zum selben (identisch benannten) Informationselement, so überschreibt die des internal Subsets die extern getroffenen Definitionen.

Wie Produktion 29 bereits andeutet, erlaubt der DTD-Mechanismus die Strukturdefinition verschiedener Elemente aus dem XML-Information Set. Die wesentlichen hierbei sind: Elemente und Attribute. Ferner können durch die DTD Entitäten, Notation und Processing Instructions definiert werden.

Web-Referenzen 8: Weiterführende Links
- JAXP: SUNs Standard-Parsing-API für Java
- XML Instance -- Ein validierender Dokumenteditor
- XML Authority -- Ein DTD-Editor
- XML Spy -- Dokument- und DTD-Editor

Definition von Elementen:
Der offensichtlichste Anteil der Struktur eines XML-Dokuments wird durch seine Elemente gebildet. Ein einzelnes Element stellt auch (siehe Definition im Information Set) den minimalsten Umfang eines gültigen XML-Dokuments dar.
(In XML-Spezifikation nachschlagen)

Die Syntax zur Definition eines Elements lautet (In XML-Spezifikation nachschlagen) :

```
[45] elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>'
[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children
[51] Mixed       ::= '(' S? '#PCDATA' (S? '|' S? Name)* S? ')*'
                   | '(' S? '#PCDATA' S? ')'
```

Jedes Element wird durch die SGML-Syntax der öffnenden Winkelklammer, gefolgt von einem Ausrufezeichen, eingeleitet. Daran schließt sich der Elementname an.
In runden Klammern wird dann das *Inhaltsmodell*, d.h. die erlaubten Kindelemente, spezifiziert. Sind keine Kindelemente erlaubt, d.h. es handelt sich um ein leeres Element, so ist stattdessen das Schlüsselwort EMPTY anzugeben.
Eine Besonderheit bildet das durch ANY ausgedrückte Inhaltsmodell. Es erlaubt das beliebige Auftreten von Elementen aus der DTD. Ebenso ist textueller Inhalt (#PCDATA) zugelassen.

Zulässige Kindelemente sind neben allen namentlich definierten Elementen der DTD auch Freitexte (#PCDATA für *parsed character data*, die keine Auszeichnungssymbole enthalten dürfen). Sollen im Elementinhalt Zeichen verwendet werden, die auch als Metasprachensymbole dienen -- wie < oder & --, so müssen diese durch die vordefinierten Textersetzungsmuster ausgedrückt werden.
Über freie Texte und explizite Kindelemente hinaus bietet der DTD-Mechanismus keinerlei Unterstützung zur näheren Definition der Inhalte eines Elements an. Diese, gerade bei Daten-orientierten XML-Dokumenten schmerzhafte, Einschränkung hat mit zur Definition der XML-Schemasprachen geführt, die sich u.a. die Überwindung dieser Einschränkung zum Ziel setzen.
Die Kindelemente werden durch namentliche Aufzählung in der Reihenfolge ihres im Dokument erwarteten Auftretens benannt. Zur Gruppierung können Elemente durch Klammerung zusammengefaßt werden.
Zur Steuerung der Auftrittshäufigkeit von Elementen und Elementgruppen existieren drei Operatoren.

- Das Fragezeichen ? deklariert ein Element oder eine Elementgruppe als optional.
- Der positive Abschluß + erlaubt die Wiederholung desselben Elements; mindestens ein Element muß jedoch im XML-Dokument auftreten.
- Der Kleene-Stern * erlaubt die beliebige Wiederholung desselben Elements, wobei auch das erste auftreten optional ist.

Als Kombination aus explizit benannten Elementen und beliebigen Markup-freien Texten ergibt sich das *gemischte Inhaltsmodell*.
Aus historischen Gründen folgt dieses Inhaltsmodell immer derselben Struktur (siehe Produktion 51): Auf das Schlüsselwort #PCDATA folgen die zugelassenen Elemente in exklusiver ODER-Beziehung.
Hinter der zunächst etwas kryptisch anmutenden Formulierung verbirgt sich ein kleiner Kunstgriff, um diese zunächst widersprüchlich anmutende Dualität zwischen markupfreiem (#PCDATA) und explizit ausgezeichnetem Inhalt aufzuheben. Das für diesen Anwendungsfall in der XML-Spezifikation vorgegebene mixed content Model erlaubt eine Mischung aus frei formulierter Information und Markupinhalten. Hierzu werden #PCDATA und das gewünschte Element zunächst exklusiv zueinander deklariert (symbolisiert durch den trennenden senkrechten Strich |); im Anschluß wird - durch den Stern -- die beliebig häufig hintereinandergereihte Wiederholung dieser Auswahlentscheidung vereinbart.
(In XML-Spezifikation nachschlagen)

Definition 10: Gemischtes Inhaltsmodell

Kann ein Element sowohl über Zeichenketten-artigen als auch Element-wertigen Inhalt verfügen, so wird sein Inhaltsmodell als *gemisches Inhaltsmodell* (*mixed content model*) bezeichnet.
Innerhalb eines solchen Elements dürfen Unicodezeichen und die zugelassenen Elemente in wahlfreier Kombination auftreten.

Nachfolgend wird eine einfache Projektverwaltung, zur Organisation von Mitarbeitern und ihrer Zuordnung zu Projekten, als durchgängiges Beispiel verwendet.

Beispiel 25: Einige Elementdefinitionen

```
(1)<!ELEMENT ProjektVerwaltung (Person+, Projekt+)>
(2)<!ELEMENT Person (Vorname+, Nachname, Qualifikationsprofil?)>
(3)<!ELEMENT Qualifikationsprofil (#PCDATA | Qualifikation | Leistungsstufe)*>
(4)<!ELEMENT Qualifikation (#PCDATA)>
(5)<!ELEMENT Leistungsstufe (#PCDATA)>
(6)<!ELEMENT Vorname (#PCDATA)>
(7)<!ELEMENT Nachname (#PCDATA)>
(8)<!ELEMENT Projekt EMPTY>
```

Download des Beispiels

Das Beispiel zeigt einige verschiedene Elementdefinitionen.
So kann ein Element des Typs `ProjektVerwaltung` mehrere, jedoch mindestens ein `Personen`-Element sowie auch mehrere `Projekt`-Elemente enthalten.
Innerhalb von `Personen`-Elementen können mehrere, wiederum jedoch mindestens ein `Vornamen` und genau ein `Nachname` auftreten. Das Auftreten des mit `Qualifikationsprofil` benannten Kindelements ist optional.
Über ein leeres Inhaltsmodell (Schlüsselwort `EMPTY`) verfügt das Element `Projekt`. Ihm dürfen keine Kindelemente im XML-Dokument folgen.
Die drei Elemente `Ueberschrift`, `Vorname` und `Nachname` erlauben jeweils das Auftreten markupfreien Textes in ihrem Elementinhalt.
Für `Qualifikationsprofil` ist das mixed content model verwirklicht. Es erlaubt in XML-Dokument das Auftreten des Elements `Qualifikation` oder `Leistungsstufe` an jeder beliebigen Stelle innerhalb eines (markupfreien) Freitextes.

Beispiel 26: Beispieldokument zur gegebenen DTD

```
(1)<?xml version="1.0" encoding="ISO-8859-1"?>
(2)<!DOCTYPE ProjektVerwaltung SYSTEM "http://www.jeckle.de/vorlesung/xml/examples/
projektverwaltung1.dtd">
(3)<ProjektVerwaltung>
(4)   <Person>
(5)   <Vorname>Hans</Vorname>
(6)   <Nachname>Hinterhuber</Nachname>
(7)   </Person>
(8)   <Person>
(9)     <Vorname>Franz</Vorname>
(10)    <Vorname>Xaver</Vorname>
(11)    <Nachname>Obermüller</Nachname>
(12)    <Qualifikationsprofil>
(13)    IT-Kompetenz verschiedene Betriebssysteme und <Leistungsstufe>professionelle</
Leistungsstufe> <Qualifikation>Programmierung</Qualifikation> verschiedener Programmiersprachen
(14)    <Qualifikation>Entwickler</Qualifikation> von 1988-1990
(15)    <Qualifikation>Projektleiterfunktion</Qualifikation> von 1990-93 im X42-Projekt in
Abteilung AB&amp;C
(16)   </Qualifikationsprofil>
(17)   </Person>
(18)   <Projekt/>
(19)</ProjektVerwaltung>
```

Download des Beispiels

Das Beispiel zeigt ein gültiges XML-Dokument zu den bisherigen Elementdefinitionen.
Es zeigt die Nutzung des mehrfachen Auftretens des Elements `Vorname` innerhalb der `Person`, sowie die Anwendung des mixed content models -- einschließlich Einsatz des Textersetzungsmusters des Und-Symbols -- für das `Qualifikationsprofil`.

Definition von Attributen:
Die Syntax der Attributdefinition ist wie folgt festgelegt:

```
[52] AttlistDecl    ::= '<!ATTLIST' S Name AttDef* S? '>'
[53] AttDef         ::= S Name S AttType S DefaultDecl
[54] AttType        ::= StringType | TokenizedType | EnumeratedType
[55] StringType     ::= 'CDATA'
[56] TokenizedType  ::= 'ID' | 'IDREF' | 'IDREFS' | 'ENTITY' | 'ENTITIES' |
                        'NMTOKEN' | 'NMTOKENS'
[57] EnumeratedType ::= NotationType | Enumeration
[59] Enumeration    ::= '(' S? Nmtoken (S? '|' S? Nmtoken)* S? ')'
[60] DefaultDecl    ::= '#REQUIRED' | '#IMPLIED'
                        | (('#FIXED' S)? AttValue)
```

Auch die Syntax der Attributdefinition in der DTD ist SGML entlehnt; daher auch hier die öffnende Winkelklammer und das Ausrufezeichen als einleitendes Symbol. Darauf folgt der Name des Elements, an die die Attributliste gebunden wird. Leicht wird ersichtlich, daß Attribute nicht wie Elemente vollkommen frei definiert werden können, sondern immer im Kontext des nutzenden Elements stehen. In der Folge ist es daher unmöglich, dieselbe Attributdefinition mehreren Elementen zuzuordnen.
An den Namen des Attribut-beinhaltenden Elements schließen sich die eigentlichen Deklarationen an. Im einfachsten Fall bestehen sie lediglich aus dem eindeutigen Attributnamen, einem Datentyp (gemäß der Produktionen 55 und 56) sowie der Angabe, ob es sich um ein optionales (`#IMPLIED`) oder zwingend anzugebendes (`#REQUIRED`) Attribut handelt.
Das Angebot der zur Verfügung stehenden Datentypen spiegelt die präsentationsorientierte Vergangenheit der Vorgängersprache SGML wieder. So fehlen Programmiersprachen-orientierte Primitivtypen vollständig. Das mit `CDATA` -- für *character data* -- benannte Äquivalent zum Elementinhaltstyp `#PCDATA` erlaubt lediglich die Beschränkung der Attributwerte auf Zeichenketten-artige Inhalte.
In Erweiterung der `IMPLIED`-Angabe können Attribute als mit Vorgabewerten belegt und zusätzlich als konstant (`#FIXED`) deklariert werden. Ihre vorgegebenen Werte werden, auch bei fehlender Angabe im XML-Dokument, durch einen validierenden XML-Parser an die verarbeitende Applikation weitergegeben. Darüberhinaus prüft der Parser für Konstanten, ob der angegebene Attributwert mit der Vorgabe übereinstimmt.
Eine besondere Bedeutung kommt den Typen `ID`, `IDREF` und `IDREFS` zu. Durch sie ist ein rudimentärer Refenzierungsmechanismus, für den die Gültigkeit der Referenzen durch einen validierenden XML-Prozessor geprüft werden kann, verwirklicht. Alle Attributwerte des Typs `ID` bilden einen Dokument-weiten Namensraum (der jedoch außer Verwendung desselben Begriffs nicht mit den XML-Namespaces in Verbindung steht!). Auf diese Schlüssel kann durch Attribute des Typs `IDREF` und `IDREFS` verwiesen werden. `IDREF` realisiert hierbei eindeutige Verweise, während `IDREFS` die Angabe einer Liste von Verweiszielen erlaubt. Im einheitlichen Namensraum aller definierten `ID`-Attribute liegt eine der Hauptschwächen des angebotenen Referenzierungsmechanismus. So ist eine typisierende Einschränkung der erlaubten Referenzziele nicht vorgesehen. Da IDREF(S)-Attribute alle Werte annehmen können, die dokumentweit in ID-Attributen auftreten, kann potentiell auf jede definierte ID verwiesen werden, unabhängig davon ob dieser Verweis semantisch sinnvoll ist. Darüberhinaus können -- insbesondere in großen Dokumenten -- durch die fehlende Partitionierung des ID-Namensraumes Konflikte (und damit invalide Dokumente) durch mehrfachauftretende IDs auftreten. Auch die Teilung von Dokumenten in ID-konfliktfreie Einheiten stellt hierbei keine Lösung dar, da der Referenzierungsmechanismus nicht dokumentübergreifend angelegt ist und daher nicht auf Elemente anderer XML-Quellen zugreifen kann.
Als Lösung dieser angerissenen Problemstellungen bieten sich neben den (deutlich mächtigeren) Referenzierungsmechanismen der XML-Schemasprachen auch dokumentübergreifende Verweise -- wie u.a. in der im Kapitel 2.1 diskutierten W3C-Sprache XLink verwirklicht -- an.
Eine einfache Variante anwenderdefinierter Datentypen werden durch Aufzählungstypen angeboten. Sie werden durch Benennung der Alternativen gebildet. Syntaktisch werden diese, durch Oder-Verknüpfungen voneinander abgetrennt, in runden Klammern dem Attributnamen nachgestellt.

Das nachfolgende DTD-Beispiel erweitert die betrachteten Elementdeklarationen um Attributdefinitionen.

Beispiel 27: Vollständige Beispiel-DTD

```
(1)<!ELEMENT ProjektVerwaltung (Person+, Projekt+)>
(2)<!ATTLIST ProjektVerwaltung
(3)     version CDATA #FIXED "1.0">
(4)
(5)<!ELEMENT Person (Vorname+, Nachname, Qualifikationsprofil?)>
(6)<!ATTLIST Person
(7)     PersID ID #REQUIRED
(8)     Gehaltsgruppe (1 | 1a | 2) "1a"
(9)     mitarbeitInProjekt IDREFS #REQUIRED>
(10)
(11)<!ELEMENT Qualifikationsprofil (#PCDATA | Qualifikation | Leistungsstufe)*>
(12)<!ELEMENT Qualifikation (#PCDATA)>
(13)<!ELEMENT Leistungsstufe (#PCDATA)>
(14)<!ELEMENT Vorname (#PCDATA)>
(15)<!ELEMENT Nachname (#PCDATA)>
(16)<!ELEMENT Projekt EMPTY>
(17)<!ATTLIST Projekt
(18)    ID ID #REQUIRED
```

```
(19)    date CDATA #IMPLIED
(20)    budget CDATA "10000"
(21)    Projektleiter IDREF #REQUIRED
(22)    Mitarbeiter IDREFS #REQUIRED>
```

Download des Beispiels

Das Beispiel zeigt verschiedene Attributdeklarationen. Zunächst `date` als ein simples optionales Zeichenkettenattribut vom Typ `CDATA`.
`budget` und `version` stellen die beiden möglichen Erweiterungen, einerseits durch Angabe eines Vorgabewertes (in Anführungszeichen dem Datentyp nachgestellt), andererseits durch zusätzliche Erklärung des Vorgabewertes als konstant (`#FIXED`-Angabe), dar.
Die Möglichkeit eines anwenderdefinierten Aufzählungstyps ist in `Gehaltsgruppe` genutzt. Die Alternative `1a` ist darüberhinaus als Vorgabewert definiert.
Zwischen den einzelnen `Projekten` und verwalteten `Personen` können über die Attribute `PersID` bzw. `ID` und `mitarbeitInProjekt` bzw. `Projektleiter` und `Mitarbeiter` Referenzierungsbeziehungen aufgebaut werden. Auch hier zeigen sich zwei Einschränkungen der ID/IDREF-Semantik deutlich: So können für `Personen` und `Projekte` nicht Identifikationsschlüssel aus demselben Namensraum vergeben werden, da diese bei Doppelbelegung kollidieren. Bei automatisch generierten Attributwerten ist daher applikationsseitig sicherzustellen, daß jeder Wert nur höchstens einmal im Dokument auftritt. Da eine spätere Vereinigung von Dokumenten nicht auszuschließen ist, verschärft sich diese Forderung zu über Zeit und Raum eindeutigen Attributbelegungen.
Zusätzlich wird deutlich, daß innerhalb des `Projekt`-Elements nicht sicherzustellen ist, daß die Attribute `Projektleiter` und `Mitarbeiter` ausschließlich auf Attribute innerhalb des `Personen`-Elements verweisen. So wäre die Zuordnung einer innerhalb von `Projekt` vergebenen ID zum Attribut `Projektleiter` desselben Elements korrekt und würde durch einen validierenden Parser akzeptiert.

Definition von Entitäten und Notationen:
Die Document-Type-Definition bietet die Möglichkeit, eigene Textersetzungsmuster (sog. Entitäten) durch den Anwender zu definieren. Dieses Sprachmerkmal hat jedoch -- wegen verschiedenster Probleme -- keinen Eingang in den XML-Schemamechanismus gefunden. Daher sei es hier nur der Vollständigkeit halber einführend erwähnt. Hinzuweisen ist jedoch bereits jetzt auf die daher abzusehende Problematik beim Übergang von existierenden DTDs zu gleichmächtigen Schemadefinitionen, da ein äquivalentes Sprachmittel dort nicht mehr zur Verfügung stehen wird. Daher sollte bei neu zu erstellenden XML-Grammatiken ganz auf die Definition eigener Entitäten verzichtet werden.

Die XML-Spezifikation definiert selbst fünf Entitäten: die bekannten Textersetzungsmuster der Metasprachensymbole. Daher ist die Referenzierungssyntax (einleitendes &-Symbol gefolgt vom Namen der Entität und abschließendem Semikolon) bereits geläufig.
Zur Definition einer Entität stehen die im Abschnitt *Unexpanded Entity Reference Information Item* vorgestellten Strukturen zur Verfügung.
Dort finden sich auch Beispiele eigener Definitionen.

Die in enger Verbindung mit den Entitäten stehenden Notationen zur Realisierung von Ersetzungsmustern, die keine XML-codierten Inhalte (z.B. Binärdaten) beinhalten, sind beispielhaft sowie in Syntax und Semantik im Abschnitt *unparsed Entities* vorgestellt.

1.5 XML Schemasprachen

Neben den Document Type Definitions ist in jüngerer Zeit ein alternativer Ansatz in den Blickpunkt des Interesses gerückt: die XML-Schemasprachen.
Sie setzen die Emanzipation der Metasprache XML von ihrer Vorgängersprache SGML fort. Bereits in engem zeitlichem Bezug zur Veröffentlichung der XML-Recommendation wurde mit *XML Data* ein erster Ansatz vorgestellt. In der Zwischenzeit fanden verschiedene konkurrierende Vorschläge ein breites Interesse. Übereinstimmende Zielsetzung aller verschiedenen vorgeschlagenen Schemasprachen ist die Schaffung eines Sprachdefinitionsmechanismus, der die Dokumenten-orientierten Strukturen und Inhaltsmodelle der DTD überwindet. An die Spitze der Bemühungen setzte sich eine Arbeitsgruppe des W3C zur Definition einer XML-Schemasprache, unter Berücksichtigung der bekanntesten und verbreitetsten Vorschläge. Durch sie wurde im Mai 2001 der *XML Schema*-Standard des W3C veröffentlicht.

Der Begriff *Schema* ist der im Datenbankumfeld gebräuchlichen Terminologie entlehnt. Dort bezeichnet er Informations- oder Datenmodelle als Konstruktionsvorlage oder Dokumentation eines Datenbankdesigns. Hierzu muß ein Schema nicht unbedingt in einer graphischen Datenmodellierungssprache vorliegen, sondern kann beispielsweise auch die Tabellenstruktur einer relationalen Datenbank bezeichnen.

Zur Notwendigkeit einer Schemasprache:
Zum Zeitpunkt der Konzeption der Metasprache SGML war das Anwendungsfeld klar umrissen und im wesentlichen auf die Digitalisierung vormals papiergestützter Dokumentation festgelegt. Daraus erklärt sich auch die Mächtigkeit der Document Type Definition, der angebotenen Grammatiksprache zur Darstellung der Dokumentstrukturen. Insbesondere war weder die Daten-orientierte Verwendung von SGML, noch die rund 30 Jahre später einsetzende Weiterentwicklung (eigentlich: Reduktion) zur eXtensible Markup Language abzusehen.
Die inzwischen eingesetzte breite Anwendung von XML-Sprachen zur Darstellung beliebiger Inhalte läßt jedoch die Beschränkungen und Unzulänglichkeiten des DTD-Mechanismus für diesen Anwendungen offenkundig werden.

Nachfolgend sind einige der durch Nutzung des DTD-Mechanismus zur Beschreibung Daten-intensiver Strukturen induzierten Einschränkungen zusammengestellt:

- Unzureichende Datentypunterstützung.

DTDs erlauben für Elemente nur die vier im vorigen Kapitel diskutierten Inhaltsmodelle: `child elements`, `PCDATA`, *mixed content* sowie das leere Inhaltsmodell `EMPTY`.
Für die Attributdefinition (siehe Bemerkungen im vorhergehenden Kapitel) stehen die in Produktion 54 angegebenen Typen zur Verfügung.
Genaugenommen werden jedoch hier nur Zeichenketten-artige Datentypen (explizit für `CDATA`, implizit für `ID` (In XML-Spezifikation nachschlagen) , `IDREF`, `IDREFS` (In XML-Spezifikation nachschlagen) , `ENTITY`, `ENTITIES` (In XML-Spezifikation nachschlagen) bzw. `NMTOKEN` und `NMTOKENS` (In XML-Spezifikation nachschlagen) durch die Beschränkung der zulässigen Inhalte auf Satzformen der Produktion 5 bzw. Produktion 7 der XML-Recommendation) angeboten.
Die manchmal anzutreffende (Nach-)Modellierung benötigter Datentypen durch anwenderdefinierte Aufzählungstypen ist zeitaufwendig und fehlerträchtig.

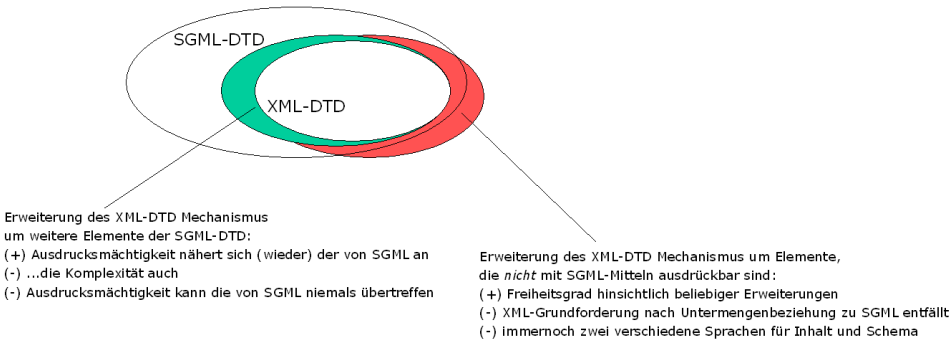
- Unzureichende Strukturierungsunterstützung.
  Zwar erlauben die vorgestellten Operatoren zur Steuerung der Auftrittshäufigkeit einzelner Kindelemente die Codierung beliebiger Kardinalitäten. Allerdings sind die hierfür anzuwendenden Prinzipien teilweise umständlich in der Schreibung, und daher fehlerträchtig. Die Lesbarkeit der entstehenden DTD wird entscheidend gesenkt.
- Keine Unterstützung von Wiederverwendbarkeit.
  Während Elementstrukturen (zumindest) innerhalb der definierenden DTD beliebig wiederverwendet werden können, sind Attribute immer an das umgebende Element gebunden (vgl. `ATTLIST`-Konstrukt).
  Eine Nutzung in anderen Dokument-Typ-Definitionen als der definierenden ist nicht vorgesehen. Zwar läßt sich dies durch die Definition von Entitäten annähernd nachbilden. Jedoch ist dies bereits zum Erstellungszeitpunkt der DTD geeignet zu berücksichtigen.
  Modularisierung im Sinne einer Wiederverwendung der durch die Elemente definierten Typen innerhalb der DTD, beispielsweise zur Definition weiterer Typen, wird lediglich durch *Parameter Entitäten* unterstützt. Sie verhalten sich für die DTDs wie die Entitäten in XML-Dokumenten.
- Starres Typsystem.
  Das angebotenen Typsystem kann durch den Anwender nicht erweitert werden.
- Keine Unterstützung von Namensräumen.
  Namensräume können in der DTD nicht angegeben werden.
  Anmerkung: Die hierfür oft angeführten Hilfskonstrukte spiegeln weder die volle Mächtigkeit, noch die korrekte Semantik der Namensräume wieder.
- Nur rudimentärer Referenzierungsmechanismus.
  Die offerierten `ID-IDREF(S)`-Verknüpfungen sind ausschließlich Dokument-lokal möglich und gestatten keine Differenzierung hinsichtlich der Semantik des eindeutig identifizierten oder referenzierten Elements.
- DTD-Syntax ist nicht XML.
  Die von SGML übernommene Syntax der DTD bildet eine eigene Sprache, die von Werkzeugen gesondert zu implementieren ist.

Technische Ansätze:
Prinzipiell lassen sich die in der Vergangenheit vorgeschlagenen Ansätze zur Definition einer Schemasprache in vier Kategorien unterscheiden:

1. Orientierung am bestehenden DTD-Mechanismus.
   Erweiterungen des bestehenden Mechanismus um zusätzliche Sprachelemente.
2. Orientierung an der programmiersprachlichen Interpretation.
   Versuch XML und ein Ausführungsmodell möglichst eng zu koppeln.
3. Orientierung an Wissensdarstellungen
   Interpretation des Schemas einer XML-Sprache als Wissen über die Sprache.
4. XML-Sprachen zur Inhaltsbeschreibung.
   Da XML i.A. zur Beschreibung beliebigster Informationen herangezogen werden kann, ist die Verwendung auch für die Beschreibung von XML-Strukturen denkbar.

Die naheliegendste Option dürfte die Erweiterung des bestehenden DTD-Sprachumfanges bilden. Durch geeignete Modifikationen und Ergänzungen ließen sich alle, mit Ausnahme der letzten, identifizierten Unzulänglichkeiten beheben.
Konzeptionell lassen sich zwei Erweiterungsvarianten aufzeigen. Zunächst die Möglichkeit, die XML-DTDs um Elemente der ursprünglichen SGML-DTD zu erweitern. In der Konsequenz nähert sich XML, positiv formuliert, wieder der Ausdrucksmächtigkeit der Ursprache SGML an. Negativ formuliert, kann jedoch XML auf diesem Wege niemals Inhaltsstrukturen ausdrücken, die nicht durch SGML ausdrückbar sind, da die Mächtigkeit des SGML-DTD-Mechanismus eine natürliche Obergrenze der Erweiterbarkeit darstellt. Zusätzlich ist anzumerken, daß ein solcher Ansatz der ursprünglichen Intention der XML-Entwicklung -- ein leichter einsetzbares SGML zu schaffen -- entgegenläuft.
Eine der bekannten Ideen zur Erweiterung des DTD-Mechanismus stellt *Datatypes for DTDs* (DT4DTD) dar.
Alternativ zur Erweiterung hin zur SGML-Mächtigkeit ließe sich der bestehende XML-DTD-Mechanismus um neue zusätzliche Konstrukte anreichern, die nicht Bestandteil der SGML-DTD-Syntax sind. Dieser Ansatz böte den Vorteil, den Vorgängerstandard nicht berücksichtigen zu müssen und beliebige Erweiterungen in Syntax und Semantik einbringen zu können. Allerdings würde damit eine zentrale Forderung der XML-Entwicklung, die sich bereits im Abstract der XML-Recommendation findet, nicht berücksichtigt: die Untermengenbeziehung zu SGML. Durch eine Erweiterung, welche über die SGML-Mächtigkeit hinausreicht, würden legale (*well formed* und sogar *valid*) XML-Dokumente entstehen, die keine gültigen SGML-Dokumentinstanzen wären.

Die nachfolgende Graphik veranschaulicht die beiden Erweiterungsoptionen und die Argumente der geführten Diskussion.



Die im zweiten Punkt angedeutete Umsetzung ist durch eine programmiersprachliche Verarbeitung der XML-Dokumente motiviert. Aus Sicht dieser Anwendungsfacette ist ein Schemamechanismus idealerweise so ausgelegt, daß er die transparente Umsetzung in Applikationsdatenstrukturen ermöglicht. Dahinter steht der Wunsch, den *impedance mismatch*, mithin den zu leistenden Abbildungsaufwand zwischen XML-Konstrukten und Datenstrukturen, möglichst gering zu halten.
Beispielsweise greift der -- durch den Einsatz im e-Commerce-System der Firma CommerceOne bekannt gewordene -- Vorschlag *Schema for Object-Oriented XML* (SOX) zur Definition der notwendigen Semantik der angebotenen Schemaprimitiven auf die bekannte plattformunabhängige Programmiersprache Java zurück.
Die aktuelle Version der Schemasprache SOX, die zur Definition der XML-Sprache xCBL eingesetzt wird, findet sich unter xCBL.org.

Der dritte technische Ansatz weist auf eine alternative Interpretation der XML-Grammatikstruktur hin. So spiegelt ein Schema auch immer *Wissen* über Struktur und Inhalt eines betrachteten Problembereichs wieder.
Der bekannteste Vorschlag -- die *Document Content Description* (DCD) -- nutzt zur Definition der Wissensstrukturen eines XML-Dokuments das *Resource Description Framework* (RDF) des World Wide Web Consortiums.
Der Ansatz hat sich durch Referenzimplementierungen durchaus als tragfähig und, wegen der RDF-basiertheit, als allgemein verwendbar erwiesen. Jedoch liegt hierin auch die offensichtlichste Limitierung. RDF als Metasprache der Schemasprache legt bereits eine gewisse Strukturierung aller Schemata zugrunde, da jedes gültige DCD-Schema definitionsgemäß ein RDF-Dokument darstellt. Ebenso ist die Semantik der eingesetzten RDF-Elemente bereits durch diese Spezifikation vorgegeben. Beide Punkte zusammengenommen offenbaren eine ausgeprägte Abhängigkeit von den weiteren RDF-Aktivitäten des World Wide Web Consortiums, die bisher nicht auf die Interdependenz von Schemasprache und Wissensbeschreibungsformat ausgerichtet ist.
Positiv fällt an DCD die Verwendung von XML zur Beschreibung von XML-Sprachen auf, womit auch die letzte der erhobenen Anforderungen zu erfüllen wäre.
Die Verknüpfung von RDF mit DCD als Schemasprache birgt allerdings ein potentielles Problem hinsichtlich der Validierbarkeit der entstehenden Strukturen. Durch den Rückgriff von DCD auf RDF entsteht bei der Angabe eines Schemas für RDF ein transitiver Zirkelschluß. In der Konsequenz wird zur Validierung eines XML-Dokuments, welches einer mittels DCD-formulierten Grammatik folgt, neben dem eigentlichen DCD-Schema des Dokuments auch das DCD-Metaschema und dessen Semantik-liefernde RDF-Beschreibung benötigt.

Diese Beschränkung mildert die vierte Familie von XML-Schemasprachen ab. Sie umfaßt die meisten Vorschläge, die alle als eigenständige XML-Sprachen ausgelegt sind; daher definieren sie ein eigenständiges XML-Vokabular zur Darstellung der benötigten XML-Strukturen, sowie die zugehörige Semantik.
In der Folge sind sie für die Meta-Schemaebene selbstbeschreibend. Das bedeutet das Schema eines Schemas kann durch sich selbst validiert werden. Da dieser Validierungsschritt statisch nur einmal erfolgen muß, kann er durch Schemawerkzeuge vorweggenommen werden.
In dieser Kategorie sind die meisten der bisher vorgeschlagenen Schemadialekte einzuordnen.

Die größte Bedeutung haben kontextfreie reguläre Sprachen zur Spezifikation von XML-Sprachstrukturen erlangt. Eine Sprache dieses Typs entwickelt auch die W3C-Arbeitsgruppe zur Definition eines XML-Schemasprachstandards. Insbesondere berücksichtigt diese Aktivität explizit die Vorgängersprachen XML Data, DCD, SOX sowie Document Definition Markup Language. Die erwähnten konkurrierenden Vorschläge unterscheiden sich semantisch lediglich in Nuancen, bieten dem Anwender jedoch teilweise (optisch) stark unterschiedliche Konstrukte zur Syntaxspezifikation an.

Einen strukturell unterschiedlichen Ansatz verfolgt die durch Rick Jelliffe vorgeschlagene Sprache *Schematron*. Sie interpretiert ein Schema als Sammlung von Regeln, denen ein gegebenes Dokument genügen muß, um als gültig akzeptiert zu werden. Dies erlaubt die Formulierung mächtiger konktextsensitiver Einschränkungen, die während des Validierungsvorganges geprüft werden.
Die Umsetzung dieser Schemasprache setzt auf den XML-Standards XPath und XSLT auf.

W3Cs XML-Schema:
Jenseits aller existierenden verschiedenen Sprachvorschläge kommt dem W3C-Standard der *XML Schema Description Language* (XSD) die größte praktische Bedeutung zu.
Tim Berners-Lee verkündete in der Eröffnungsrede der WWW-Konferenz in Hong Kong am 2. Mai 2001 die Verabschiedung als Recommendation. Gleichzeitig deutete er bereits weitere Schema-Aktivitäten des World Wide Web Consortiums an.
XML-Schema bildet zusammen mit XML v1.0 2nd edition und den Namensräumen die Basis aller weiteren W3C-XML-Sprachstandards.

Aus formalen Gründen ist nicht mit dem Ersatz der DTD durch Schema zu rechnen. Jedoch werden mittelfristig neu entwickelte XML-Sprachen keine Grammatiken mehr in der Syntax der DTD entwickeln, sondern direkt Schemata definieren.

XSD bildet eine vollständig in XML-Syntax formulierte kontextfreie reguläre Grammatik zur Formulierung beliebiger

XML-Strukturen ab. Hierbei handelt es sich um die bekannten Grundprimitive *Element* und *Attribut*, andere -- wie *Entitäten* oder *Notationen* -- können hingegen nicht durch Schemata ausgedrückt werden. Somit erfolgt durch XSD implizit eine Weiterentwicklung von XML, dahingehend, daß ursprünglich von SGML übernommene -- jedoch inzwischen als unpraktikabel oder potentiell fehlerträchtig angesehene -- Sprachbestandteile nicht mehr durch den Grammatikmechanismus unterstützt werden.
Gleichzeitig wurde, neben zahlreichen anderen Neuerungen, die Kommentarsyntax für Schemata neu definiert.

Inhaltlich gliedert sich der XSD-Sprachvorschlag in zwei große Teilbereiche: Part 1: Structures zur Definition von Inhaltsmodellen für Elemente, Attributstrukturen und wiederverwendbaren Strukturen und Part 2: Datatypes zur Festlegung diverser inhaltlicher Charakteristika wie Datentypen und konsistenzgarantierende Einschränkungen.
In beiden Teilen werden XML-Namensräume explizit berücksichtigt. Konzeptionell rekonstruiert XSD-Part1 zunächst die bekannte Mächtigkeit der DTD um so die evolutionäre Weiterentwicklung bestehender XML-Sprachen zu ermöglichen.
Der zweite Teil der XSD-Spezifikation definiert ein eigenständiges Typsystem, das neben der naheliegenden Verwendung im ersten Teil der Schemasprache XSD auch in anderen W3C-Arbeitsgruppen Verwendung findet. Inhaltlich baut auch Part2 auf den in der DTD definierten Typen auf und erlaubt zunächst direkt ihre Angabe in Schemata. Darauf aufbauend wird eine Fülle verschiedenster Typen angeboten, die an die verschiedenen verfügbaren Typsysteme aus den Programmiersprachen, Datenbanken und internationalen Standards angelehnt sind.
Alle durch XSD definierten Elemente, d.h. alle Primitive zur Definition eines eigenen Schemas, befinden sich im Namensraum `http://www.w3.org/2001/XMLSchema`, der üblicherweise an das Präfix `xsd` gebunden wird. Elemente und Attribute aus XML-Schema, die in Instanzdokumenten verwendet werden könne sind im Namensraum `http://www.w3.org/2001/XMLSchema-instance` (übliches Präfix `xsi`) organisiert.
Wegen des Umfanges der offiziellen Schemadokumente wird zusätzlich durch das W3C ein Part 0: Primer herausgegeben. Er stellt die beiden XSD-Teile in der Zusammenschau an Beispielen dar.

Nachfolgend werden die Bestandteile der Schemasprache XSD an einigen Beispielen eingeführt, die Struktur der Darstellung orientiert sich dabei an der bereits für die Vorstellung der DTD genutzten Gliederung. Im Verlauf wird das dort verwendete Beispiel zu einem XSD-konformen Schema weiterentwickelt.

Schemareferenz:
Jedes XML-Schema bildet als XML-Dokument eine eigenständige Speichereinheit, üblicherweise eine Datei. Mithin ist die Einbettung in das Instanzdokument, also die Bildung eines internal subset, nicht möglich.
Die Verbindung zwischen Schema und beschriebenem Dokument wird durch das in der XSD-Spezifikation vordefinierte Attribut `schemaLocation` bzw. `noNamespaceSchemaLocation` definiert. Eines dieser Attribute muß zwingend im Wurzelelement des XML-Dokuments angegeben werden.
Legt das Schema keinen Namensraum für die enthaltenen Deklarationen fest, d.h. alle darin deklarierten Elemente befinden sich im Vorgabenamensraum, so findet sich die Schemareferenz in `noNamespaceSchemaLocation`; andernfalls in `schemaLocation`.
Das nachfolgende Beispiel zeigt die Deklaration:

Beispiel 28: Definition einer Schemareferenz

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<ProjektVerwaltung
(3)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)     xsi:schemaLocation="http://www.jeckle.de/vorlesung/xml/examples/projektverwaltung.xsd">
(5)     ...
```

Im Beispiel wird zunächst der XML-Schema-Instanzen-Namensraum an das Präfix `xsi` gebunden. Dies ermöglicht die Einbindung von Elementen und Attributen aus der Schemaspezifikation in das eigene Dokument.
Als erste Nutzung eines solchen Elements aus XSD wird das Attribut `schemaLocation` im Wurzelelement mit der URI des Schemas als Wert belegt. Die Deklaration des XSI-Namensraumes ist daher zwingend. Die angegebene URI kann zur Ermittlung des Schemas für Validierungszwecke durch einen XML-Prozessor genutzt werden.

Durch die Einführung von XSD als weiterer Grammatiksprache verliert der Begriff der Gültigkeit an Qualität. So sind Dokumente denkbar, die zwar hinsichtlich einer gegebenen DTD als *valid* eingestuft werden, jedoch ein zugehöriges Schema verletzen.
Daher findet sich in der XSD-Spezifikation der Begriff der *schema validness* in Erweiterung der bisherigen (DTD-bezogenen) validness.

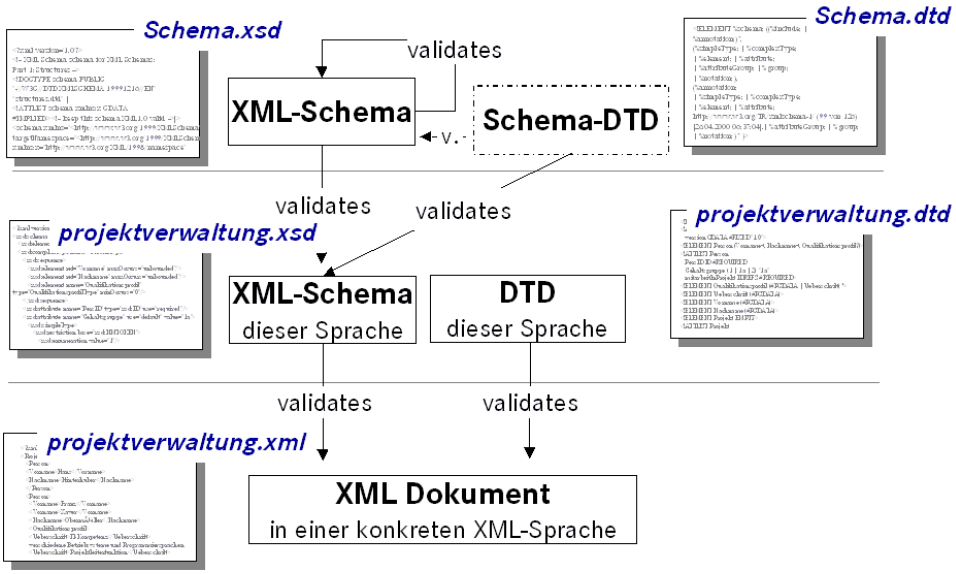
Definition 11: Gültigkeit hinsichtlich eines Schemas

Ein XML-Dokument heißt *gültig hinsichtlich eines Schemas* (*schema valid*), wenn es über ein Schema verfügt, und konform zu diesem aufgebaut ist.

Der hier gegebene Gültigkeitsbegriff stützt sich explizit nicht auf der bisherigen validness ab, da er die Konformität hinsichtlich einer eventuell existierenden DTD außer Acht läßt.

So etabliert XML-Schema einen vom DTD-gebundenen Gültigkeitsbegriff losgelösten Validierungsmechanismus. Aufgrund der Realisierung der Schemasprache als XML-Sprache ist jedes Schema auch ein XML-Dokument. Daher eröffnet sich die Möglichkeit, das Schema selbst durch ein Schema zu beschreiben. Dieses *Schema für Schema* -- auch *Metaschema* genannte -- XML-Dokument erlaubt die Validierung (im Sinne der *schema validness*) jedes Schemas. Damit erfüllt sich eine der Anforderungen an den Schemamechanismus: die Validierbarkeit der erstellten Schemata selbst, was für DTDs nicht gegeben war. In der praktischen Anwendung zeigt sich dies in der Möglichkeit, erstellte Schemata mit denselben Werkzeugen zu analysieren, verarbeiten und zu prüfen, die auch für Instanzdokumente verwendet werden.
Da das Metaschema selbst wiederum ein XML-Dokument ist, folgt, daß hierfür auch ein Schema angegeben werden kann. Die XML-Standardisierung hat hier -- nicht zuletzt um eine unendliche Reihung zur Validierung notwendiger

Schemata zu vermeiden -- den Ansatz gewählt, das Schema für Schema durch sich selbst zu beschreiben.
Um den schrittweisen Übergang zu XSD zu befördern, hat die XSD-Arbeitsgruppe eine DTD für XSD herausgegeben. Mittels dieser können neu erstellte Schemata auch mit „klassischen" Werkzeugen auf (DTD-)Gültigkeitkeit geprüft werden.
Die Abbildung stellt die getroffenen Aussagen und Validierungsbeziehungen nochmals graphisch zusammen.



Die Schema-Definition:
Wuzelknoten jedes XSD-Dokuments ist das Element Information Item schema. Alle Definitionen eines Schemas sind direkte Kindknoten dieses Elements oder dessen Kindknoten.
Durch die Attribute des schema-Elements werden verschiedene Eigenschaften festgelegt, die für alle im Schema definierten Elemente und Attribute gelten.

Zunächst wird durch eine Reihe von Attributen das Verhalten des Schemas in Bezug auf Namensräume festgelegt. Als Besonderheit eines XML-Schemas fällt hier die ständige Berücksichtigung von mindestens zwei Namensräumen ins Auge. Während ein Schema mit Elementen des Schemanamensraumes aufgebaut wird, trifft es zeitgleich Aussagen über einen zweiten Namensraum -- den Namensraum des Vokabulars für das das Schema erstellt wird. Dieser Namensraum wird *Zielnamensraum* (*target namespace*) genannt.
Daher findet sich im Attribut `targetNamespace` die URI des Zielnamensraumes. In diesen Namensraum werden automatisch alle durch das Schema deklarierten Elemente und Attribute übernommen. Als Konsequenz müssen diese in jedem Schema-gültigen XML-Dokument im entsprechenden Namensraum auftreten. Hierbei wird nicht zwischen expliziter Namensraumdeklaration durch ein gebundenes Präfix und impliziter Deklaration durch Überschreiben des Vorgabenamensraumes unterschieden.
Durch Angabe der Attribute `elementFormDefault` und `attributeFormDefault` kann der durch `targetNamespace` implizierte Namensraumzwang für das XML-Instanzdokument gelockert werden. Wird der Wert der beiden Attribute auf `unqualified` gesetzt, so können die Attribute auch außerhalb des Zielnamensraumes auftreten. Dies entspricht auch dem Vorgabeverhalten.

Definition von Elementen:
Als Obermenge der Ausdrucksmächtigkeit der DTD unterstützt auch XSD die Inhaltsmodelle

- unstrukturierter Inhalt
- beliebig (jedes Element kann als Kindelement angegeben werden)
- leer (keine Kindelemente)
- explizit angegebene Kindelemente
- gemischter Inhalt (gleichzeitiges Auftreten von Elementen und unstrukturiertem Text)

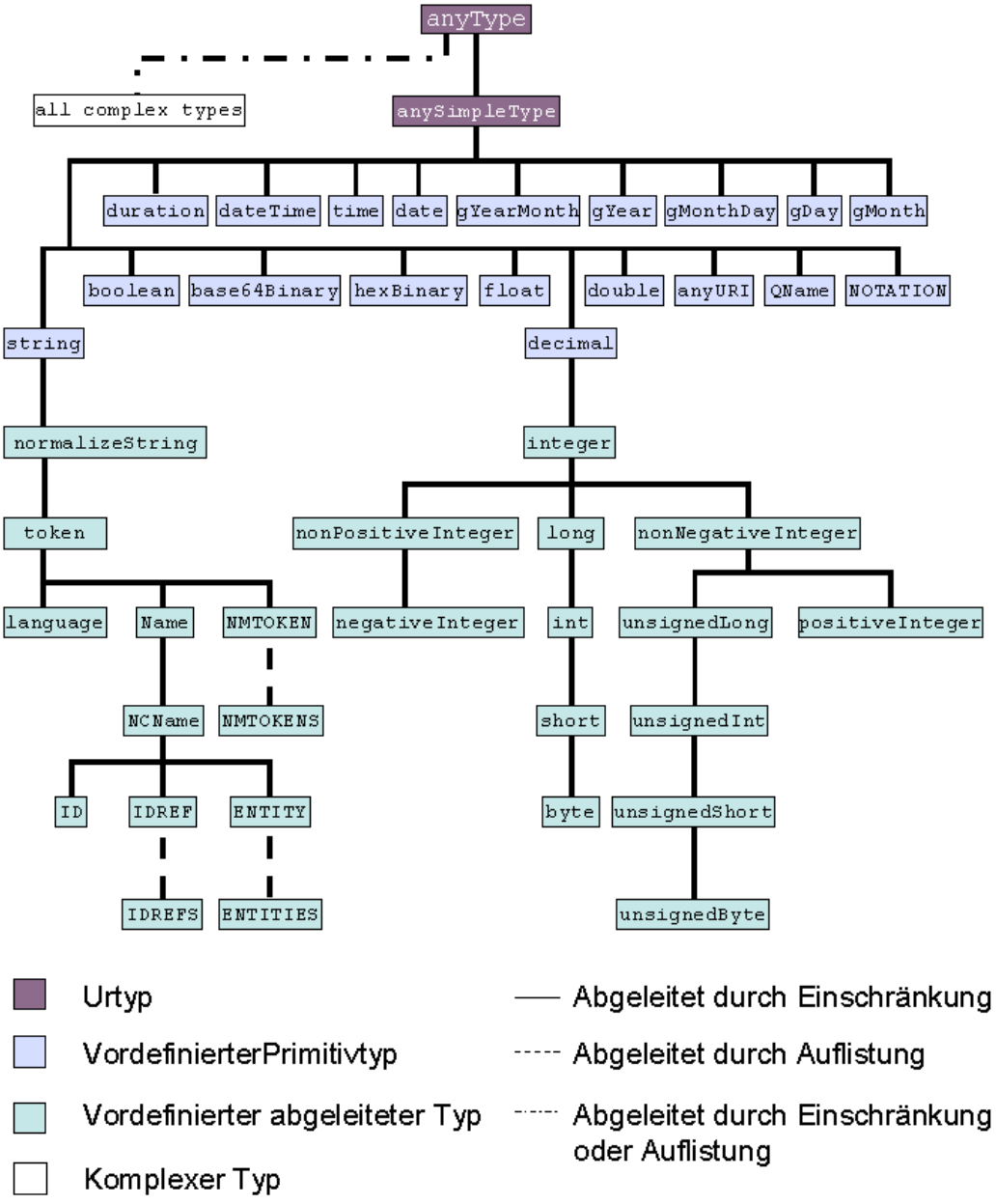
Generell wird jedes Element durch das XSD-Element `element` ausgedrückt.

Für unstrukturierten Inhalt bot die DTD lediglich Zeichenketten-artige Inhalte des Typs `#PCDATA` .
XML-Schema Part 2 definiert insgesamt 44 Primitivtypen. Darunter finden sich die aus der DTD bekannten Element- und Attributtypen, sowie eine Fülle Neuer.

Im Kern zerfallen die XSD-Typen in drei Typklassen:

- Atomare und aggregierte Typen
  Atomare Typen bestehen aus unteilbaren Werten. Der Begriff der *Unteilbarkeit* soll dabei auf die nicht erkennbare Unterstrukturierung (`string` besteht zwar aus einzelnen Zeichen) bzw. falls erkennbar, dem nicht explizit unterstützten Zugriff auf die Komponenten (`date` bietet keine Zugriffsmöglichkeit auf die Komponenten Tag, Monat und Jahr), verweisen.
  Die aggregierten Typen teilen sich in Listen-artige und Vereinigungstypen (*Union*). Listen bestehen dabei aus einer geordneten Menge atomarer Typen.
  Vereinigungstypen erlauben hingegen die Verknüpfung Typ-verschiedener Datentypen zu einem neuen.
- Primitive und abgeleitete Typen
  Primitive Datentypen existieren unabhängig von anderen Datentypen, während abgeleitete Datentypen von der Definition ihres Elterntyps abhängig sind.
- Vorgegebene und anwenderdefinierte Typen
  Die vorgegebenen Typen sind diejenigen, die in XML-Schema Part2 beschrieben sind. Alle weiteren Typen eines Schemas sind durch den Anwender definierte abgeleitete Typen.

Durch Erweiterungs- und Aggregationsmechanismen ergibt sich das in der nachfolgenden Abbildung dargestellte Typsystem.



Die Tabelle stellt die angebotenen Typen mit einigen Beispielen dar:

Tabelle 5: Typen in XSD-Schema Part 2

Typname	Beispiel	Bemerkung
string	Hello
World	Jedes beliebige Unicode Symbol gemäß XML-Syntaxproduktion 2
normalizedString	&#20Hello&#20;World	Jedes beliebige Unicode Symbol außer Zeilenvorschub, Wagenrücklauf und Tabulatoren normalizedString ist eine einschränkende Spezialisierung des Typs string
token	Hello World	Jeder normalizedString, unter Weglassung führender, abschließender und mehrfacher Leerzeichen (#x20), sowie Zeilenvorschüben (#xA) und Tabulatoren (#x9). token ist eine einschränkende Spezialisierung des Type normalizedString
Name	aName, _helloWorld, :notAGoodIdea	XML Name gemäß Syntaxproduktion 5. Name ist eine einschränkende Spezialisierung des Typs token
QName	xsd:element, element	Durch Namensraumpräfix qualifizierter Name gemäß Produktion 6 der XML Namespace Recommendation

NCName	aName, _anotherName, X	Name, der keinen Doppelpunkt enthält (*non colonized name*), gemäß Produktion 4 der XML Namespace Recommendation
decimal	-1.23, 12678967.543233, +100000.00, 210	Wertebereich: i*10^{-n}, mit i, n aus integer, n>=0 Ein Prozessor muß mindestens 18 Dezimalstellen unterstützen
long	-9223372036854775808, ... -1, 0, 1, ... 9223372036854775807	Wertebereich: 2^{63} <= long <= 2^{63}-1 long ist eine einschränkende Spezialisierung des Typs integer
int	-2147483648, ... -1, 0, 1, ... 2147483647	Wertebereich: -2^{31} <= int <= 2^{31}-1 int ist eine einschränkende Spezialisierung des Typs long
short	-32768, ... -1, 0, 1, ... 32767	Wertebereich: -2^{15} <= short <= 2^{15}-1 short ist eine einschränkende Spezialisierung des Typs int
byte	-128, ...-1, 0, 1, ... 127	Wertebereich: -2^{7} <= byte <= 2^{7}-1 byte ist eine einschränkende Spezialisierung des Typs short
integer	...-1, 0, 1, ...	Wertebereich: entspricht der mathematischen Menge der ganzen Zahlen (Z) integer ist eine einschränkende Spezialisierung des Typs decimal
positiveInteger	1, 2, ...	Wertebereich: entspricht der mathematischen Menge der natürlichen Zahlen (N) positiveInteger ist eine einschränkende Spezialisierung des Typs nonNegativeInteger
negativeInteger	... -2, -1	Wertebereich: {..., -2, -1}, die unendliche Menge der negativen Zahlen negativeInteger ist eine einschränkende Spezialisierung des Typs nonPositiveInteger
nonNegativeInteger	0, 1, 2, ...	Wertebereich: 0 <= nonNegativeInteger nonNegativeInteger ist eine einschränkende Spezialisierung des Typs integer
nonPositiveInteger	... -2, -1, 0	Wertebereich: {..., -2, -1, 0} die unendliche Menge der negativen Zahlen, und die Null nonPositiveInteger ist eine einschränkende Spezialisierung des Typs integer
unsignedLong	0, 1, ... 18446744073709551615	Wertebereich: 0 <= unsignedLong <= 2^{64}-1 unsignedLong ist eine einschränkende Spezialisierung des Typs nonNegativeInteger
unsignedInt	0, 1, ...4294967295	Wertebereich: 0 <= unsignedInt <= 2^{32}-1 unsignedInt ist eine einschränkende Spezialisierung des Typs unsignedLong
unsignedShort	0, 1, ... 65535	Wertebereich: 0 <= unsignedShort <= 2^{16}-1 unsignedShort ist eine einschränkende Spezialisierung des Typs unsignedInt
unsignedByte	0, 1, ... 255	Wertebereich: 0 <= unsignedByte <= 2^{8}-1 unsignedByte ist eine einschränkende Spezialisierung des Typs unsignedShort
float	-1E4, 1267.43233E12, 12.78e-2, 12, INF	32-Bit-Zahl mit einfacher Genauigkeit gemäß IEEE 754-1985. Wertebereich: $m * 2^{e}$, wobei m und $e$integer-Elemente mit m <= 2^{24}, und -149 <= e < 104 sind.
double	-1E4, 1267.43233E12, 12.78e-2, 12, INF	64-Bit-Zahl mit doppelter Genauigkeit gemäß IEEE 754-1985. Wertebereich: $m * 2^{e}$, wobei m und $e$integer-Elemente mit m <= 2^{53}, und -1075 <= e < 970 sind.
boolean	true, false, 1, 0	Unterstützung der klassischen zweiwertigen Logik

time	13:20:00-05:00, 13:20:00.000	Uhrzeit, die täglich wiederkehrt, ausgedrückt im Format gemäß ISO 8601
date	2004-05-24	Datumsformat: CCYY-MM-DD, gemäß ISO 8601
gYear	1999, 2001, 2004	Darstellung von Jahren des gregorianischen Kalenders gemäß ISO 8601
gYearMonth	2004-05	Darstellung eines Monats eines bestimmten Jahres des gregorianischen Kalenders gemäß ISO 8601
gDay	----05, ----31	Darstellung eines wiederkehrenden Tages eines Monats gemäß ISO 8601
gMonthDay	--31-12, --01-01	Darstellung eines wiederkehrenden gregorianischen Datums, gebildet aus Tag Monat und Monat im Format --MM-DD, gemäß ISO 8601
gMonth	--03, --12	Monatsformat: --MM-- gemäß ISO 8601
dateTime	2004-05-24T13:34:31.000+02:00	Zeitpunkt, ausgedrückt durch Datum und Uhrzeit; beide gemäß ISO 8601 codiert.
duration	P1Y2M3DT10H30M12.3S Zeitraum von einem Jahr, zwei Monaten, drei Tagen, zehn Stunden, 30 Minuten und 12,3 Sekunden	Nach Größe (Signifikanz) geordnete Koordinate im sechs-dimensionalen Raum aus Jahr, Monat, Tag, Stunde, Minute und Sekunde. Formatdefinition laut ISO 8601
base64Binary	SGVsbG8gd29ybGQhCg==	Base64-Darstellung eines beliebigen Binär-interpretierten Inhaltes gemäß IETF RFC 2045
hexBinary	0FB7	Hexadezimale Darstellung beliebiger Binär-interpretierter Inhalte
anyURI	http://www.jeckle.de	Jede gemäß IETF RFC 2396 bzw. IETF RFC 2732 gültige URI
language	en-GB, en, de-de	Sprachcodierung gemäß IETF RFC 1766 und XML Recommendation language identification. Die Identifikationsnamen werden durch ISO 639 sowie ISO 3166 definiert. language ist eine einschränkende Spezialisierung des Typs token
ID	test, XYZ	XSD-Darstellung des DTD-Typen ID. Zugelassen sind alle Ausprägungen der Namespaceproduktion 4 (NCName). ID ist eine einschränkende Spezialisierung des Typs NCName
IDREF	test, XYZ	XSD-Darstellung des DTD-Typen IDREF. Zugelassen sind alle Ausprägungen der Namespaceproduktion 4 (NCName). IDREF ist eine einschränkende Spezialisierung des Typs NCName
IDREFS	test1 test2 test4, test3 test5	XSD-Darstellung des DTD-Typen IDREFS. Zugelassen sind Listen aus white space separierten Ausprägungen der Namespaceproduktion 4 (NCName). IDREFS ist eine nichtleere Aufzählung von IDREF-Ausprägungen
ENTITY		XSD-Darstellung des DTD-Typen ENTITY. Zugelassen sind alle Satzformen, die der Produktion NCName der XML-Namensräume entsprechen und als ungeparste Entität definiert sind. ENTITY ist eine einschränkende Spezialisierung des Typs NCName
ENTITIES		XSD-Darstellung des DTD-Typen ENTITIES. Zugelassen sind Listen aus white space separierten Ausprägungen des Typs ENTITY. ENTITIES ist eine nichtleere Aufzählung von ENTITY-Ausprägungen

NOTATION		XSD-Darstellung des DTD-Typen NOTATION. Zur Verwendung dieses Typs in einem Schema muß eine Ableitung von NOTATION durch den Anwender definiert werden.
NMTOKEN	US, Deutschland	XSD-Darstellung des DTD-Typen NMTOKEN. Ausprägungen dieses Typs müssen konform zur Produktion 7 der XML-Spezifikation sein. NMTOKEN ist eine einschränkende Spezialisierung des Typs token
NMTOKENS	US UK Aus, Ger	XSD-Darstellung des DTD-Typen NMTOKENS. Zugelassen sind Listen aus white space separierten Ausprägungen des Typs NMTOKEN. NMTOKENS ist eine nichtleere Aufzählung von NMTOKEN-Ausprägungen
anyType	1, 2.3, aGVsb, 06b8f45, test&#20;für&#20; anyType&#0A; <sentence>the quick brown <animal>fox</animal>...</sentence>	Allgemeinster Datentyp. Konzeptionell bildet er die Vereinigung aller angebotenen XSD-Typen.

Die einfachste Form zur **Definition eines Elements mit unstrukturiertem typisierten Inhalt** lautet:

```
<xsd:element
                    name="elementName"
                    type="typeName"/>
```

In dieser Darstellung entspricht die Definition -- von der Typisierung abgesehen -- der Mächtigkeit der ELEMENT-Deklaration einer DTD mit PCDATA-artigem Inhalt.
XSD definiert ferner folgende Charakteristika für Elemente, die durch Attribute der Elementdeklaration ausgedrückt werden:

- **abstract**: falls auf `true` gesetzt, darf ein solches Element nicht in einem XML-Dokument auftreten. Es kann ausschließlich zur Strukturierung des Schemaentwurfs eingesetzt werden und als Basis von Spezialisierungen dienen. Vorgabewert ist `false`
- **block**: erlaubt die Kontrolle der Verwendung abgeleiteter Typen. Zugelassene Belegungen beliebige Kombinationen aus `extension`, `restriction` und `substitution` oder der Einzelwert `#all`.
  Ist das Attribut auf `restriction` gesetzt, so dürfen keine (einschränkend) abgeleiteten Typen Ausprägungen des Originaltyps in Instanzdokumenten ersetzen. Dasselbe gilt für `extension` oder als Substitutionsgruppen deklarierte Typen (`substitution`-Belegung).
  Der Wert `#all` versammelt alle möglichen Varianten und verbietet generell die Ersetzung eines Elements durch andere.
- **default**: Vorgabebelegung des Inhalts durch eine beliebige Zeichenkette, die konform zum gewählten Typ ist. Dieser Wert wird durch den XML-Prozessor an die Applikation gemeldet, wenn kein Wert im Dokument angegeben wird.
- **final**: verhindert die Ableitung von Typen. Die zulässigen Belegungen sind mit denen für `block` identisch, nur daß durch dieses Attribut die Vererbungsmechanismen bereits auf Schemaebene verboten werden, während `block` ihre Nutzung im Instanzdokument einschränkt.
- **fixed**: erlaubt die konstante Wertbelegung.
- **form**: legt fest, ob das Element im Instanzdokument mit Namensraumpräfix erscheint.
  Zulässige Belegungen: `qualified` (Namensraumpräfix muß angegeben werden) und `unqualified`.
- **id**: erlaubt die eineindeutige Kennzeichnung eines Elements durch eine Schema-weit eindeutige Zeichenkette.
- **minOccurs**: Minimalkardinalität, d.h. Mindestzahl zulässiger Vorkommen dieses Elements. Der Attributinhalt ist ein Element aus nonNegativeInteger.
  Das Attribut ist optional, und wird bei fehlender Angabe mit dem Vorgabewert `1` belegt.
- **maxOccurs**: Maximalkardinalität, d.h. Höchstzahl zulässiger Vorkommen dieses Elements. Der Attributinhalt ist entweder ein Element aus nonNegativeInteger oder die Zeichenkette unbounded zur Kennzeichnung beliebig vieler Auftreten.
  Das Attribut ist optional, und wird bei fehlender Angabe mit dem Vorgabewert `1` belegt.
- **name**: Unqualifizierter Name des Elements, konform zur NCName-Produktion der Namensraum-Spezifikation.
- **nillable**: Erlaubt Null-Werte im Instanzdokument, die Semantik ist dabei an die in relationalen Datenbanksystemen verwirklichte angelehnt.
  Die Belegung ist entweder `true` oder `false`, was auch als Vorgabe bei Fehlen dieses Attributs angenommen wird.
- **ref**: Referenz auf eine andere Elementdeklaration zur Übernahme der dort spezifizierten Definitionen.
- **substitutionGroup**: Name einer Gruppe von Elementen, die anstatt des aktuellen Elements im Instanzdokument auftreten dürfen.
- **type**: Ein durch Schema Part 2 vordefinierter Typ, oder jeder beliebige anwenderdefinierte.

Nachfolgend sind einige Elementdeklarationen für unstrukturierten Inhalt versammelt

Beispiel 29:

```
(1)<element name="geburtsdatum" type="xsd:date"/
>
(2)<element name="pi"
(3)     type="xsd:double"
(4)     fixed="3.141592653"
(5)     block="#all"
(6)     final="#all"/>
(7)<element name="vorname"
(8)     type="xsd:token"
(9)     minOccurs="1"
(10)    maxOccurs="unbounded"/>
(11)<element name="artikelNummer"
(12)    type="xsd:NCName"
(13)    form="qualified"/>
```

Die Deklaration `geburtsdatum` definiert ein XML-Element des Typs date zur Darstellung eines Datums. Weitere Festlegungen sind nicht getroffen, daher wird das Element mit `minOccurs` und `maxOccurs` 1 belegt, wodurch es als zwingend anzugebend (*mandatory*) und skalar (d.h. nicht mengenwertig) ausgewiesen wird.
`pi` legt die gleichnamige mathematische Konstante fest. Als Datentyp wurde double, eine Gleitkommazahl mit doppelter Genauigkeit gewählt. Als konstante Belegung wird durch das `fixed` Attribut der entsprechende Zahlenwert festgelegt. Daher muß eine Vorgabebelegung durch das Attribut `default` nicht erfolgen; gemäß Schema-Spezifikation darf sie sogar nicht erfolgen, `fixed` und `default` schließen sich gegenseitig aus. Um eine weitere Spezialisierung des Elements durch Vererbung oder Aggregation zu verhindern wird der Wert von `block` auf `#all` gesetzt, wodurch die Teilnahme an allen Typbildungsmechanismen unterbunden wird.
Die Definition für `vorname` nutzt als Datentyp den token, der automatisch mehrfache, führende und abschließende Leerzeichen sowie sonstige Formatierungssymbole entfernt. Ferner kann dieses Element beliebig häufig auftreten -- `maxOccurs` ist daher auf unbounded gesetzt. Die Fixierung der minimalen Auftrittshäufigkeit auf 1 (`minOccurs`) entspricht der Vorgabebelegung.
Für das Element `artikelNummer` ist als Typ `NCName` ausgewählt, was beliebigen Zeichenketten -- die keinen Doppelpunkt enthalten -- entspricht. Darüberhinaus ist das Attribut `form` mit dem Wert `qualified` versehen. Dies führt dazu, daß das Namensraumkürzel für dieses Element zwingend im Instanzdokument anzugeben ist.

Zur Umsetzung des freien Inhaltsmodells, das beliebige Inhalte aus den definierten Elementen und freien Texten zuläßt, wird ebenfalls auf das Typsystem zurückgegriffen.
Wird das `type` Attribut nicht belegt, so wird gemäß Vorgabe der Typ anyType angenommen. Elemente dieses Typs können beliebige wohlgeformte Inhalte beherbergen.
Die beiden nachfolgenden Angaben sind daher äquivalent.

```
<element
   name="elementName"
   type="xsd:anyType"/>
<element name="elementName"/>
```

XSD prägt den bereits im Kontext der DTD genutzten Typbegriff strenger. Dies zeigt sich deutlich in der Existenz des XSD-Elements `complexType`. Es führt die Möglichkeit einer expliziten, d.h. von der Verwendung losgelösten Typbildung, ein. Syntaktisch kann die complexType-Definition sowohl innerhalb einer Elementdefinition, als auch separat erfolgen.
Den einfachsten Anwendungsfall bildet die eingebettete leere complexType-Definition zur Darstellung des **leeren Inhaltsmodells**.
Die Syntax hierfür lautet (der XSD-Namensraum sei an das Präfix `xsd` gebunden):

```
<xsd:element
   name="elementName">
   <xsd:complexType/>
</xsd:element>
```

Ein XML-Schema-validierender Parser verhält sich in diesem Falle identisch zu einem (DTD-)validierenden Parser für das Inhaltsmodell `EMPTY`. Daher werden für die obige Festlegung ausschließlich die beiden Darstellungsformen zur Angabe eines leeren Elements (`<elementName/>` bzw. `<elementName></elementName>`) akzeptiert.

Die Befüllung des `complexType`-Elements leitet direkt zum wichtigsten Inhaltsmodell über, dem explizit angegebener Kindelemente.
Während die DTD das sequentielle Auftreten der Kindelemente in der angegebenen Reihenfolge nur implizit zu Grunde legt, stellt XML-Schema diesen Sachverhalt durch ein eigenes Sprachkonstrukt klar dar. Hierfür werden alle Kindelemente in ein weiteres Element sequence eingebettet.
Das Auswahlinhaltsmodell (auch: Selektionsmodell) -- in der DTD durch Oder-Verknüpfung der einzelnen Elemente einer Elementgruppe ausgedrückt (`...|...|...`) -- wird entsprechend durch das XSD-Element `choice` ausgedrückt.
Eine besondere Variante des Selektionsmodells stellt die `all`-Gruppe dar. Sie kürzt das häufig genutzte Inhaltsmodell `(...|...|...)*` ab. Es erlaubt die Angabe der Kindelemente in beliebiger Reihenfolge.
In Entsprechung zur Klammerdarstellung in der DTD muß zwingend einer der drei Reihenfolgetypen sequence, choice oder `all` als Element spezifiziert werden.
Analog der Klammer-Schachtelung in der DTD, können auch die verschiedenen Modellgruppen ineinander kombiniert werden.
Am Beispiel der Elementdefinitionen der Projektverwaltung:

Beispiel 30: Einige Elementdefinitionen

```
(1)<?xml version = "1.0" encoding = "UTF-8"?>
(2)<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
(3)     <xsd:element name = "ProjektVerwaltung">
(4)             <xsd:complexType>
(5)                     <xsd:sequence>
(6)                             <xsd:element ref = "Person" maxOccurs = "unbounded"/>
(7)                             <xsd:element ref = "Projekt" maxOccurs = "unbounded"/>
(8)                     </xsd:sequence>
(9)             </xsd:complexType>
(10)    </xsd:element>
(11)    <xsd:element name = "Person">
(12)            <xsd:complexType>
(13)                    <xsd:sequence>
(14)                            <xsd:element name = "Vorname" type = "xsd:token" maxOccurs =
"unbounded"/>
(15)                            <xsd:element name = "Nachname" type = "xsd:token"/>
(16)                            <xsd:element ref = "Qualifikationsprofil" minOccurs = "0"/>
(17)                    </xsd:sequence>
(18)            </xsd:complexType>
(19)    </xsd:element>
(20)    <xsd:element name = "Projekt">
(21)            <xsd:complexType/>
(22)    </xsd:element>
(23)    <xsd:element name = "Qualifikationsprofil">
(24)            <xsd:complexType mixed = "true">
(25)                    <xsd:sequence>
(26)                            <xsd:element name = "Qualifikation" type = "xsd:string" minOccurs =
"0" maxOccurs = "unbounded"/>
(27)                            <xsd:element name = "Leistungsstufe" type = "xsd:string" minOccurs
= "0" maxOccurs = "unbounded"/>
(28)                    </xsd:sequence>
(29)            </xsd:complexType>
(30)    </xsd:element>
(31)</xsd:schema>
```

Download des Beispiels

Das Schema enthält alle Elementdefinitionen für die Projektverwaltung. Innerhalb jedes `element`-Elements sind die entsprechenden Kindelemente in sequence-Strukturen eingebettet. Analog dem entsprechenden DTD-Mechanismus müssen sie daher in der Reihenfolge ihres Auftretens im Schema auch im Instanzdokument wiedergegeben werden.
Von besonderem Interesse ist die Definition des *Qualifikationsprofil*s. Es handelt sich dabei um ein **mixed content model**, ausgedrückt durch das Boole'sche Attribut `mixed` (in Spezifikation nachschlagen). Die dargestellte Definition läßt an dieser Stelle einen weiteren Vorteil der Schemasprache gegenüber der Dokument-Typ-Definition offensichtlich werden. Während gemischte Inhalte aus Auszeichnungssymbolen und freiem Text durch DTD-validierende Parser nur rudimentär geprüft werden konnten, ermöglicht XSD die vollständige inhaltliche Validierung gemischter Inhalte. Die lediglich partielle Strukturvalidierung der DTD lag in deren syntaktischer Konstruktion zur Darstellung gemischter Inhalte begründet. Diese erlaubt zwar die Spezifikation von innerhalb unstrukturierter Textpassagen (möglicherweise) auftretenden Elementen, nicht jedoch die Überwachung der Auftrittshäufigkeit oder Reihenfolge. So wäre gemäß der Beispiel-DTD auch die Hintereinanderreihung von `Qualifikationen` ohne zugehörige `Leistungsstufe` zulässig. Insgesamt kann durch DTD-basierte Validierung das Auftreten von Elementen in gemischten Inhaltsmodellen nicht geprüft werden. Durch den Einsatz von XML-Schema ergibt sich auch für *mixed content models* die Möglichkeit zur strikten Validierung. Dies bedeutet konkret die Prüfbarkeit der Anzahl und Auftretensreihenfolge der angegebenen Kindelemente (in Spezifikation nachschlagen).
Darüberhinaus enthält das Beispiel neben *lokalen Elementdeklarationen*, die sich vollständig im Elternelement finden (wie `Vorname`, `Nachname` und `Qualifikation`), auch *globale Elementdeklarationen*, die zunächst deklariert und in einem zweiten Schritt durch Referenzierung als Kindelemente verwendet werden (wie `Person` und `Projekt` innerhalb `Projektverwaltung`, oder `Qualifikationsprofil` innerhalb des Elements `Person`). Hierdurch können vollständige Elemente an verschiedenen Stellen im Schema referenziert und so verwendet werden. Die Definition ist der lokalen ebenbürtig und wird im Instanzdokument identisch behandelt. Zusammenfassend läßt sich festhalten: Mit dem Referenzierungsmechanismus für Elemente kann eine einfache Form der Wiederverwendung umgesetzt werden.
Den Zeichenketten-artigen Elementtypen wurde durchgehend der XSD-Typ `string` zugewiesen.

Durch die Referenzierungsmöglichkeit existiert eine erste Möglichkeit zur Wiederverwendung bereits im Schema definierter Elemente. Jedoch werden Elemente hierbei zwingend in ihrer vollständigen Definition, d.h. Name, Typ und Inhaltsmodell, eingebunden. Im Grunde genommen ist diese Art der Wiederverwendung bereits mit den Mitteln der DTD möglich. Allerdings, wie im eben betrachteten Beispiel, auch dergestalt, daß nur die vollständige Definition übernommen werden kann.
XML-Schema bietet zusätzlich die Möglichkeit, strukturierte Typen, die ausschließlich durch ihr Inhaltsmodell definiert werden, festzulegen. In der Konsequenz verändert sich der durch die DTD formulierte Typbegriff hin zu einer eher an den Programmiersprachen orientierten Sichtweise, da die Benennung des Typs von der Namensgebung der typisierten Instanz separiert wird.
Syntaktisch erfolgt die Typbildung durch die Benennung des `complexType`-Elements durch ein Attribut `name`. Um die mehrfache Verwendung eines solchen Typen zu ermöglichen, muß seine Definition zwingend auf einer Baumstufe erfolgen, die für alle nutzenden Elemente erreichbar ist. Üblicherweise werden daher diese Definitionen auf der ersten Stufe, direkt unterhalb des Wurzelknotens, plaziert.
Zur Unterscheidung dieser *benannten komplexen Typen* werden die bisher genutzten -- namenlosen Typen -- als *anonyme komplexe Typen* bezeichnet.
Das nachfolgende Beispiel zeigt die Definition eines benannten komplexen Typen am Beispiel des Elements `Person`:

Beispiel 31: Nutzung benannter komplexer Typen

```
(1)<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
(2)   <xsd:complexType name="PersonType">
(3)        <xsd:sequence>
(4)        <xsd:element name = "Vorname" type = "xsd:string"
(5)                        maxOccurs = "unbounded"/>
(6)        <xsd:element name = "Nachname" type = "xsd:string"/>
(7)        <xsd:element ref = "Qualifikationsprofil" minOccurs = "0"/>
(8)   </xsd:sequence>
(9)   </xsd:complexType>
(10)
(11)   <xsd:element name = "ProjektVerwaltung">
(12)      <xsd:complexType>
(13)         <xsd:sequence>
(14)            <xsd:element name="Person" type="PersonType" maxOccurs = "unbounded"/>
(15)            <xsd:element ref = "Projekt" maxOccurs = "unbounded"/>
(16)         </xsd:sequence>
(17)      </xsd:complexType>
(18)   </xsd:element>
(19)
(20)   <xsd:element name = "Projekt">
(21)      <xsd:complexType/>
(22)   </xsd:element>
(23)
(24)   <xsd:element name = "Qualifikationsprofil">
(25)      <xsd:complexType mixed = "true">
(26)         <xsd:sequence>
(27)            <xsd:element name = "Qualifikation" type = "xsd:string"
(28)                           minOccurs = "0" maxOccurs = "unbounded"/>
(29)            <xsd:element name = "Leistungsstufe" type = "xsd:string"
(30)                           minOccurs = "0" maxOccurs = "unbounded"/>
(31)         </xsd:sequence>
(32)      </xsd:complexType>
(33)   </xsd:element>
(34)</xsd:schema>
```

Download des Beispiels

Das Schema zeigt die Definition des komplexen Typen `PersonType`. Dieser Typ wird zur Festlegung des Inhaltsmodells des Elements `Person` verwendet.

Definition eigener Datentypen durch Vererbung:
Zur Unterstützung von Wiederverwendung und Erhöhung der Strukturierung des Entwurfs definiert XSD ein Vererbungskonstrukt zur Bildung neuer komplexer Typen auf der Basis bereits bestehender.
Zwei verschiedene Ableitungssemantiken werden angeboten:

- Ableitung durch Einschränkung (*derivation by restriction*)
  Der erbende Subtyp gibt eine engere Definition des Supertypen
- Ableitung durch Erweiterung (*derivation by extension*)
  Der erbende Subtyp erweitert die Definition des Supertypen

Das nachfolgende Beispiel zeigt die Anwendung der einschränkenden Ableitung.
Hierbei erbt der benannte komplexe Typ `childType` von `parentType`. Innerhalb des -- aus syntaktischen Gründen notwendigen -- Elements `complexContent` findet sich die Definition der Vererbung im Element `restriction`, das `base`-Attribut verweist auf den benannten Elterntypen.
Der Inhalt des `restriction`-Elements gleicht der Inhaltsmodelldefinition des komplexen Typen: Auch hier werden Elemente und ihre Auftrittsstruktur (im betrachteten Beispiel `sequence`) angegeben. Die Elementdefinition des Elements `elementA` in `childType` schränkt die gleichnamige Elementdefinition innerhalb des Elterntypen ein.
Nachvollziehbar wird diese Einschränkungsbeziehung zwischen `short` und `int` bei Betrachtung der Datentyphierarchie und der Typdefinition der verwendeten Primitivtypen. So bildet `short` per definitionem eine eingeschränkte Untermenge von `int` an. (Die entsprechende XSD-Definition findet sich im *Schema für Schema*).
Die beiden Elementdefinitionen `usage1` und `usage2` zeigen die Verwendung der anwenderdefinierten Typen.

Beispiel 32: Einschränkende Typableitung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(3)<xsd:complexType name="parentType">
(4)     <xsd:sequence>
(5)             <xsd:element name="elementA" type="xsd:int"/>
(6)     </xsd:sequence>
(7)</xsd:complexType>
(8)
(9)<xsd:complexType name="childType">
(10)<xsd:complexContent>
(11)    <xsd:restriction base="parentType">
(12)            <xsd:sequence>
(13)                    <xsd:element name="elementA" type="xsd:short"/>
(14)            </xsd:sequence>
(15)    </xsd:restriction>
(16)</xsd:complexContent>
```

```
(17)</xsd:complexType>
(18)
(19)<xsd:element name="usage1" type="parentType"/>
(20)<xsd:element name="usage2" type="childType"/>
(21)
(22)</xsd:schema>
```

Download des Beispiels

Durch das strukturierte Inhaltsmodell ergeben sich über die reine Typisierung hinausgehende Möglichkeiten zur Einschränkung der Inhalte. Die nachfolgende Tabelle stellt einige Varianten zusammen.

Tabelle 6: Beispiele für zulässige Restriktionen

Basistyp	Restriktion	Bemerkung
	default	Zusätzliche Belegung eines Elements mit einem Vorgabewert
	fixed	Beschränkung eines zunächst frei wählbaren Elements auf konstanten Inhalt
	type	Definition eines Typen für ein zunächst untypisiertes Element. (Auch hierbei handelt es sich um eine einschränkende Redefinition, da allen Elementen ohne Typdefinition standardmäßig der Typ anyType zugeordnet wird.)
minOccurs=n_1, maxOccurs=m_1	minOccurs=n_2, maxOccurs=m_2	Restriktion der Auftrittshäufigkeit auf eine geringere Anzahl. Daher gilt: $n_1 <= n_2$ und $m_1 >= m_2$

Die direkte Umkehrung der einschränkenden Spezialisierung bildet die *erweiternde Spezialisierung*. Sie greift nicht verändernd auf die Elemente des Supertyps zu, sondern definiert zusätzliche neue.
Untenstehendes XSD-Schema zeigt dies am Beispiel des Supertyps `parentElement`, der durch das abgeleitete Kindelement `childElement` erweitert wird. Hierzu definiert `childElement` ein zusätzliches `elementB`.

Beispiel 33: Erweiternde Typableitung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(3)     <xsd:complexType name="parentElement">
(4)             <xsd:sequence>
(5)                     <xsd:element name="elementA"/>
(6)             </xsd:sequence>
(7)     </xsd:complexType>
(8)
(9)     <xsd:complexType name="childElement">
(10)            <xsd:complexContent>
(11)                    <xsd:extension base="parentElement">
(12)                            <xsd:sequence>
(13)                                    <xsd:element name="elementB"/>
(14)                            </xsd:sequence>
(15)                    </xsd:extension>
(16)            </xsd:complexContent>
(17)    </xsd:complexType>
(18)</xsd:schema>
```

Download des Beispiels

Zusätzlich sieht XML Schema die Möglichkeit vor, komplexe Typen von simplen abzuleiten. Dies mag auf den ersten Blick ungewöhnlich erscheinen, eröffnet es doch scheinbar einen Weg, unstrukturierte Typen in strukturierte zu überführen.
Bei näherer Betrachtung offenbart sich jedoch, daß hier lediglich der Ableitungsbegriff überladen wurde, um einen einfachen Weg zur Verknüpfung der beiden Inhaltsmodelle strukturierter „XML-artiger“ Inhalt -- wie er durch `complexTypes` repräsentiert wird -- auf der einen, und unstrukturierter Inhalt -- wie er durch die einfachen Datentypen repräsentiert wird -- auf der anderen Seite, zu erhalten.

Beispiel 34: Ableitung eines komplexen Typen von einem Simplen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xs:schema
(3)     xmlns:xs="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)     <xs:element name="Vorname">
(7)             <xs:complexType>
(8)                     <xs:simpleContent>
(9)                             <xs:extension base="xs:string">
(10)                                    <xs:attribute
(11)                                            name="rufname"
(12)                                            type="xs:boolean"/>
(13)                            </xs:extension>
(14)                    </xs:simpleContent>
```

```
(15)            </xs:complexType>
(16)    </xs:element>
(17)</xs:schema>
```

Download des Beispiels

Durch die im Beispiel dargestellte Syntax wird es ermöglicht unstrukturiert-getypten Elementen Attribute zuzuordnen, obwohl diese eigentlich Bestandteil der Definition komplex-getyper Elemente sind.

So wird im Beispiel dem Element `Vorname` sowohl der simple Typ `string`, als auch durch den Ableitungsmechanismus das Attribut `rufname` -- im Rahmen eines `complexType`, zugeordnet.
Die Typisierung des Elements erfolgt hierbei nicht durch das `type`-Attribut innerhalb der Elementdeklaration, sondern innerhalb der `simpleContent`-Festlegung.

Neben der anwenderdefinierten Bildung komplexer Typen steht es dem XSD-Modellierer auch offen, eigene (primitive) Datentypen festzulegen oder eigene Typen von bestehenden abzuleiten.
Hierfür definiert XML-Schema Part1 das Element simpleType. Für einfache Typen ist jedoch nur die einschränkende Vererbung (*restriction*) zugelassen. Dies liegt in der praktischen Beherrschbarkeit des Typsystems begründet. Durch die strikte Restriktionssemantik ergibt sich die Möglichkeit kontravarianter Substitution, wie sie bei objektorientierten Typsystemen und Vererbungsstrukturen anzutreffen ist. Dies bedeutet, daß an jeder Stelle, an der eine Ausprägung eines Supertyps erwartet wird, auch -- unter Erhalt der Typrestriktion -- eine Ausprägung eines Subtypen auftreten darf. Beispielhaft: Wird an einer Stelle des Instanzdokumentes durch das Schema das Auftreten einer Ausprägung von `integer` verlangt, so kann der Anwender auch Ausprägungen der Subtypen `int`, `short` oder `byte` angeben ohne die Gültigkeit des XML-Dokuments zu beeinträchtigen.

Vereinigungstypen werden aus einer nichtleeren Menge von Ausgangstypen gebildet.
Das Beispiel zeigt die Definition eines Typen `termin`, der den vorgegebenen Primitivtypen `date` und eine Liste `NamenDerWochentage` (deren Definition nicht dargestellt ist) vereinigt. Insbesondere zeigt der Ausschnitt die Möglichkeit der Vereinigungsbildung auch über aggregierte Typen.

```
(1)<xs:simpleType name="termin">
(2)     <xs:union memberTypes="xs:date NamenDerWochentage"/>
(3)</xs:simpleType>
```

Das XSD-Beispiel zeigt, als Fragment der XML-Schemaspezifikation, die Definition des vorgegebenen Typs `short`, einer einschränkenden Spezialisierung des Typs `int`.
Am Beispiel gut nachvollziehbar sind die beiden Schritte zur Bildung eines eigenen Typen:

1. Auswahl eines Ausgangstypen (später Elementtyp (bei aggregierten Typen) oder Basistyp (bei abgeleiteten Typen))
2. Typdefinition durch Anwendung der entsprechenden Typkonstruktion und evtl. Einschränkung verschiedener Charakteristika

Im Beispiel wird der kleinste und größte gültige Wert (`minInclusive` bzw. `maxInclusive`) des neuen Typen `short` gegenüber dem Basistypen beschränkt.

Beispiel 35: Einschränkende Spezialisierung eines simplen Typen

```
(1)<xsd:simpleType name="short" id="short">
(2)     <xsd:restriction base="xsd:int">
(3)             <xsd:minInclusive value="-32768"
(4)                     id="short.minInclusive"/
>
(5)             <xsd:maxInclusive value="32767"
(6)                     id="short.maxInclusive"/>
(7)     </xsd:restriction>
(8)</xsd:simpleType>
```

Die Bildung aggregierter Typen folgt demselben Muster. Jedoch tritt an die Stelle der Ableitung die Spezifikation des Aggregationstyps (im Beispiel *Liste*) und Angabe des Inhaltstyps (im Beispiel `string`).

Beispiel 36: Bildung eines Aggregationstypen

```
(1)<xsd:simpleType name="WarenkorbElemente">
(2)     <xsd:list itemType="xsd:string"/>
(3)</xsd:simpleType>
```

Nachfolgend sind die verschiedenen Beschränkungsmöglichkeiten zusammengefaßt:

- `length`
  Längenbeschränkung bei atomaren Typen bzw. Beschränkung der Elementanzahl bei aggregierten Typen.
  Längenbeschränkung eines simplen Typs:

```
(1)<xs:simpleType name="Postleitzahl">
(2)     <xs:restriction base="xs:string">
(3)             <xs:length value="5"/>
(4)     </xs:restriction>
```

```
(5)</xs:simpleType>
```

Beschränkung der Elementanzahl einer Liste:

```
(1)<xs:simpleType name="VornamenList">
(2)     <xs:list itemType="xs:token"/>
(3)</xs:simpleType>
(4)<xs:simpleType name="VornamenRestrictedList">
(5)     <xs:restriction base="VornamenList">
(6)             <xs:length value="5"/>
(7)     </xs:restriction>
(8)</xs:simpleType>
```

- minLength
  Minimale Länge (bei atomaren Typen bzw. minimale Elementanzahl bei aggregierten Typen.
  Beispiel (der aggregierte Typ VornamenRestrictedList muß mindestens einen Eintrag enthalten):

```
(1)<xs:simpleType name="VornamenList">
(2)     <xs:list itemType="xs:token"/>
(3)</xs:simpleType>
(4)<xs:simpleType name="VornamenRestrictedList">
(5)     <xs:restriction base="VornamenList">
(6)             <xs:minLength value="1"/>
(7)     </xs:restriction>
(8)</xs:simpleType>
```

  Beispiel (der spezialisierte atomare Typ Titel muß aus mindestens fünf Zeichen bestehen):

```
(1)<xs:simpleType name="Titel">
(2)     <xs:restriction base="xs:string">
(3)             <xs:minLength value="5"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- maxLength
  Maximale Länge bei atomaren Typen bzw. maximale Elementzahl bei aggregierten Typen.
  Beispiel (der aggregierte Type VornamenRestrictedList muß mindestens einen, jedoch höchstens drei, Einträge enthalten):

```
(1)<xs:simpleType name="VornamenList">
(2)     <xs:list itemType="xs:token"/>
(3)</xs:simpleType>
(4)<xs:simpleType name="VornamenRestrictedList">
(5)     <xs:restriction base="VornamenList">
(6)             <xs:minLength value="1"/>
(7)             <xs:maxLength value="3"/>
(8)     </xs:restriction>
(9)</xs:simpleType>
```

  Beispiel (der spezialisierte atomare Typ Titel muß aus mindestens fünf, darf jedoch aus höchstens 80 Zeichen bestehen):

```
(1)<xs:simpleType name="Titel">
(2)     <xs:restriction base="xs:string">
(3)             <xs:minLength value="5"/>
(4)             <xs:maxLength value="80"/>
(5)     </xs:restriction>
(6)</xs:simpleType>
```

- pattern
  Erlaubt die Inhaltsdefinition durch Muster (reguläre Ausdrücke).
  XML-Strukturen sind nicht durch reguläre Ausdrücke darstellbar, hierfür können lediglich die vorgestellten Strukturierungsprimitive eingesetzt werden; daher sind Muster auf atomare Typen beschränkt.
  Die XML-Schemaspezifikation definiert einzelne Symbole und Symbolsequenzen, sowie einige abkürzende Schreibweisen zum Aufbau beliebiger Ausdrücke. Alle anderen Zeichen können direkt in die Musterspezifikation übernommen werden.
  Die aus der Verarbeitung regulärer Ausdrücke mit anderen Programmiersprachen oder Werkzeugen bekannten Quantifier stehen in der bekannten Semantik zur Verfügung.
  Sei S eine Zeichenkette aus einer beliebigen Anzahl von Zeichen (d.h. sie kann auch leer sein!), dann gilt:

Tabelle 7: Übersicht der Quantifier

Quantifiziertes Mustersymbol	Bedeutung
S	Alle Zeichenketten, die S genau entsprechen.
S?	Alle Zeichenketten, die S genau entsprechen oder die leere Zeichenkette.
S*	Alle Reihungen von S; insbesondere entspricht S* auch der leere Zeichenkette.
S+	Alle Reihungen von S, die S mindestens einmal enthalten. Entspricht der Formulierung: S S*
S{n,m}	Alle Zeichenketten, die aus mindestens *n*, jedoch höchstens *m* Auftreten von S bestehen. Durch *n=0* lassen sich somit optionale, nach oben begrenzte, Auftrittsanzahlen realisieren, sowie durch *n=m=0* die leere Zeichenkette ausdrücken
S{n}	Alle Zeichenketten, die aus genau *n* Auftreten von S bestehen. Entspricht der Formulierung: S{n,n}
S{n,}	Alle Zeichenketten, die aus mindestens *n* Auftreten von S bestehen. Entspricht der Formulierung: S{n} S*

Zur Darstellung nicht-druckbarer Zeichen oder von Metasymbolen werden folgende Fluchtsymbole angeboten:

Tabelle 8: Übersicht der Escape-Symbole

Mustersymbol (*escape character*)	ausgedrücktes Zeichen
\n	Zeilenumbruch (#xA)
\r	Zeilenvorschub (#xD)
\t	Tabulator (#x9)
\\	\
\\|	|
\.	.
\-	-
\^	^
\?	?
*	*
\+	+
\{	{
\}	}
\(	(
\)	)
\[	[
\]	]

Ferner sind noch eine Reihe häufig benötigter Zeichenfamilien definiert und in den Kategorien Buchstaben (Letter), Marken (Marks), Zahlen (Numbers), Interpunktion (Punctuation), Trennsymbole (Separators) sowie sonstige (Others) zusammengefaßt. Die Zeichenfamilien innerhalb dieser Kategorien entsprechen den in Unicode v3.1 definierten general categories in Namen und Aufbau.

Tabelle 9: Buchstaben

symbolische Darstellung	Bedeutung
L	(All Letters) Alle Buchstaben
Lu	(Uppercase) Alle Großbuchstaben
Ll	(Lowercase) Alle Kleinbuchstaben
Lt	(Titlecase) Sprachabhängige (potentielle) Großschreibung des ersten Buchstabens eines Wortes. (vgl. UNICODE technical report #21 sowie Derived General Category Lt).
Lm	(Modifiers) Zusammenfassung der verschiedensten Ton- und Betonungszeichen
Lo	(Others) Zusammenfassung von Zeichen, die in keine der sonstigen L-Zeichenfamilien fallen

Tabelle 10: Markierungssymbole

symbolische Darstellung	Bedeutung
M	(All Marks) Alle Markierungssymbole
Mn	(Non-Spacing) Markierungssymbole, ausschließlich Leerzeichen
Mc	(Space combining) Markierungssymbole mit Leerzeichen kombiniert
Me	(Enclosing) Markierungssymbole, die andere Zeichen umschließen

Tabelle 11: Zahlen

symbolische Darstellung	Bedeutung
N	(All Numbers) Beliebige Ziffer; daher nicht notwendigerweise arabisch. Unicode stellt eingekreiste (#x2460-#x2473), geklammerte (#x2474-#x2487) sowie Ordinalzahlen (mit abschließendem Punkt) (#x2488-#x249b) zur Verfügung.
Nd	(Decimal Digit) eine arabische Ziffer
Nl	(Letter) auf Buchstaben beruhende Zifferndarstellungen, wie römische Ziffernsymbole (#x20dd-#x217f)
No	(Other) Alle sonstigen Ziffernsymbole

Tabelle 12: Interpunktionszeichen

symbolische Darstellung	Bedeutung
P	(Punctuation) Alle Interpunktionssymbole
Pc	(Connector) Verbindende Interpunktionssymbole (z.B. Unterstrich (#x5f)
Pd	(Dash) Verschiedene Verbindungsstriche
Ps	(Open) Öffnende Bereichssymbole wie die verschiedenen Klammertypen
Pe	(Close) Schließende Bereichssymbole wie die verschiedenen Klammertypen
Pi	(Initial Quote) Öffnende Anführungszeichen. In einigen Fällen Verhalten identisch zu Ps oder Pe
Pf	(Final Quote) Schließende Anführungszeichen. In einigen Fällen Verhalten identisch zu Ps oder Pe
Po	(Other) Alle anderen Interpunktionssymbole

Tabelle 13: Separatoren

symbolische Darstellung	Bedeutung
Z	Alle Separatoren
Zs	(Space) Trennende Leerzeichen
Zl	(Line) Zeilentrenner (#x2028)
Zp	(Paragraph) Absatztrenner (#x2029)

Tabelle 14: Symbole

symbolische Darstellung	Bedeutung
S	Alle Symbole
Sm	(Math) Verschiedene mathematische Symbole (Operatoren, Pfeile, etc.)
Sc	(Currency) Währungssymbole (Dollarzeichen: #x24, Eurozeichen: #x20ac)
Sk	(Modifier) Ton- und Betonungssymbole (ähnlich zu Lm)
So	(Other) Alle anderen Symbole

Tabelle 15: Sonstige Zeichen

symbolische Darstellung	Bedeutung
C	Alle sonstigen
Cc	(Control) Nicht-druckbare Kontrollzeichen
Cf	(Format) Formatierungszeichen (z.B. syrisches Abkürzungssymbol)
Co	(Private Use) ungefähr 137500 Zeichen zur freien anwenderdefinierten Belegung
Cn	(Not Assigned) Zeichen, denen innerhalb Unicode explizit keine Belegung zugewiesen wurde

Reguläre Ausdrücke können innerhalb des pattern-Elements direkt angegeben werden. Die Zeichenkettenfamilien werden durch ihre symbolische Darstellung, eingeleitet durch \p und durch geschweifte Klammern umschlossen, dargestellt.
Zusätzlich ist durch \P das Komplement zu jeder der aufgeführten Zeichenkettenfamilien definiert.
Ergänzend kann die Definition der zulässigen Zeichengruppen vollständig wahlfrei erfolgen. Hierzu wird das Negationssymbol ^ angeboten, welches eine Aufzählung von Zeichen von der Verwendung ausschließt.
Darüberhinaus können auf der Basis simpler Mengendifferenzoperationen eigene Zeichenklassen komfortabel definiert werden.
Für die am häufigsten benötigten Zeichenklassen sind durch XML-Schema bereits vorgegebene abkürzende Schreibweisen definiert. Hierzu werden bereits die bisher vorgestellten Syntaxmechanismen angewendet.

Tabelle 16: Zeichensequenzen

abkürzende Schreibweise	entsprechende Langform
.	[^\n\r]
\s	[#x20\t\n\r]
\S	[^\s]
\i	Die initialen Zeichen eines gültigen XML-Namens; entspricht dem ersten Teil der Syntaxproduktion 5
\I	[^\i]
\c	Diejenigen Zeichen, die der XML-Syntaxproduktion 4 entsprechen
\C	[^\c]
\d	\p{Nd}
\D	[^\d]
\w	[#x0000-#x10FFFF]-[\p{P}\p{S}\p{C}] Alle Zeichen, außer der Klassen für Interpunktions- und Separatorenzeichen, sowie der Klasse der sonstigen Zeichen.
\W	[^\w]

Beispiel (eine vereinfachte, d.h. unter Nicht-Berücksichtigung von Behörden- und Diplomatennummern) deutsche Autonummer im bekannten Format: Einführende Bezeichnung des Landkreises durch mindestens einen, jedoch höchstens drei Großbuchstaben, gefolgt von einem trennenden Bindestrich, an den sich ein oder zwei weitere Großbuchstaben anschließen. Auf ein Leerzeichen folgen eine, jedoch höchstens vier Ziffern:

```
(1)<xs:simpleType name="gerAutoNummer">
(2)     <xs:restriction base="xs:string">
(3)             <xs:pattern value="\p{Lu}{1,3}-\p{Lu}{1,2} \p{Nd}{1,4}"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

Weitere Informationen: Anhang F -- Reguläre Ausdrücke -- der XML-Spezifikation

- enumeration
  Festlegung von zulässigen Werten eines Typs durch vollständige Aufzählung.
  Beispiel (die Ampelfarben: rot, gelb, grün):

```
(1)<xs:simpleType name="ampelfarben">
(2)     <xs:restriction base="xs:string">
(3)             <xs:enumeration value="rot"/>
(4)             <xs:enumeration value="gelb"/>
(5)             <xs:enumeration value="grün"/>
(6)     </xs:restriction>
(7)</xs:simpleType>
```

- whitespace
  Behandlung von white spaces innerhalb eines Elements des definierten Typs. Erlaubte Belegungen und ihre Bedeutung:
  preserve: Keine Veränderung des Inhaltes durch Normalisierung; evtl. enthaltene white spaces bleiben erhalten
  replace: Alle Vorkommen von Tabulatoren, Zeilenvorschub und Wagenrücklauf werden durch Leerzeichen (#x20)

ersetzt
collapse: Nach der Substitution gemäß *replace* werden mehrfach auftretende Leerzeichen zu einem einzigen kompaktifiziert, sowie führende und abschließende Leerzeichen entfernt.
(in Spezifikation nachschlagen)

- maxInclusive
  Höchster zulässiger numerischer Wert. Im mathematischen Sinne sind daher alle Werte kleiner oder gleich dem im value-Attribut angegebenen zugelassen.
  Beispiel (Zahlen <= 100):

```
(1)<xs:simpleType name="uHu">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:maxInclusive value="100"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

  Anmerkung: In vielen Fällen kann eine maxInclusive-Festlegung ohne Informationsverlust in eine äquivalente maxExclusive-Definition überführt werden.
- maxExclusive
  Wert „unterhalb“ (im mathematischen Sinne „kleiner“) dem alle numerischen Belegungen zugelassen sind.
  Beispiel (Zahlen < 100):

```
(1)<xs:simpleType name="uHu">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:maxExclusive value="100"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

  Als Belegungen des Typs uHu sind alle Zahlen kleiner als 100 zugelassen. Durch die Verwendung des XSD-Typen decimal als Basistyp kann auch keine verlustfreie Überführung in eine maxInclusive-Festlegung überführt werden, da hierfür die größte zugelassene Zahl fixiert werden müßte, was jedoch die Typdefinition von decimal explizit offen läßt.
- minInclusive
  Kleinster zugelassener Wert.
  Beispiel (Zahlen >= 25):

```
(1)<xs:simpleType name="uFz">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:minInclusive value="25"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- minExclusive
  Kleinster Wert, der nicht mehr zugelassen ist. Im mathematischen Sinne sind alle Zahlen, die größer sind, gültige Belegungen.
  Beispiel (Zahlen > 25):

```
(1)<xs:simpleType name="uFz">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:minExclusive value="25"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- totalDigits
  Gesamtstellen einer Zahl, gebildet aus der Summe der Vorkomma- und der Nachkommastellen.
  Beispiel (Zahlen mit höchstens sieben Stellen):

```
(1)<xs:simpleType name="myNumber">
(2)     <xs:restriction base="xs:decimal">
(3)             <xs:totalDigits value="7"/>
(4)     </xs:restriction>
(5)</xs:simpleType>
```

- fractionDigits
  Anzahl der Nachkommastellen eines Dezimalbruches.
  Beispiel (Zahl mit höchstens neuen Stellen, davon zwei Nachkommastellen):

```
(1)<xs:simpleType name="myNumber">
(2)     <xs:restriction base="decimal">
(3)             <xs:totalDigits value="9"/>
(4)             <xs:fractionDigits value="2"/>
(5)     </xs:restriction>
(6)</xs:simpleType>
```

Definition von Attributen:
Die Attributdeklaration erfolgt durch das XSD-Element attribute. Die Mächtigkeit entspricht auch hier, wie bereits für die Elemente verwirklicht, einer Obermenge der DTD. So können neben optionalen, zwingenden und konstanten

Attributen auch Aufzählungsattribute und Mengen realisiert werden. Hierbei wurde auf die Orthogonalität zum durch `simpleType` geschaffenen Typmechanismus geachtet.
Die Charakteristika (ausgedrückt in Attributen des XSD-Elements `attribute`) einer Attributdeklaration umfassen:

- **name**: Ein Doppelpunkt-freier Namen (`NCName`) gemäß Namensraumproduktion 7.
- **id**: erlaubt die eineindeutige Kennzeichnung eines Attributs durch eine Schema-weit eindeutige Zeichenkette.
- **default**: Belegung mit Vorgabewert.
- **fixed**: Konstante Belegung.
- **type**: Typ des Attributes, definiert durch einen `simpleType`.
- **form**: Legt fest, ob der Attributname im XML-Instanzdokument durch ein Namensraumpräfix eingeleitet wird (Belegung: `qualified`, andernfalls `unqualified`).
- **ref**: Verweis auf eine globale Attributdefinition.
- **use**: Verwendung des Attributes, Wert entspricht `optional`, `required` oder `prohibited`.
  Vorgabegemäß wird `optional` angenommen und das Attribut damit nicht zwingend im XML-Dokument erwartet. Den Gegensatz hierzu bildet `required`, wodurch das Attribut als zwingend anzugeben definiert wird. `prohibited` verbietet die Nutzung des Attributes im XML-Dokument.

Anmerkung: Einen Anwendungsfall der Belegung `prohibited` für use bilden Attribute, die innerhalb des Schemas bereits definiert sind, jedoch noch nicht zur allgemeinen Nutzung freigegeben wurden.

Beispiel 37: Einige Attributdefinitionen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
(3)     <xsd:attribute name="myAtt1"/>
(4)
(5)     <xsd:attribute name="myAtt2" type="xsd:decimal"/>
(6)
(7)     <xsd:attribute name="myAtt3">
(8)             <xsd:simpleType>
(9)                     <xsd:restriction base="xsd:int">
(10)                            <xsd:minInclusive value="10"/>
(11)                            <xsd:maxInclusive value="20"/>
(12)                    </xsd:restriction>
(13)            </xsd:simpleType>
(14)    </xsd:attribute>
(15)
(16)    <xsd:simpleType name="myType1">
(17)            <xsd:restriction base="xsd:string">
(18)                    <xsd:maxLength value="5"/>
(19)            </xsd:restriction>
(20)    </xsd:simpleType>
(21)    <xsd:attribute name="myAtt4" type="myType1"/>
(22)
(23)
(24)
(25)    <xsd:element name="foo">
(26)            <xsd:complexType>
(27)                    <xsd:attribute ref="myAtt1" use="optional"/>
(28)                    <xsd:attribute ref="myAtt2" use="required"/>
(29)                    <xsd:attribute ref="myAtt3" use="prohibited"/>
(30)                    <xsd:attribute ref="myAtt4"/>
(31)                    <xsd:attribute name="myAtt5" type="xsd:date" id="myDate"/>
(32)                    <xsd:attribute name="myAtt6">
(33)                            <xsd:simpleType>
(34)                                    <xsd:restriction base="xsd:float">
(35)                                            <xsd:totalDigits value="5"/>
(36)                                    </xsd:restriction>
(37)                            </xsd:simpleType>
(38)                    </xsd:attribute>
(39)            </xsd:complexType>
(40)    </xsd:element>
(41)
(42)</xsd:schema>
```

Download des Beispiels

Das Beispiel zeigt einige Varianten der Attributdeklaration. So definieren `myAtt1` mit `myAtt4` globale Attribute, die innerhalb verschiedener Elemente verwendet werden können. Hierdurch wird die bereits für Elemente verwirklichte Mimik der einmaligen Deklaration und anschließenden beliebigen Verwendung auch auf Attribute ausgedehnt. Die Nutzung der so deklarierten Attribute geschieht durch das `ref`-Attribut innerhalb des `Attribute`-Elements des beherbergenden Elements.
`myAtt1` definiert ein typenloses Attribut, dem vorgabegemäß der allgemeinste Typ `anyType` zugeordnet wird. Die Angabe dieses Attributes ist optional (`use="optional"`), was der Vorgabe entspricht.
Der XSD-Standardtyp `decimal` findet zur Definition des Attributs `myAtt2` Verwendung. Die zwingend anzugebenden (`use="required"`) Inhalte dieses Attributs werden durch einen XML-Schema-Parser auf Typkonformität geprüft.
`myAtt3` veranschaulicht die Bildung eines anonymen (inneren) atomaren Typen zur Definition eines Attributs. Der durch Restriktion gebildete neue Datentyp steht ausschließlich innerhalb des Attributs `myAtt3` zur Verfügung. Die Syntax der Datentypspezialisierung entspricht der im vorhergehenden Abschnitt diskutierten. Zudem ist die Verwendung des Attributes innerhalb eines XML-Dokumentes untersagt; ausgedrückt durch die Belegung `use="prohibited"`

Analog der Typisierung eines Elementinhaltes durch einen anwenderdefinierten Typen gestaltet sich das Vorgehen für Attribute. Veranschaulicht wird dies durch die Definition von `myAtt4`. Sie greift auf den eigen-definierten Typen `myType1` zurück.
Dem Attribut `myAtt5` ist zusätzlich zur Benennung, die innerhalb des verwendenden Elementes eindeutig sein sollte, ein Dokument-weiter Schlüssel (`id`) zugeordnet.
Innerhalb des Elements `foo` werden die fünf zuvor definierten Attribute verwendet. Trotz der Reihenfolge der Definitionen im `complexType`-Element verfügen die Attribute im XML-Instanzdokument -- auch bei der Verwendung von XML-Schema -- über keinerlei Reihenfolge (vgl. XML-Spezifikation).
Zusätzlich enthält die Elementdefintion für `foo` mit `myAtt6` ein „lokales“ Attribut. Diese Definitionsvariante entspricht am ehesten der der Document Type Definition, da sie eine Wiederverwendung außerhalb des definierenden Elements ausschließt.

Beispiel 38: Vollständiges XML-Schema der Projektverwaltung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(3)     <xsd:element name="Nachname" type="xsd:string"/>
(4)     <xsd:complexType name="PersonType">
(5)             <xsd:sequence>
(6)                     <xsd:element ref="Vorname" maxOccurs="unbounded"/>
(7)                     <xsd:element ref="Nachname" maxOccurs="unbounded"/>
(8)                     <xsd:element name="Qualifikationsprofil" type="QualifikationsprofilType"
minOccurs="0"/>
(9)             </xsd:sequence>
(10)            <xsd:attribute name="PersID" type="xsd:ID" use="required"/>
(11)            <xsd:attribute name="Gehaltsgruppe" default="1a">
(12)                    <xsd:simpleType>
(13)                            <xsd:restriction base="xsd:NMTOKEN">
(14)                                    <xsd:enumeration value="1"/>
(15)                                    <xsd:enumeration value="1a"/>
(16)                                    <xsd:enumeration value="2"/>
(17)                            </xsd:restriction>
(18)                    </xsd:simpleType>
(19)            </xsd:attribute>
(20)            <xsd:attribute name="mitarbeitInProjekt" type="xsd:IDREFS" use="required"/>
(21)    </xsd:complexType>
(22)    <xsd:complexType name="ProjektType">
(23)            <xsd:attribute name="ID" type="xsd:ID" use="required"/>
(24)            <xsd:attribute name="date" type="xsd:date"/>
(25)            <xsd:attribute name="budget" default="10000.00">
(26)                    <xsd:simpleType>
(27)                            <xsd:restriction base="xsd:double">
(28)                                    <xsd:fractionDigits value="2"/>
(29)                            </xsd:restriction>
(30)                    </xsd:simpleType>
(31)            </xsd:attribute>
(32)            <xsd:attribute name="Projektleiter" type="xsd:IDREF" use="required"/>
(33)            <xsd:attribute name="Mitarbeiter" type="xsd:IDREFS" use="required"/>
(34)    </xsd:complexType>
(35)    <xsd:element name="ProjektVerwaltung">
(36)            <xsd:complexType>
(37)                    <xsd:sequence>
(38)                            <xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/>
(39)                            <xsd:element name="Projekt" type="ProjektType"
maxOccurs="unbounded"/>
(40)                    </xsd:sequence>
(41)                    <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(42)            </xsd:complexType>
(43)    </xsd:element>
(44)    <xsd:complexType name="QualifikationsprofilType" mixed="true">
(45)            <xsd:choice minOccurs="0" maxOccurs="unbounded">
(46)                    <xsd:element ref="Qualifikation"/>
(47)                    <xsd:element ref="Leistungsstufe"/>
(48)                    <xsd:any namespace="http://www.w3.org/1999/xhtml"/>
(49)            </xsd:choice>
(50)    </xsd:complexType>
(51)    <xsd:element name="Qualifikation" type="xsd:string"/>
(52)    <xsd:element name="Leistungsstufe" type="xsd:string"/>
(53)    <xsd:element name="Vorname" type="xsd:string"/>
(54)</xsd:schema>
```

Download des Beispiels

Abschließend eine gültige (sowohl *valid* als auch *schema valid*) Dokumentinstanz der Projektverwaltungsstruktur.

Beispiel 39: Gültiges Projektverwaltungsdokument

```
(1)<?xml version="1.0" encoding="ISO-8859-1"?>
(2)<ProjektVerwaltung
(3)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)     xsi:noNamespaceSchemaLocation="http://www.jeckle.de/vorlesung/xml/examples/
projektverwaltung.xsd">
(5)     <Person PersID="Pers01" mitarbeitInProjekt="Prj01">
(6)             <Vorname>Hans</Vorname>
(7)             <Nachname>Hinterhuber</Nachname>
(8)     </Person>
(9)     <Person PersID="Pers02" mitarbeitInProjekt="Prj02">
(10)            <Vorname>Franz</Vorname>
(11)            <Vorname>Xaver</Vorname>
(12)            <Nachname>Obermüller</Nachname>
(13)            <Qualifikationsprofil>
(14)     IT-Kompetenz verschiedene Betriebssysteme und <Leistungsstufe>professionelle</
Leistungsstufe>
(15)                    <Qualifikation>Programmierung</Qualifikation> verschiedener
Programmiersprachen
(16)     <Qualifikation>Entwickler</Qualifikation> von 1988-1990
(17)     <Qualifikation>Projektleiterfunktion</Qualifikation> von 1990-93 im X42-Projekt in
Abteilung AB&amp;C
(18)   </Qualifikationsprofil>
(19)    </Person>
(20)    <Person PersID="Pers03" mitarbeitInProjekt="Prj02">
(21)            <Vorname>Fritz</Vorname>
(22)            <Nachname>Meier</Nachname>
(23)    </Person>
(24)    <Projekt ID="Prj01" Projektleiter="Pers01" Mitarbeiter="Pers01"/>
(25)    <Projekt ID="Prj02" Projektleiter="Pers02" Mitarbeiter="Pers03"/>
(26)</ProjektVerwaltung>
```

Download des Beispiels

Werkzeuge:
Zwar existiert -- wie für alle XML-Dokumente -- die Möglichkeit, Dokument Typ Definitionen und XML-Schemata „per Hand“ mit einem Texteditor zu erstellen, jedoch ist dieses Vorgehen, insbesondere für umfangreiche XML-Vokabulare, zeitaufwendig und fehlerträchtig. Zusätzlich läßt die rein textuelle Formulierung die entstehenden Schemadokumente schnell unübersichtlich werden.
Inzwischen existieren einige gute DTD- und Schemaeditoren, die zumeist neben visueller Syntaxhervorhebung auch die kontextsensitive Editierung erlauben und so eine wesentliche Erleichterung der Schemaerzeugung bilden. Gleichzeitig bieten die meisten verfügbaren Werkzeuge dieser Klasse auch Möglichkeiten zur Validierung des erzeugten Schemas an.
Ergänzend wird vielfach auch eine graphische Repräsentation der DTD- oder XSD-Struktur angeboten.
Die Abbildungen zeigen Ansichten der Werkzeuge XML Authority bzw. XML Spy

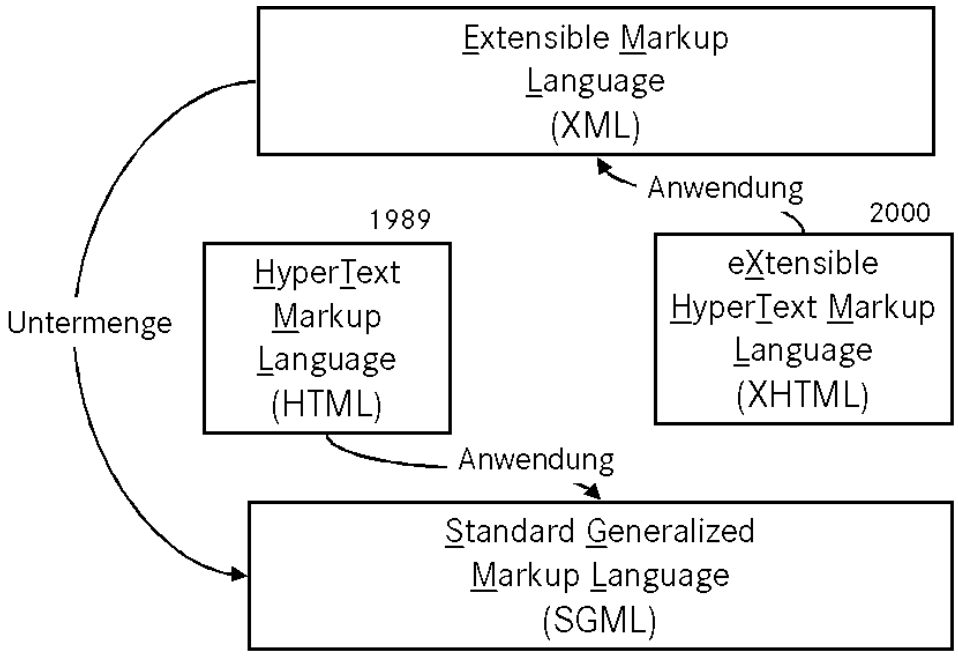


Web-Referenzen 9: Weiterführende Links und Werkzeuge

- XML Schema Part 0: Primer
- XML Schema Part 1: Structures
- XML Schema Part 2: Datatypes
- XML Schema @ Cover-Pages
- Parsing the Atom -- Diskussion über die Vor- und Nachteile inhärent komplexer atomarer Typen
- Schema-Informationen @ jeckle.de
- XML-Authority (DTD- und XSD-Editor)
- XML Spy (DTD- und XSD-Editor)

1.6 XHTML -- XML und das Web

Die Verbindung zwischen der generischen Metasprache XML und dem World Wide Web datiert zurück bis vor die Entstehung der XML ...
Wie bereits im Eingangskapitel erwähnt, bildet die Hypertext Markup Language HTML die eigentliche Initialzündung des World Wide Web. Technisch ist sie als Anwendung der normierten generischen Metasprache SGML realisiert und kann heute als vermutlich bekannteste und verbreitetste Auszeichnungssprache überhaupt gelten.

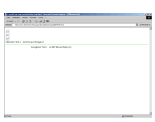
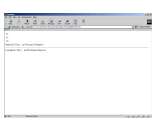


Die Graphik faßt noch einmal die Beziehung von HTML und SGML zusammen, und schlägt eine Brücke in die XML-Welt.
Ursprünglich setzte Tim Berners-Lee das *Ur-HTML* 1989 als SGML-Anwendung durch Definition einer geeigneten (SGML-)Document Type Definition um (vgl. IETF RFC 1866). Zwar ähnelt die für HTML definierte Syntax der später für XML-Dokumente eingeführten, jedoch läßt sich an einigen Stellen (deutlich) die Abkunft von SGML ablesen. Dies wird insbesondere bei Syntaxkonstrukten offenkundig, die zwar in SGML erlaubt, jedoch in der später gebildeten Untermenge XML illegal sind. Die fehlenden schließenden Tags (beim `img`- oder `br`-Element) oder die gelockerte Attributsyntax (beim `compact`-Attribut des `ul`-Elements) können hierfür als bekannte Beispiele dienen.
Diese SGML-Sprachmittel werden entweder durch den Anwender vorgegeben (wie `SHORTTAG = YES` für das `img`-Element) oder sind inhärent zur Kompaktifizierung der Instanzstrukturen (wie die Attributminimierung) vorgegeben. Sie stellen einen Gutteil der Unübersichtlichkeit und Komplexität der Sprache SGML dar, da durch sie prinzipiell unterschiedliche Darstellungen identischer Bedeutung erlaubt werden. Als Resultat des sich (theoretisch) ergebenden Umsetzungsaufwandes realisieren HTML-Browser bis heute nicht die volle Mächtigkeit von SGML, die notwendig wäre um HTML korrekt zu interpretieren, sondern nur einen herstellerabhängigen Ausschnitt. Daher kommt es immer wieder zu sichtbaren Inkompatibilitäten der HTML-Interpretation verschiedener Web-Browser.
Zusätzlich haben die verschiedenen Hersteller im Laufe der Zeit den ursprünglichen HTML-Standard um eigene proprietäre Elemente erweitert. Naturgemäß werden diese neuen Elemente nur durch den Browser des jeweiligen Herstellers interpretiert; alle anderen Browser ignorieren die ihnen unbekannten Tags spezifikationsgemäß und zeigen stattdessen nur den Elementinhalt an.
Die nachfolgenden Graphiken zeigen dies an einem einfachen Beispiel:

Beispiel 40: Ein (höchst) inkompatibles HTML-Dokument

```
(1)<html>
(2)   <table rules=none>
(3)      <tr>
(4)         <td>x1</td>
(5)      <tr>
(6)         <td>x2
(7)      <tr>
(8)         <td>x3
(9)   </table>
(10)   <blink>blinkender Text (... im Netscape Navigator)</blink>
(11)   <hr color="86db7f">
(12)   <marquee>beweglicher Text (... im MS Internet Explorer)</
marquee>
```

Download des Beispiels

Das dargestellte HTML-Dokument nutzt das in der HTML-Spezifikation ab der Version 3.2 vorgesehene `rules`-Attribut. Die Abbildungen zeigen jedoch, daß es nur durch den Internet Explorer ausgewertet und dargestellt wird. Die Tabellendefinition nutzt in der wiedergegebenen Form die Freiheiten des HTML-Standards aus, und verzichtet weitestgehend auf schließende Tags. Beide Browser geben trotzdem die Struktur korrekt wieder.
Bei den drei abschließenden Elementen handelt es sich um herstellerspezifische Erweiterungen durch Netscape (`blink`) bzw. Microsoft (`color`-Attribut und `marquee`-Element).

Einerseits zur (Wieder-)Vereinheitlichung der inzwischen offen konkurrierenden HTML-Dialekte, andererseits zur leichteren technischen Umsetzbarkeit der HTML-Grammatik initiierte das World Wide Web Konsortium 1998 die XHTML-Aktivität, welche sich die Umsetzung von HTML als XML-Anwendung zum Ziel setzte.

Mit der Verabschiedung des HTML-Nachfolgers XHTML v1.0 im Januar 2000 wurde für die bis dato prominenteste Web-Sprache die Abkehr von SGML vollzogen. Im Kern stellt die neue Sprache die Reformulierung der letzten SGML-basierten HTML-Variante 4.01 in XML dar. Wie bereits die Ur-Sprache verwendet auch XHTML (zunächst) den Grammatikmechanismus der Document Type Definition und ist „äußerlich" (d.h. bei Betrachtung der Instanzdokumente) nicht von der Vorgängersprache zu unterscheiden. Genaugenommen sind die angebotenen Minimierungsmöglichkeiten für SGML optional und müssen nicht zwingend genutzt werden. Daher ist weiterhin jedes gültige XHTML v1.0-Dokument ebenfalls konform zur HTML v4.01-Definition.

Nachfolgend sind die Änderungen an HTML zusammengetragen, welche zum XHTML-Standard führten:

- Groß- und Kleinschreibung der Element- und Attributnamen ist signifikant.
- Attribute müssen über Wertangaben verfügen, die Definition `<ul compact>` als Kurzschreibweise von `<ul compact="compact">` ist nicht mehr zugelassen.
- Attributwerte müssen in Anführungszeichen eingeschlossen sein.
- Im Instanzdokument dürfen die Zeichen & und < nicht direkt auftreten, sondern müssen durch Enititätsrefernzen & bzw. < ausgedrückt werden.
- Bisher optionale Endtags sind in XHTML zwingend anzugeben. Elemente enden daher wie in XML üblich mit dem schließenden Tag.
- Korrekte Element-Schachtelung ist zwingend einzuhalten.
- Leere Elemente müssen über schließende Tags verfügen, oder durch die abkürzende Schreibweise dargestellt werden.
- Das Instanzdokument muß über genau ein Wurzelelement (üblicherweise `html`) verfügen.
- Innerhalb der Kommentarsyntax müssen die beiden Bindestriche nach der Eröffnungssequenz `<!` folgen.
- Die XML-Encodingdeklaration ist für alle Zeichensätze außer UTF-8 und -16 zwingend anzugeben.

Wird ein HTML-Dokument entsprechend modifiziert, so erhält man ein wohlgeformtes XML-Dokument. Wird zusätzlich die DOCTYPE-Deklaration angegeben, und so die XHTML-DTD referenziert, so kann zusätzlich die Gültigkeit sichergestellt werden.
Die aktuell verfügbaren Browser können überwiegend bereits XHTML v1.0-konforme Dokumente darstellen.
Im Detail bedeutet dies, daß neben der Einhaltung der durch die DTD definierten Strukturen auch die Nutzung des XHTML-Namensraumes für das gesamte Dokument erfolgen muß. Konsequenterweise muß jedes XHTML-Dokument ferner die entsprechende XHTML-DTD referenzieren. Zur korrekten Behandlung bereits existierender Dokument-Instanzen, die noch Elemente früherer HTML-Versionen beinhalten, welche nicht mehr durch den Standard unterstützt werden, sind insgesamt drei HTML-DTDs durch den Standard definiert. Neben den DTDs zur Darstellung Frame-basierter (*frameset-DTD*) bzw. „normaler" (*strict-DTD*) Dokumente ermöglicht die *transitional* Dokument Typ Definition den leichteren Übergang zu XHTML.
Zur automatisierten Anpassung existierender HTML-Dokumente auf den neuen Sprachstandard hat das W3C das kostenfreie Werkzeug Tidy veröffentlicht.
Der nachfolgende XHTML-Code wurde mit diesem Werkzeug automatisch aus dem vorangegangenen Beispiel erzeugt. (Aufruf: `tidy -asxml incompatibleHTML.html > validXHTML.html`)

Beispiel 41: Ein einfaches XHTML-Dokument

```
(1)<?xml version="1.0"?>
(2)<!DOCTYPE html PUBLIC
(3)     "-//W3C//DTD XHTML 1.0 Transitional//EN"
(4)     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
(5)<html xmlns="http://www.w3.org/1999/xhtml">
(6)     <head>
(7)             <meta name="generator" content="HTML Tidy, see www.w3.org" /
>
(8)             <title></title>
(9)     </head>
(10)    <body>
(11)            <table rules="none">
(12)                    <tr>
(13)                            <td>x1</td>
(14)                    </tr>
(15)                    <tr>
(16)                            <td>x2</td>
(17)                    </tr>
(18)                    <tr>
(19)                            <td>x3</td>
(20)                    </tr>
(21)            </table>
(22)            <blink>blinkender Text (... im Netscape Navigator)</blink>
(23)            <hr color="86db7f" />
(24)            <marquee>beweglicher Text (... im MS Internet Explorer)</
marquee>
(25)    </body>
(26)</html>
```

Modulares XHTML

XHTML v1.1 dekomponiert das ehemals durch genau eine DTD dargestellte HTML-Vokabular in einzelne problemspezifische Module. Zusätzlich wird die Sprache um die Ruby-Anmerkungen erweitert. Hierbei handelt es sich um einen simplen Annotationsmechanismus für beliebige Anmerkungen im Text, wie er im südasiatischen Sprachraum breite Verwendung findet.

Breite Aufmerksamkeit dürften die in XHTML v1.1 verwirklichten Modularisierungsaktivitäten wecken. Sie haben es

sich sich zum Ziel gesetzt, die Sprache in verschiedene eigenständige Module aufzuteilen, um so die Wiederverwendung einzelner Teilbereiche (wie z.B. Tabellen, Graphiken, Listen) in anderen XML-Sprachen zu ermöglichen. Gleichzeitig können ausgewählte Module für bestimmte Endgeräte umgesetzt werden, beispielsweise für ressourcenbeschränkte mobile Endgeräte.
Im Detail definiert der Standard 22 Einzelmodule mit ihren spezifischen Aufgaben. Das Zentrum der Dekomposition bilden die vier Kernmodule *Struktur*, *Text*, *Hypertext* und *Listen*. Sie müssen zwingend durch eine Implementierung umgesetzt werden, um als konform zu XHTML v1.1 anerkannt zu werden.
Nachfolgend sind die Module und ihre Aufgabe sowie typische Elemente daraus zusammengefaßt:

- **Attribute Collections**: Versammelt XML-Attribute die für verschiedenste XHTML-Elemente angegeben werden können.
  Die Attribute Collections werden durch fünf Einzelsammlungen (Core, I18N (Internationalisierung), Events, Style, Common) gebildet, die jeweils ein oder mehrere Attribute umfassen.
  Beispiele: `class` (Core-Modul), `onclick` (Events-Modul), `style` (Style-Modul)
- **Structure Module**: Bestandteil des XHTML-Kernmoduls.
  Definiert die zentralen strukturellen Elemente von XHTML, die als Inhalt in den meisten Dokumenten der XHTML-Familie auftreten.
  Beispiele: `html`, head, body, ...
- **Text Module**: Bestandteil des XHTML-Kernmoduls.
  Definiert alle grundlegenden Containerelemente sowie deren Attribute und Inhaltsmodelle.
  Beispiele: `abbr`, `p`, `strong`, ...
- **Hypertext Module**: Bestandteil der XHTML-Kernmodule.
  Definiert das (eine!) Element welches zur Verlinkung in XHTML-basierten Hypertexten verwendet wird.
  Beispiel: `a`
- **List Module**: Bestandteil des XHTML-Kernmoduls.
  Definiert alle Listen-orientierten Elemente und deren Attribute.
  Beispiele: `ul`, `li`, `dl`, ...
- **Presentation Module**: Bestandteil des XHTML-Texterweiterungsmoduls.
  Definiert einige einfache Elemente zur Realisierung präsentationsorientierter Darstellungen.
  Beispiele: `b`, `small`, `tt`, ...
- **Edit Module**: Bestandteil des XHTML-Texterweiterungsmoduls.
  Definiert Elemente zum Ausdruck editorieller Änderungen.
  Beispiele: `ins`, `del`
- **Bi-directional Text Module**: Bestandteil des XHTML-Texterweiterungsmoduls.
  Definiert ein Element zur Behandlung von Text in verschiedenen Lesrichtungen.
  Beispiel: `bdo`
- **Basic Forms Module**: Bestandteil des XHTML-Formularmoduls.
  Versammelt die gurndlegenden Elemente zur Realisierung von Formularen.
  Beispiel: `form`, `input`, `option`, ...
- **Forms Module**: Bestandteil des XHTML-Formularmoduls.
  Versammelt alle in HTML v4.0 definierten Formularelemente. Dieses Modul hat daher Überschneidungen mit dem *Basic Forms* Modul.
  Beispiele: `form`, `input`, `button`, ...
- **Basic Tables Module**: Bestandteil des XHTML-Tabellenmoduls.
  Definiert die grundlegendsten Elemente die zur Formulierung einer Tabelle notwendig sind.
  Beispiele: `table`, `td`, `th`, ...
- **Tables Module**: Bestandteil des XHTML-Tabellenmoduls.
  Umfaßt als Obermenge des *Basic Tables* Moduls alle Elemente die zur Definition einer Tabelle benötigt werden.
  Beispiele: `table`, `td`, `thead`, ...
- **Image Module**: Umfaßt das ausschließlich Element zur referenzierenden Einbettung von Bildern.
  Beispiel: `img`
- **Client-side Image Map Module**: Versammelt Elemente zur clientseitigen Verarbeitung von Ereignissen die durch Anwenderinterkation mit Bildern entstehen. Dieses Modul erfordert die Umsetzung des *Image* Moduls.
  Beispiel: Die Attribute `coords` oder `usemap` für die Elemente `a`, `area`, ...
- **Server-side Image Map Module**: Versammelt Elemente zur serverseitigen Verarbeitung von Ereignissen die durch Anwenderinteraktion mit Bildern entstehen. Dieses Modul erfordert die Umsetzung des *Image* Moduls.
  Beispiel: Das Attribut `ismap` für die Elemente `img` und `input`
- **Object Module**: Definiert ein Element zur allgemeinen Einbettung beliebiger Elemente.
  Beispiel: `object`
- **Frames Module**: Umfaßt die Elemente die zur Formulierung von Frames, d.h. die Anzeige separierter XHTML-Dokumente in einem Darstellungsfenster.
  Beispiel: `frameset`, `frame`, `noframes`
- **Target Module**: Ermöglicht es aus Frames heraus andere anzusprechen bzw. neue Fenster zu öffnen.
  Beispiel: Das `target`-Attribut für die Elemente `a`, `link`, `base`, ...
- **Iframe Module**: Definiert ein Element zur Realisierung von inline Frames
  Beispiel: `iframe`
- **Intrinsic Events Module**: Definiert Attribute zur Behandlung von benutzergetriebenen Ereignissen.
  Beispiel: `onblur`, `onload`, `onsubmit`, ... für die Elemente `a`, body, `form`, ...
- **Metainformation Module**: Definiert ein Element zur Darstellung von Metainformation über XHTML-Seiten.
  Beispiel: `meta`
- **Scripting Module**: Definiert Elemente zur Einbettung von aktiven Elementen in XHTML-Seiten durch Scriptsprachen.
  Beispiel: `noscript`, `script`
- **Style Sheet Module**: Definiert ein Element zur Einbettung von präsentationsorientierter Information in XHTML-Dateien.
  Beispiel: `style`
- **Link Module**: Definiert ein Element, das zur Referenzierung externer Ressourcen verwendet werden kann.
  Beispiel: `link`
- **Base Module**: Definiert ein Element welches verwendet werden kann um die Basis URI eines Dokuments festzulegen. Diese URI wird im Rahmen einer Auswertung gemäß XMLBase herangezogen.
  Beispiel: `base`

- **Name Identification Module**: Dieses -- inzwischen als *veraltet* (engl. deprecated) gekennzeichnete -- Modul definiert ein Attribut zur Benennung verschiedener XHTML-Elemente.
  Beispiel: `name`-Attribut für die Elemente `a`, `applet`, `form`...
- **Legacy Module**: Dieses Modul versammelt eine Reihe von Attributen und Elementen, die bereits in vorhergehenden (X)HTML-Versionen als veraltet gekennzeichnet wurden.
  Beispiel: `basefont`, `center`, `font`, ...

Das Schema-Fragment zeigt die Nutzung beliebiger Elemente aus dem XHTML-Namensraum im `QualifikationsprofilType`.

Beispiel 42: Nutzung von XHTML in anderen XML-Sprachen

```
(1)...
(2)     <xsd:complexType
(3)             name="QualifikationsprofilType"
(4)             mixed="true">
(5)             <xsd:choice
(6)                     minOccurs="0"
(7)                     maxOccurs="unbounded">
(8)                     <xsd:element ref="Qualifikation"/>
(9)                     <xsd:element ref="Leistungsstufe"/>
(10)                    <xsd:any
(11)                            namespace="http://www.w3.org/1999/
xhtml"
(12)                            minOccurs="0"
(13)                            maxOccurs="unbounded"/>
(14)            </xsd:choice>
(15)    </xsd:complexType>
(16)...
(17)    <xsd:element
(18)            name="Qualifikationsprofil"
(19)            type="QualifikationsprofilType"
(20)            minOccurs="0"/>
(21)...
```

Eine gültige Dokumentinstanz kann innerhalb des Elements `Qualifikationsprofil` jedes Element aus dem XHTML-Namensraum beinhalten:

Beispiel 43: Nutzung des XHTML-Namensraumes in einem eigenen Dokument

```
(1)<?xml version="1.0" encoding="ISO-8859-1"?>
(2)<ProjektVerwaltung
(3)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)     xmlns:xhtml="http://www.w3.org/1999/xhtml"
(5)     xsi:noNamespaceSchemaLocation="http://www.jeckle.de/vorlesung/xml/examples/
projektverwaltung.xsd">
(6)     <Person PersID="Pers01" mitarbeitInProjekt="Prj01">
(7)             <Vorname>Hans</Vorname>
(8)             <Nachname>Hinterhuber</Nachname>
(9)     </Person>
(10)    <Person PersID="Pers02" mitarbeitInProjekt="Prj02">
(11)            <Vorname>Franz</Vorname>
(12)            <Vorname>Xaver</Vorname>
(13)            <Nachname>Obermüller</Nachname>
(14)            <Qualifikationsprofil>
(15)                    <xhtml:u>IT-Kompetenz</xhtml:u>
(16)                    <xhtml:em>verschiedene</xhtml:em> Betriebssysteme und
(17)               <Leistungsstufe>professionelle</Leistungsstufe>
(18)                    <xhtml:em>
(19)                            <Qualifikation>Programmierung</Qualifikation>
(20)                    </xhtml:em>
(21)               verschiedener Programmiersprachen
(22)               <xhtml:em>
(23)                            <xhtml:u>
(24)                                    <Qualifikation>Entwickler</Qualifikation>
(25)                            </xhtml:u>
(26)                    </xhtml:em> von 1988-1990
(27)               <xhtml:u>
(28)                            <Qualifikation>Projektleiterfunktion</Qualifikation>
(29)                    </xhtml:u>
(30)               von <xhtml:b>1990-93</xhtml:b> im X42-Projekt in Abteilung AB&amp;amp;C
(31)            </Qualifikationsprofil>
(32)    </Person>
(33)    <Person PersID="Pers03" mitarbeitInProjekt="Prj02">
(34)            <Vorname>Fritz</Vorname>
(35)            <Nachname>Meier</Nachname>
(36)    </Person>
(37)    <Projekt ID="Prj01" Projektleiter="Pers01" Mitarbeiter="Pers01"/>
(38)    <Projekt ID="Prj02" Projektleiter="Pers02" Mitarbeiter="Pers03"/>
(39)</ProjektVerwaltung>
```

Download des Beispiels

XHTML und XML Schema

Parallel zur Modularisierung und den laufenden Erweiterungsaktivitäten zu XHTML wurde inzwischen auch an der Neufassung der Strukturbeschreibung gearbeitet. Da der Sprachumfang der SGML-basierten Sprache HTML mit der Reformulierung in XML (XHTML v1.0) nicht verändert wurde (auch XHTML v1.1 läßt diesen unangetastet) bestand auch keinerlei Notwendigkeit zur Überarbeitung der DTD-basierten Vokabularbeschreibung.
Mit dem abzusehenden Übergang zur nächsten Version des XHTML-Vokabulars, welche erstmals seit 1999 neue Elemente in die Sprache einführen wird, vollzieht sich auch die Umstellung auf XML Schema.
Bereits für XHTML v1.0 liegt Formulierung als XML Schema in einer nicht normativen W3C Note vor.

Ausblick auf XHTML v2.0

XHTML markiert die erste „echte“ Weiterentwicklung der XHTML-Sprachfamilie seit ihrer Löslösung von SGML und der daran anschließenden Fundierung auf XML.
Das gegenwärtig erst im Status eines Working Draft zugängliche Vokabular definiert einige neue Elemente und entfernt einige bekannte Konstrukte. Daher stellt es keine abwärtskompatible Weiterentwicklung der existierenden XHTML-Sprachfamilie dar, sondern entwirft vielmehr eine neue Hypertextsprache.
Nachfolgend werten einige Erweiterungen und Modifikationen gegenüber dem vertrauten XHTML an Beispielen diskutiert.

XHTML v2 wird keine explizite Angabe des Zeilenumbruchs durch den Anwender mehr unterstützen, sondern dies vollständig der Darstellungskomponente überlassen. Konkret wird das `br`-Element als veraltet gekennzeichnet und stattdessen zukünftig durch das neu eingeführte Element `line` ersetzt werden welches sich in ähnlicher Weise nutzen läßt.
Beispiel 44 zeigt ein XHTML-Fragment welches korrekt im Sinne der Festlegungen der ersten XHTML-Generation formuliert ist. Beispiel 45 stellt diesem die XHTML v2 konforme Darstellung gegenüber.

Beispiel 44: Ein gültiges XHTML v1.x-Dokumentfragment

```
(1)<p>
(2)Habe nun, ach! Philosophie,<br/>
(3)Juristerei und Medizin,<br/>
(4)Und leider auch Theologie!<br/>
(5)</p>
```

Download des Beispiels

Beispiel 45: ... XHTML v2-konforme Formulierung

```
(1)<p>
(2)<line>Habe nun, ach! Philosophie,</line>
(3)<line>Juristerei und Medizin,</line>
(4)<line>Und leider auch Theologie!</line>
(5)</p>
```

Download des Beispiels

Die Beispiele 46 und 47 illustrieren die wohl augenscheinlichste Neuerung des XHTML v2 Sprachvorschlages. Das bisher zur Referenzierung von Bilddaten vorhergesehene Element `img` wird zugunsten des bereits existierenden Elements `object` für veraltet erklärt.
Hintergrund dieser Entscheidung ist der Versuch der Aufhebung der Asymmetrie in der Adressierung von referenzierten Bilddaten und sonstigen externen Daten eines nicht näher definierten Formats wie beispielsweise Applets.
Innerhalb des `object`-Elements übernimmt das Attribut `data` die Rolle der bisher durch `src` ausgedrückten Lokalisationsreferenz. Neu hinzu kommt die die Typisierung des referenzierten Objekts mittels MIME-Type, welcher im `type`-Attribut angegeben wird.
Zusätzlich zeigt das Beispiel 47 die Nutzung der `standby`-Angabe welche zur Aufnahme von Text dient der während des Objektladevorganges (d.h. der Dereferenzierung der durch `data` identifizierten Ressource) dem Anwender präsentiert werden kann.

Beispiel 46: Referenzierung einer externen Bilddatei in XHTML v2

```
(1)<img src="http://www.jeckle.de/images/logo.gif" alt="ein Logo" />
```

Download des Beispiels

Beispiel 47: ... und in XHTML v2

```
(1)<object
(2)     data="http://www.jeckle.de/images/logo.gif"
(3)     type="image/gif"
(4)     standby="ein Logo" />
```

Download des Beispiels

Folgenschwerste Modifikation der bestehenden XHTML-Struktur dürfte die Entfernung der Überschriftenelemente h1 mit h6 sein. Diese Elemente welche bisher die hierarchische Position einer Überschrift absolut durch die Wahl des Elementtyps ausdrückten werden zukünftig durch zugunsten eines allgemeinen Hierarchieelements h nicht mehr unterstützt. Ein Grund dieser Entscheidung mag in der praktischen Verwendung der bisher angebotenen Überschriftselemente liegen. So wurden diese häufig zur Erreichung eines gewissen visuellen Verhaltens herangezogen, welches sich aus dem im Elementnamen ablesbaren Verhältnis der Elemente zueinander ablesen lies. So wird typischerweise ein Überschriftselement der Ebene zwei (h2) in einer kleineren oder schwächeren optischen Darstellung umgesetzt als eines der Ebene eins (h1).
In XHTML v2 ergibt sich die visuelle Präsentation ausschließlich aus dem Verhältnis der h-Elemente zu der sie umgebenden section. Daher entspricht die Formulierung des Beispiels 48 der aus 49 semantisch.

Beispiel 48: Überschriftsebenen in XHTML v1

```
(1)<h2>Der Tragödie erster Teil</h2>
(2)<h3>Nacht</h3>
(3)<h3>Vor dem Tor</h3>
(4)...
(5)<h2>Der Tragödie zweiter Teil</h2>
(6)<h3>Anmutige Gegend</h3>
(7)<h3>Hochgewölbtes enges gotisches Zimmer</h3>
(8)...
```

Download des Beispiels

Beispiel 49: Äquivalente Formulierung in XHTML v2

```
(1)<section>
(2)   <h>Der Tragödie erster Teil</h>
(3)   <section>
(4)      <h>Nacht</h>
(5)   </section>
(6)   <section>
(7)      <h>Vor dem Tor</h>
(8)   </section>
(9)</section>
(10)...
```

Download des Beispiels

Die unauffälligste und gleichermaßen wirkungsmächtigste Neuerung ergibt sich aus der Inklusion des href-Attributes in die Zusammenstellung der allgemeine verwendbaren Attribute der *Common Attribute Collection*.
Diese „Aufwertung“ des genannten Attributs eröffnet die Möglichkeit beliebige XHTML-Elemente als Quellen von Hyperlinkverknüpfungen heranzuziehen wie es 50 zeigt.

Beispiel 50: Erweiterte Hyperlinks in XHTML v2

```
(1)<p href="http://spiegel.gutenberg.de/faust">
(2)   <line>Habe nun, ach! <span href="http://www.philosophie.de">Philosophie</span>,</line>
(3)   <line><acronym title="Juristerei" xml:lang="de-au" href="http://www.austria.at">Jus</acronym>
(4)   <abbr title="und" href="http://www...">u.</abbr> Medizin,</line>
(5)   <line href="http://www.was-hat-goethe-gegen-theologie.de">Und leider auch Theologie!</line>
(6)</p>
```

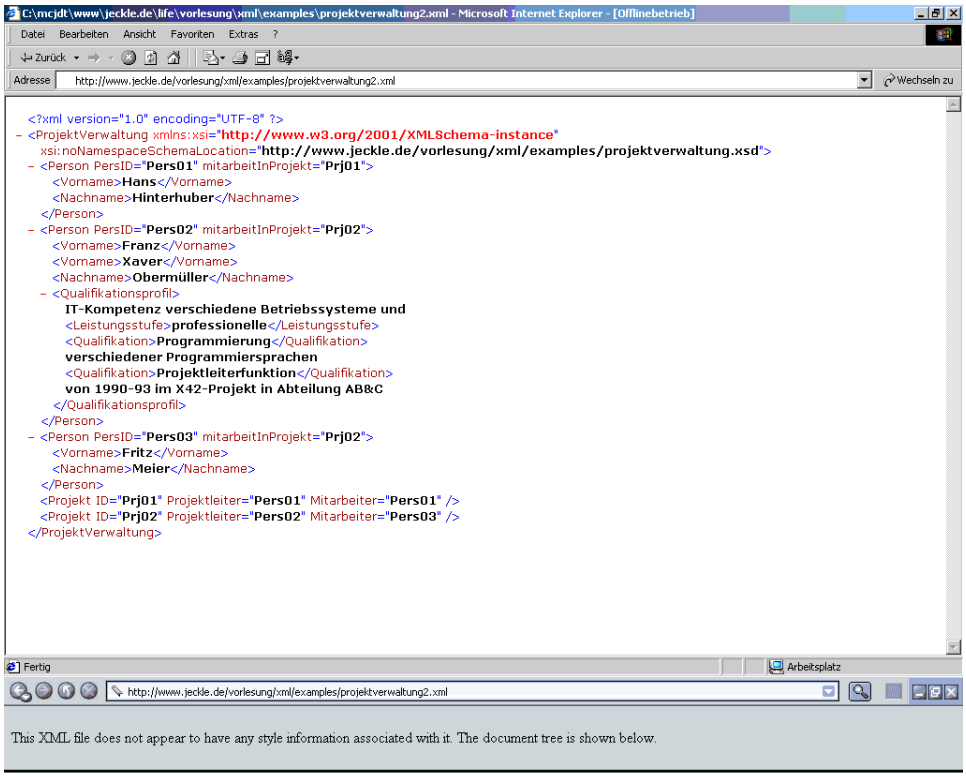
Download des Beispiels

Obwohl das Beispiel auf den ersten Blick einen interessanten Mächtigkeitsgewinn erkennbar werden läßt, birgt es jedoch verschiedene Problemstellungen.
So ist das Interaktionsverhalten von Hyperlinks jenseits des bekannten a vollständig undefiniert und ihre überlappende Definition (wie durch die Einbettung des Elements line in das bereits mit einem Hyperlink versehene Element p illustriert) wirft die Frage nach der Behandlung durch das Anzeigegerät auf.
Auch führt dieser neu einzuführende Mechanismus notwendigerweise mit der aus verschiedenen Browsern bekannten Darstellungsweise von Hyperlinks durch blau gefärbten und unterstrichenen Text. So müßte das Beispieldokument vollständig gefärbt und unterstrichen dargestellt werden, was sich zweifellos als der Lesbarkeit nicht zuträglich erweisen würde.
Überdies entwickelt XHTML durch die diskutierte Vorgehensweise einen allgemein verwendbaren Referenzierungsmechanismus neu, der bereits durch den Standard XLink verfügbar ist. Vielmehr noch ergeben sich auf der zulässigen Kombination beider Verfahren weitere Problemstellungen.
Vor dem Hintergrund dieser Kritikpunkte ist nicht damit zu rechnen, daß die bestehende Form der erweiterten Hyperlinks in den letztendlich verabschliedeten Standard eingang finden wird.

XML über HTTP

Ähnlich der Übertragung von klassischem HTML über HTTP zur Anzeige in Web-Browsern kann auch reines XML „direkt“ per HTTP transportiert werden. Naturgemäß ist hierfür jedoch, anders als für (X)HTML, keine visuelle Repräsentation durch das empfangende Endgerät festgelegt.
Einige Hersteller bieten für natives XML strukturierte baumartige Darstellungen an. Der bekannteste Vertreter dieser Klasse dürfte Microsofts Internet Explorer sein, der ab Version 5.0 auch XML-Eingangsdaten unterstützt. Seine Browseransicht für das Beispieldokument ist im nachfolgenden Bildschirmabzug wiedergegeben.
Über Zuordnung von Präsentationsinformation durch XML Stylesheets (siehe Kapitel 2.4) kann XML direkt an einen

Web-Client gesendet werden. Zur Anzeige der Informationen wird im Browser ein *Stylesheet* herangezogen. Genau dieses Prinzip ist auch durch den Internet Explorer verwirklicht; dort wird auf jedem XML-Dokument, welches keine eigene Stylesheetquelle identifiziert, eine Vorgabetransformation durchgeführt, welche die abgebildete Baumstruktur erzeugt.
Andere Browser unterstützen dieses Verhalten nicht und präsentieren daher für XML-Dokumente ohne assoziiertes Stylesheet lediglich einen leeren Bildschirm.



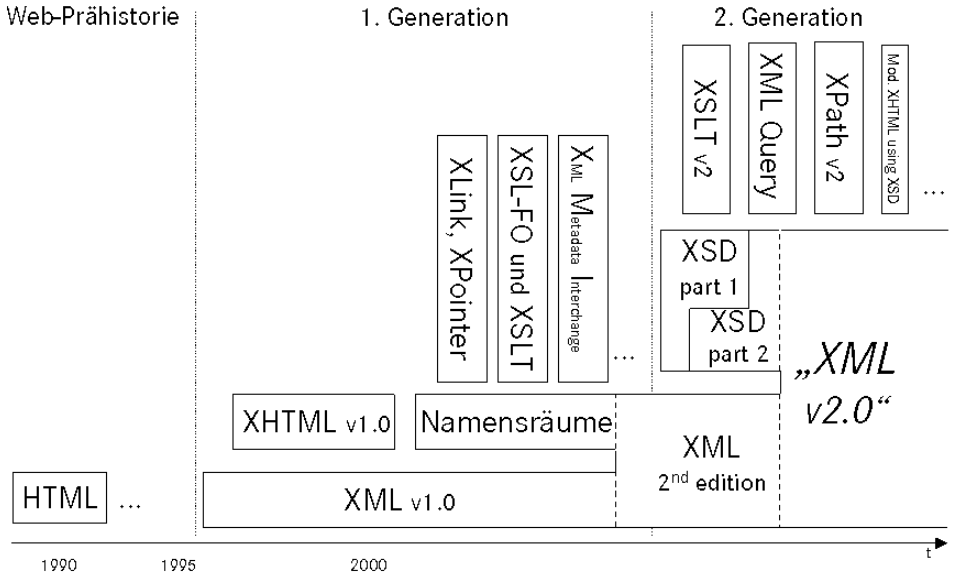

```
- <ProjektVerwaltung xsi:noNamespaceSchemaLocation="http://www.jeckle.de/vorlesung/xml/examples/projektverwaltung.xsd">
  - <Person PersID="Pers01" mitarbeitInProjekt="Prj01">
      <Vorname>Hans</Vorname>
      <Nachname>Hinterhuber</Nachname>
    </Person>
  - <Person PersID="Pers02" mitarbeitInProjekt="Prj02">
      <Vorname>Franz</Vorname>
      <Vorname>Xaver</Vorname>
      <Nachname>Oberm?ller</Nachname>
    - <Qualifikationsprofil>
        IT-Kompetenz verschiedene Betriebssysteme und
        <Leistungsstufe>professionelle</Leistungsstufe>
        <Qualifikation>Programmierung</Qualifikation>
        verschiedener Programmiersprachen
        <Qualifikation>Entwickler</Qualifikation>
        von 1988-1990
        <Qualifikation>Projektleiterfunktion</Qualifikation>
        von 1990-93 im X42-Projekt in Abteilung AB&C
      </Qualifikationsprofil>
    </Person>
  - <Person PersID="Pers03" mitarbeitInProjekt="Prj02">
      <Vorname>Fritz</Vorname>
      <Nachname>Meier</Nachname>
    </Person>
    <Projekt ID="Prj01" Projektleiter="Pers01" Mitarbeiter="Pers01"/>
    <Projekt ID="Prj02" Projektleiter="Pers02" Mitarbeiter="Pers03"/>
  </ProjektVerwaltung>
```

Web-Referenzen 10: Weiterführende Links

- Modularization of XHTML
- XHTML 1.0 in XML Schema
- XHTML v2.0

2 XML-Standards und -Anwendungen der zweiten Generation ...

Das nachfolgende Kapitel beleuchtet aufbauend auf den Grundlagen des ersten Abschnittes einige der sog. *XML Standards der zweiten Generation*.
Sie bilden zugleich als erste Anwendungen der eXtensible Markup Language auch Sprachen im engeren Sinne, da sie neben der Syntax auch Semantikdefinition, zumeist in Form eines Ausführungsmodells, umfassen.



Die Abbildung stellt die verschiedenen Generationen der XML-Standards in der Übersicht dar. Den Anfang -- augenzwinkernd als *Web-Prähistorie* bezeichnet -- bildet die bekannte Hypertextsprache HTML. Die erste Standardgeneration läßt sich an der Verabschiedung der XML-Recommendation im Februar 1998 festmachen. In der Folge wurden erste XML-Sprachen, wie XHTML, entwickelt. Wesentlich komplettiert wird diese erste Entwicklungsstufe durch die XML-Namensräume. Auf der Basis des Urstandards und dieser ersten Erweiterung, die selbst als XML-Sprache realisiert ist, entstanden die ersten weiter verbreiteten XML-Sprachen für produktive Zwecke. Beispielsweise die erweiterten Hyperlinks der Spezifikation XLink, aber auch die Stylesheetsprache XSL zur Erzeugung verschiedener Präsentationssichten auf Basis von XML-Dokumenten, sowie die Transformationssprache XSLT ! zur Umwandlung von XML-Dokumenten in verschiedene Ausgabeformate.
Mit der Integration der Namensräume in die XML-Spezifikation (XML 2nd edition) vollzieht sich bereits andeutungsweise der Übergang zur neuen konzeptionellen Basis der Metasprache XML. Am deutlichsten wird dieser Evolutionsschritt durch die Verabschiedung des XML-Schemastandards im Mai 2001 markiert. Er löst den bisherigen -- von SGML übernommenen -- Grammatikmechanismus der Document Type Definition ab, und legt die XML als selbstbeschreibende Metasprache an. Gemeinsam mit XML v1.0 2nd edition bildet diese W3C-Recommendation die neue technische Basis der weiteren Standardisierungsbemühungen. Die Graphik verwendet hierfür den teilweise anzutreffenden Begriff einer *XML v2.0*, der jedoch in dieser Form nicht durch das W3C geprägt wurde. Aufbauend auf diesem Fundament werden einige der bereits bestehenden XML-Co-Standards überarbeitet und auf die neuen Randbedingungen, wie XML-Schema, abgestimmt.

2.1 (Dokument-)Verknüpfungen: XML Links

Hyperlinking, als Möglichkeit der nichtlinearen wahlfreien Navigation in Dokumenten und zwischen beliebigen Ressourcen, ist in Zeiten des WWW allgegenwärtig. Zumeist werden die bekannten Sprungstellen im (X)HTML-Dokument farblich und zusätzlich durch Unterstreichung hervorgehoben dargestellt; ein simpler Mausklick verzweigt zur referenzierten Adresse oder aktiviert die hinterlegte Reaktion.
Bei genauerer Betrachtung läßt sich noch ein zweiter Verweismechanismus in (X)HTML identifizieren: die Referenzierung Dokument-externer Graphikdateien. Die (X)HTML-Datei enthält lediglich einen Verweis auf die URI der Graphik; die automatische Auflösung der Referenz, das Laden der Datei und die Integration in das (X)HTML-Dokument an der Stelle des `img`-Elements übernimmt der HTML-Agent (Browser oder sonstiges Anzeigegerät) ohne Interaktion durch den Anwender. Dieser Vorgang der *automatisierten einbettenden Referenzierung* wird *Transklusion* genannt. Der Begriff geht auf Ted Nelson und sein Xanadu-Projekt zurück. Der Terminus verbindet die beiden Worte *Transfer* und *Inklusion* um auf die beliebige physische Lokation der referenzierten Ressource und ihre transparente Einbindung in das Zieldokument gleichermaßen hinzuweisen.
Konzeptionell bilden beide Verweisarten unidirektional navigierbare (d.h. vom Quelldokument zur referenzierten Zielressource traversierbare) Verknüpfungen, die syntaktisch in das Quelldokument eingebunden werden. Das Verweisziel bleibt von der Bildung des Links unberührt.
So flexibel und naheliegend dieser Ansatz auf den ersten Blick scheinen mag, dennoch birgt er im praktischen Einsatz durchaus ernstzunehmende Unwägbarkeiten. Die bekannteste Ausprägung dürfte der HTTP-Fehler 404 -- *Not Found* -- sein. Er signalisiert der anfragenden Instanz, daß die angeforderte URI nicht gefunden werden konnte (spezifikationsgemäß möglich, wenn auch seltener genutzt, ist für diesen Fall auch Fehlercode 403 -- *Forbidden* -- zugelassen (siehe HTTP v1.0 Spezifikation IETF RFC 1945)).
Die Fehlersituation entsteht zumeist durch Referenzierung einer zwischenzeitlich gelöschten oder umgezogenen URI. Zwar läßt die HTTP v1.1 Spezifikation mit dem Fehlercode 410 -- *Gone* -- Rückschlüsse auf die frühere Gültigkeit der URI zu, jedoch schlägt auch in diesem Falle der Referenzierungsversuch fehl (siehe HTTP v1.1 Spezifikation IETF RFC 2068).
Die Grundidee des in (X)HTML verwirklichten Linkingmechanismus läßt keine Möglichkeit zur Vermeidung diese Situation zu, da die referenzierte (d.h. unabhängige) Instanz keinerlei Information über den gesetzten Verweis innerhalb des abhängigen Dokuments erhält.

Gemeinsames Kennzeichen der beiden Verweisarten ist die direkte explizite Einbettung der Links in das Quelldokument. Hierdurch werden Pflege des Dokumentinhaltes und der Verweise an derselben organisatorischen Stelle konzentriert, Änderungen an den gesetzten Verweisen ziehen in jedem Falle Änderungen am verweisenden Dokument nach sich.
Als Folge können Dokumente nicht durch Externe auf dem Wege der Annotation durch Verweise ergänzt werden.

Technisch erfordert dieses Vorgehen neben der verknüpften Quell- und Zielressource eine dritte Instanz zur unabhängigen Speicherung der Verweise.
Eine konkrete Umsetzung der freien Annotation beliebiger WWW-Seiten durch Dritte bot der, mittlerweile eingestellte, Dienst *Third Voice*.

Die klassischen HTML-Hyperlinks sind als Bestandteil der XML-Sprache XHTML durch Strukturen ihrer Grammatikdefinition in Form von Elementen und Attributen realisiert. Sie siedeln sich damit im Anwendungsbereich der XML an, und stehen daher nicht sprachunabhängig für alle XML-Sprachen zur Verfügung.
Durch die XML-Grundauffassung einer Welt vernetzter unabhängiger Dokumente ergibt sich die natürliche Notwendigkeit zur Schaffung eines Verweismechanismus, der für beliebige XML-Sprachen gleichermaßen universell eingesetzt werden kann. Dieser Anforderung soll die XML Linking Language (XLink) genügen.

Der Sprachstandard XLink behebt dieses Manko und schlägt für beliebige XML-Sprachen ein Vokabular mit zugehöriger Semantikdefinition zur Realisierung beliebiger Verweisstrukturen vor. Konzeptionell bildet XLink eine Obermenge des klassischen `href`-Elements.

Definition 12: XML Linking Language

Die XML Linking Language (XLink) definiert ein XML-basiertes Vokabular zur Darstellung allgemeiner Verweise in einem XML-Dokument.
Die Verwendung von XLink ist nicht an ein konkretes XML-Vokabular gebunden.
XLink-konforme Verweise können sowohl in einem Dokument ausgedrückt sein, als auch in einer externen Datenstruktur -- der sog. *Linkbase* -- verwaltet werden.

Erste Umsetzungen der Standardvorschlages existieren, neben einigen Spezialwerkzeugen, in den Web-Browsern Netscape (ab Version sechs) und dem Open-Source Projekt Mozilla.

Die XML Linking Spezifikation definiert in einem eigenen Namensraum eine Reihe von Attributen mit zulässigen Wertbelegungen, die für die Verwendung in eigenen XML-Sprachen zur Verfügung stehen.
Durch die Beschränkung auf Attributdefinitionen wird dem Anwender ein zusätzlicher Freiheitsgrad gegenüber der bisherigen (X)HTML-Verweismimik eröffnet. Während in Hypertextdokumenten Links ausschließlich durch Spezifikation des Attributs `href` im Ankerelement a definiert werden konnten, können die XLink-Attribute in beliebigen Elementen verwendet werden.

Im Einzelnen definiert die XLink-Spezifikation folgende Attribute und Wertbelegungen:

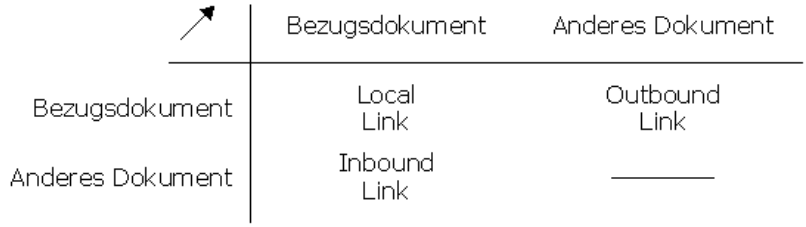
Tabelle 17: XLink-Attribute

Attribut	zulässige Belegung	Semantik
`type`	`simple, extended, locator, arc, resource, title`	Die verschiedenen durch die Spezifikation vorgegebenen XLink-Typen
`href`	Jede gültige URI gemäß IETF RFC 2396	Verweis auf die referenzierte Ressource
`role`	Jede gültige URI gemäß IETF RFC 2396	Verweis auf eine Ressource, welche eine nähere Beschreibung der referenzierten Ressource enthält; verwendet für `simple`, `extended`, `locator` oder `resource` typisierte Links
`arcrole`	Jede gültige URI gemäß IETF RFC 2396	Verweis auf eine Ressource, verwendet für `simple` oder `arc` typisierte Links
`title`	Freier Text	Beschreibt die Bedeutung des Verweises in lesbarem Format
`show`	`new, replace, embed, other, none`	Verhalten des Verweises bei seiner Aktivierung. `new` öffnet die referenzierte Resource in einem neuen Fenster (entspricht damit der (X)HTML-Definition `<a href="http://www.jeckle.de" target="_blank">...</a>`), `replace` ersetzt den aktuellen Fensterinhalt (entspricht damit der (X)HTML-Definition `<a href="http://www.jeckle.de" target="_self">..</a>`), `embed` ersetzt den Verweis durch den Inhalt der referenzierten Resource (entspricht damit der (X)HTML-Definition `<img src="http://www.w3.org/Icons/w3c_home.gif" />`), `other` erlaubt die Definition beliebiger eigener Aktionen durch Markup. `none` definiert, daß durch den Client keine Standardaktion bei Linkaktivierung auszuführen ist. Die hier gegebenen Interpretationen der Aktionen sind stark an denen eines Web-Browsers orientiert, und können daher für andere Endgeräte beliebig variieren bzw. unter Umständen garnicht angeboten werden.

`actuate`	`onLoad, onRequest, other, none`	`onLoad` bewirkt die automatische Aktivierung des XLinks, das Verhalten entspricht damit dem des (X)HTML-Elements `img`. Wird `onRequest` spezifiziert, so kann der Verweis durch den Anwender interaktiv wahlfrei aktiviert werden. Das Verhalten entspricht hierbei den bekannten Hypertextlinks. `other` definiert durch weiteres Markup spezifiziertes freies Verhalten, welches durch das Endgerät umgesetzt wird. Bei `none` kann durch den interpretierenden Client eine im XML-Sinne proprietäre, d.h. nicht durch Markup spezifizierte, Verhaltensdefinition angegeben werden.
`label`	Jeder zulässige NCName.	Definiert textuelle Identifikation des XLinks
`from`	Jeder zulässige NCName, der angegebene Wert muß dem eines existierenden `label`-Attributs entsprechen.	Verweisquelle eines erweiterten XLinks
`to`	Jeder zulässige NCName, der angegebene Wert muß dem eines existierenden `label`-Attributs entsprechen.	Verweisziel eines erweiterten XLinks

---

Je nach ihrer Belegung definierten die XLink-Attribute drei verschiedene Linktypen, die relativ zu einem Bezugsdokument benannt werden.
Verweist das `href`-Attribtut eines XLinks lediglich auf eine Ressource innerhalb des Bezugsdokuments, dann handelt es sich um einen *Local Link*. Wird auf eine anderes Dokument verwiesen, so spricht man von einem *Outbound Link*. Umgekehrt, d.h. im Falle eins externen Verweises auf das Bezugsdokument, so wird dieser als *Inbound Link* bezeichnet. Inbound und Outbound Links entsprechen sich daher, jeweils mit geändertem Bezugspunkt.
Abbildung 27 stellt die Typen in der Übersicht zusammen.



Als besondere Form existiert mit dem Linktyp des *External Links* ein Verweistyp, der sich nicht in obiges Schema fügt. In seinem Falle wird der gesamte XLink in einer externen Ressource gehalten, die selbst nicht notwendigerweise in XML-Dokument sein muß.
Dieses externe Linkverzeichnis wird *Link Base* genannt, es ist typischerweise durch eine Datenbank umgesetzt.

Beispiel 51 zeigt die Definitionen der einfachsten Form eines XLinks, den sog. *simple* Link. Benannt ist diese Darstellung nach dem Wert des `type`-Attributs.
Im Beispiel wird durch das Element `myElementA` ein Verweis definiert, dessen Verhalten den (X)HTML-Hyperlinks entspricht. Zunächst ist zur Identifikation der XLink-Attribute der durch die Spezifikation vorgegebene Namensraum `http://www.w3.org/1999/xlink` festzulegen; daher wird dieser an das Präfix `xlink` gebunden.
Die Definition als Vorgabenamensraum ist für diesen Anwendungsfall nicht möglich, da sich Namensräume vorgabegemäß nur auf Elemente auswirken.
Das Verweisziel wird in Form des `href`-Attributs (im XLink-Namensraum) angegeben.

Beispiel 51: Ein simple XLink

```
(1)<myElementA
(2)     xmlns:xlink="http://www.w3.org/1999/xlink"
(3)     xlink:type="simple"
(4)     xlink:href="http://www.jeckle.de">...</myElementA>
```

Der transklutorischen (d.h. die referenzierte Ressource einbettenden und automatisch aktivierten) Graphik-Referenz des `img`-Elements aus (X)HTML entspricht folgendes Konstrukt:

Beispiel 52: Ein transklutorischer Link

```
(1)<myElementB
(2)     xmlns:xlink="http://www.w3.org/1999/xlink"
(3)     xlink:type="simple"
(4)     xlink:href="http://www.w3.org/Icons/w3c_home.gif"
(5)     xlink:actuate="onLoad"
(6)     xlink:show="embed">...</myElementB>
```

Durch die zusätzlichen angebbaren Attribute läßt sich die Semantik eines Links spezifizieren. Dies stellt eine erste Erweiterung der Mächtigkeit gegenüber den bisher im Vergleich zu (X)HTML diskutierten Verweisen dar.
So erlaubt das `title`-Attribut die genauere Charakterisierung eines Links durch beliebigen anwenderdefinierten Freitext.
Am Beispiel der Projektverwaltung: Das nachfolgende Codefragment realisiert die Beziehung zwischen `Projekt` und seinen `Mitarbeitern`, die in einer anderen Datei abgespeichert sind, durch einen XLink.

Beispiel 53: Nutzung des title-Attributs

```
(1)<Projektverwaltung
(2)     xmlns:xlink="http://www.w3.org/1999/xlink">
(3)     ...
(4)     <Projekt
(5)             xlink:type="simple"
(6)             xlink:href="../xyz.xml"
(7)             xlink:title="Liste der Projektmitarbeiter">...</Projekt>
(8)      ...
(9)</Projektverwaltung>
```

Definition 13: Simpler XLink

Ein simpler XLink ist eine unidirektionale Verbindung zwischen zwei Ressourcen, wobei eine Ressource als Quelle und die andere als Ziel des Links interpretiert wird.

Während die simplen XLinks größtenteils in Analogie oder behutsamer Erweiterung zum entsprechenden (X)HTML-Konzept aufgebaut sind, führen die *extended links* bisher nicht realisierbare Möglichkeiten ein.
Wiederum am Beispiel der Projektverwaltung: Zwar läßt das vorhergehende Codefragment die Referenzierung der Projektmitarbeiter zu, jedoch kann innerhalb des `Projekt`-Elements kein weiterer XLink als Verweis auf die Projektleiter gesetzt werden, da sonst mehrfach dasselbe Attribut auftreten würde.
Das zugrundeliegende Problem, nämlich der Wunsch durch einen Link mehrere Ressourcen gebündelt referenzieren zu können, tritt in praktischen Anwendungen häufig auf. Beispielsweise beim Verweis auf mehrere Versionen eines Dokuments, alternativen Downloadservern oder Bildern verschiedener Auflösung.
Zu Realisierung erlaubt XLink die Bündelung von verschiedenen Verweisen zu einem einzigen erweiterten XLink. Nachfolgend ein Codeausschnitt, der dies für die Projektverwaltung zeigt.

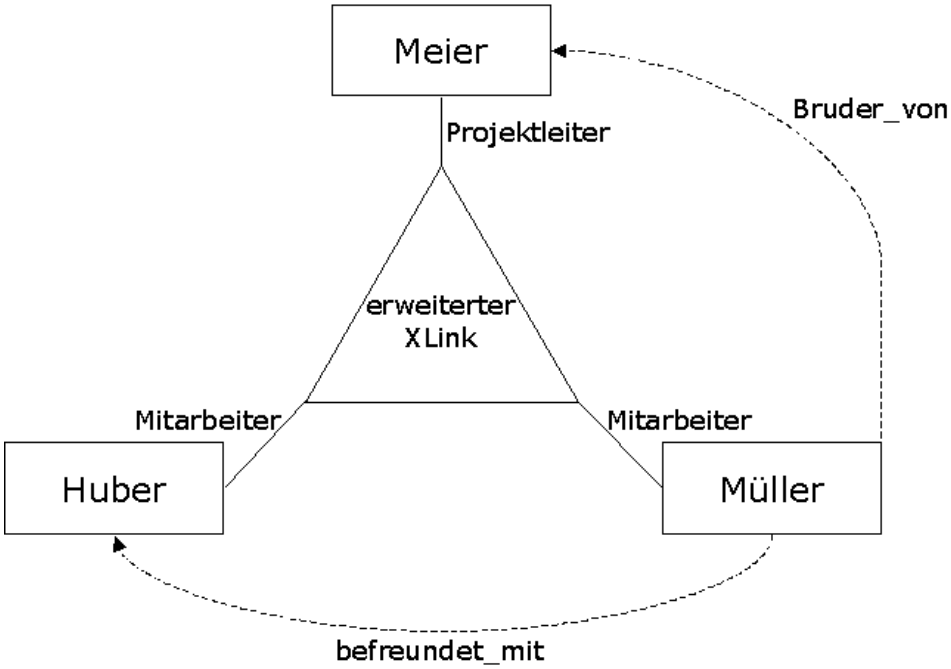
Definition 14: Extended XLink

Ein erweiterter XLink kann gleichzeitig auf mehr als eine Ressource verweisen.

Beispiel 54: Ein Verweis auf mehrere Ressourcen

```
(1)<Projekt
(2)     xmlns:xlink="http://www.w3.org/1999/xlink"
(3)     xlink:type="extended">
(4)     <Person
(5)             xlink:type="locator"
(6)             xlink:href="urn:names:meier"
(7)             xlink:label="ID001"
(8)             xlink:title="Projektleiter"
(9)             xlink:role="http://www.example.com/contracts/projectLeader"/>
(10)    <Person
(11)            xlink:type="locator"
(12)            xlink:href="urn:names:huber"
(13)            xlink:label="ID002"
(14)            xlink:title="Mitarbeiter"
(15)            xlink:role="http://www.example.com/contracts/employee"/>
(16)    <Person
(17)            xlink:type="locator"
(18)            xlink:href="urn:names:müller"
(19)            xlink:label="ID003"
(20)            xlink:title="Mitarbeiter"
(21)            xlink:role="http://www.example.com/contracts/employee"/>
(22)    </Projekt>
```

Das Beispiel definiert einen dreigliedrigen *extended link*. Zur Realisierung ist zunächst das `type`-Attribut des die Einzellinks umschließenden Elements auf `extended` zu setzen. Im Rumpf dieses Elements folgen die als `locator` typisierten Verweise auf die verschiedenen Ressourcen. Die von den simple-Links bekannten XLink-Attribute `href` zur URI-basierten Referenzierung der Ressource, `title` zur natürlichsprachlichen Beschreibung des Verweises, sowie `role` zur Referenzierung einer beschreibenden Resource, können mit gleicher Semantik weiterverwendet werden. Zusätzlich nutzt das Beispiel die Möglichkeit der eindeutigen Identifikation der Verweisteile innerhalb des extended Links durch das `label`-Attribut.



Generell trifft das Konstrukt des *erweiterten Links* keine Vorgabe über mögliche Navigationsrichtungen zwischen allen im Rumpf des Linkelements aufgeführten Ressourcen.
Die Graphik stellt die referenzierten externen Ressourcen *Meier*, *Huber* und *Müller* als Rechteckte die durch den als Dreieck dargestellten erweiterten Link verbunden sind dar.
Es kann jedoch der Wunsch bestehen Beziehungen zwischen den durch einen erweiterten XLink verbundenen Ressourcen zu definieren. Hierfür gibt der Standard das `arc`-Attribut vor.

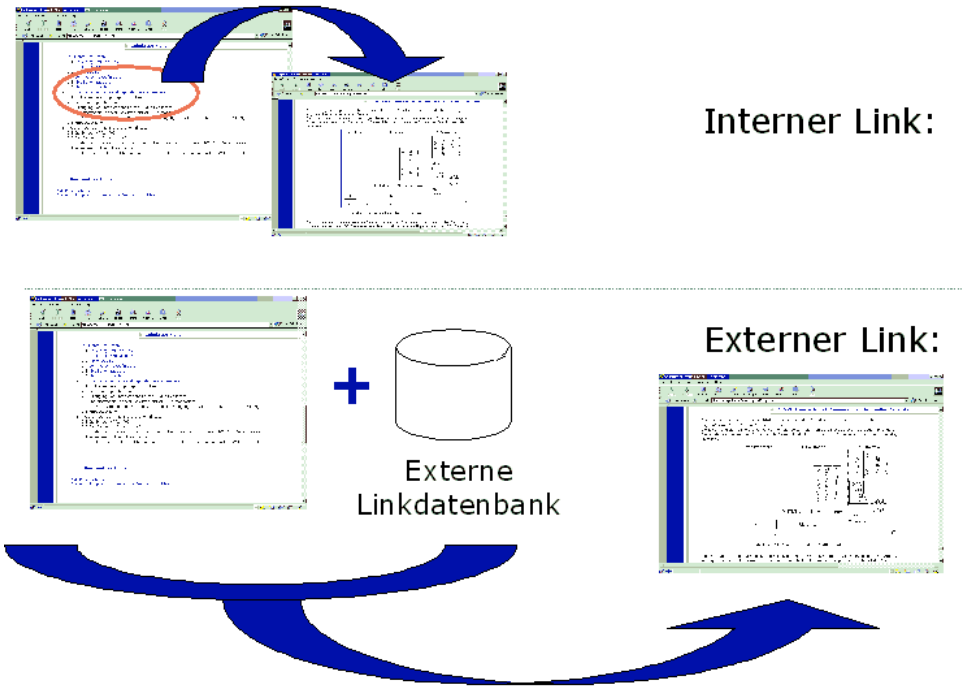
Zusätzlich zu diesem Attribut kann die Traversierungsrichtung des Links durch die Attribute `from` und `to` festgelegt werden. Die Graphik stellt diese Beziehungen als unterbrochene gerichtete Kanten dar.
Implizit kann daher die Existenz dieses Attribut-Tripels für simple XLinks angenommen werden.
Für das vorhergehende Beispiel ist die Navigationsdefinition dergestalt gewählt, daß eine Beziehung zwischen der Ressource *Müller* und *Huber* besteht. Diese Beziehung wird durch das Element `befreundet_mit` definiert. Zusätzlich etabliert das Element `Bruder_von` durch das `arc`-Attribut eine Beziehung zwischen *Müller* und *Meier*.

Beispiel 55: Eingeschränkte Navigation innerhalb eines extended XLinks

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<Projekt
(3)   xmlns="http://www.jeckle.de/vorlesung/xml"
(4)     xmlns:xlink="http://www.w3.org/1999/xlink"
(5)     xlink:type="extended">
(6)     <Person
(7)             xlink:type="locator"
(8)             xlink:href="urn:names:meier"
(9)             xlink:label="ID001"
(10)            xlink:title="Projektleiter"
(11)            xlink:role="http://www.example.com/contracts/projectLeader" /
>
(12)    <Person
(13)            xlink:type="locator"
(14)            xlink:href="urn:names:huber"
(15)            xlink:label="ID002"
(16)            xlink:title="Mitarbeiter"
(17)            xlink:role="http://www.example.com/contracts/employee" />
(18)    <Person
(19)            xlink:type="locator"
(20)            xlink:href="urn:names:müller"
(21)            xlink:label="ID003"
(22)            xlink:title="Mitarbeiter"
(23)            xlink:role="http://www.example.com/contracts/employee" />
(24)    <befreundet_mit
(25)            xlink:type="arc"
(26)            xlink:from="ID003"
(27)            xlink:to="ID002"/>
(28)    <Bruder_von
(29)            xlink:type="arc"
(30)            xlink:from="ID003"
(31)            xlink:to="ID001"/>
(32)</Projekt>
```

Das Beispiel verwendet zur Referenzierung der einzelnen Ressourcen das XLink-Attribut `label`.
Abschließend bleibt anzumerken, daß die `arc`-Typisierung lediglich die positive Definition explizit erlaubter Traversierungspfade erlaubt. Eine Einschränkung der Vorgabepfade ist nicht möglich.

Neben den bisher diskutierten Verweisen, bei denen es sich ausschließlich um *Outbound Links* handelte, läßt XLink auch die Auslagerung der Referenzierungen in eine sog. *Link Database* (kurz: *Linkbase*) zu. Ein solcher Datenspeicher enthält eine Menge von Zwei-Tupeln , welche Quelle und Ziel eines Verweises wiedergeben. Eine geeignete Applikation muß zur Realisierung der Verzeigerung immer eine Anfrage an diese Datenbank absetzen, und das Quell-Dokument unter Umständen geeignet aufbereiten, um dem Anwender die Möglichkeit zur Interaktion zu geben.
Das Schema der Abbildung stellt die beiden Verweismechanismen in der Zusammenschau dar. Im oberen Bereich ist ein *inline link* angedeutet, der im Quelldokument die zur Lokalisierung und Anzeige der referenzierten Ressource notwendigen Daten enthält. Im unteren Bereich ist das Konzept einer externen Linkdatenbank, die vor Anzeige des durch sie annotierten Quelldokuments zu konsultieren ist.



Für das Anwendungsszenario der externen Linkdatenbank stellt sich die Frage nach dem Aussehen der in der Datenbank abgespeicherten Zwei-Tupel. Während die Notation des Verweisziels durch die URI (IETF RFC 2396) vorgegeben ist, eröffnet sich die Frage nach der Lokalisierung des Verweises selbst innerhalb des Quelldokuments. Unter Rückgriff auf (X)HTML ließen sich zwar durch die explizite Definition von Textankern (a-Element mit `name`-Attribut) beliebige Stellen als Aufsetzpunkt von extern verwalteten Verweisen definieren, jedoch würde dies erhebliche Modifikationen am Quelldokument nach sich ziehen, die bei Dokumentänderungen ihrerseits wiederum Wartungsarbeiten erfordern würden.
Idealerweise sollte das Quelldokument durch die Definition extern abgelegter Verweise nicht berührt werden. Im Falle eines freien Annotationsdienstes, der durch Dritte genutzt werden kann, stellt dies sogar eine Grundforderung dar. Dieser Anwendungsfall bildet -- neben anderen -- ein Anwendungsszenario, welches eine Möglichkeit zur flexiblen Lokalisierung von Dokumentteilen erfordert. Hierfür wurde durch das W3C die im nachfolgenden Abschnitt vorgestellte *XML Path Language* verabschiedet.

XML Base

Um die Möglichkeiten der klassischen Hypertextverknüpfungen aus HTML auch für XML vollständig nachzubilden fehlt XLink in der Grundform die Unterstützung relativer Verweise.
Diese aus (X)HTML bekannte Kurzschreibweise dient zur Adressierung eines Teils einer durch URI identifizierten Ressource. In Hypertextdokumenten werden solche Verweise häufig zur direkten Navigation zu Teilkapitel oder Zwischenüberschriften verwendet.

Zur Realisierung müssen die Sprungziele im Dokument durch den Autor festgelegt werden. (X)HTML bietet hierfür das `id`-Attribut an. In Verbindung mit dem Anker-Element erlaubt es die freie Definition von Sprungzielen innerhalb beliebiger XHTML-Dokumente.
Syntaktisch kann dabei der durch das `#`-Symbol vom Ankernamen separierte Identifikator einer URI nachgestellt werden um auf ein Sprungziel innerhalb der durch die URI bezeichneten Ressource zu verweisen. Abkürzend kann -- durch Weglassung der URI -- auf Anker des aktuellen Dokuments verwiesen werden.

Das XHTML-Dokument des Beispiels 56 zeigt die beiden Nutzungsvarianten relativer Verweise.
Die mit `Zueignung`, `Vorspiel` und `Prolog` benannten Anker adressieren die im Beispieldokument plazierten Ressourcen gleichen Namens (a-Elemente mit gesetztem `id`-Attribut).
Die übrigen Verweise (etwa `Teil1.html#Nacht`) nehmen Bezug auf benannte Ressourcen anderer Quellen (etwa innerhalb des Dokuments `Teil1.html`). Diese Quellen müssen zur Gewährleistung der Auflösbarkeit die notwendigen Anker-Elemente definieren.

Beispiel 56: Hypertext-Dokument mit HTML-konformen Verweisen

```
(1)<?xml version="1.0" encoding="ISO-8859-1"?>
(2)<html xmlns="http://www.w3.org/1999/xhtml">
(3)   <head>
(4)      <title>J. W. von Goethe: FAUST</title>
(5)   </head>
(6)
(7)   <body>
(8)      <h1>Inhaltsverzeichnis</h1>
(9)      <ul>
(10)         <li><a href="#Zueignung">Zueignung</a></li>
(11)         <li><a href="#Vorspiel">Vorspiel auf dem Theater</a></li>
(12)         <li><a href="#Prolog">Prolog im Himmel</a></li>
(13)      </ul>
(14)
(15)      <ol>
(16)         <li>
(17)            <h2>Der Tragödie erster Teil</h2>
(18)            <ul>
(19)               <li><a href="http://www.example.com/Teil1/Akt1.html#Nacht">Nacht</a></li>
(20)               <li><a href="http://www.example.com/Teil1/Akt1.html#Tor">Vor dem Tor</a></li>
(21)               <li><a href="http://www.example.com/Teil1/Akt1.
html#Studierzimmer">Studierzimmer</a></li>
(22)               <li><p>...</p></li>
(23)            </ul>
(24)         </li>
(25)
(26)         <li>
(27)            <h2>Der Tragödie zweiter Teil</h2>
(28)            <ul>
(29)               <li><a href="http://www.example.com/Teil2/Akt1.html#AnmGegend">Anmutige Gegend</
a></li>
(30)               <li><a href="http://www.example.com/Teil2/Akt1.html#gotZimmer">Hochgewölbtes
enges gotisches Zimmer</a></li>
(31)               <li><a href="http://www.example.com/Teil2/Akt1.html#Palast">Vor dem Palaste des
Menelas zu Sparta</a></li>
(32)               <li>...</li>
(33)            </ul>
(34)         </li>
(35)      </ol>
(36)
(37)      <hr />
(38)
(39)      <h1><a id="Zueignung">Zueignung</a></h1>
(40)      <p>Ihr naht euch wieder, ...</p>
(41)
(42)      <h1><a id="Vorspiel">Vorspiel auf dem Theater</a></h1>
(43)      <p>Direktor. Ihr beiden, die ihr mir so oft, ...</p>
(44)
(45)      <h1><a id="Prolog">Prolog im Himmel</a></h1>
(46)      <p>Raphael. Die Sonne t?nach alter Weise ...</p>
(47)   </body>
(48)</html>
```

Download des Beispiels

Während innerhalb des (X)HTML-Dokuments relative Verweise durch die URI des umgebenden Dokumentes ergänzt werden können ist dies für XML-Dokumente im allgemeinen Falle nicht anzunehmen. Aus diesem Grunde definiert der W3C-Standard XML Base einen einfachen Mechanismus zur Nachbildung der in (X)HTML verwirklichten Mimik.

Zur Realisierung relativer Verweise definiert XML Base ein aus genau einem Attribut bestehendes XML-Vokabular. Dieses Attribut -- `xml:base` -- dient zur Aufnahme des URI-Anteils. Alle referenzierenden Informationselemente, beispielsweise `href`, die innerhalb des mit diesem Attribut versehenen Element oder innerhalb seiner Kindelemente auftreten werden relativ zur im `xml:base`-Attribut angegebenen URI interpretiert. Intuitiv wird diese dynamisch vor dem Fragmentidentifikator plaziert und mit ihm zu einer neuen URI konkateniert.
Dieser Vorgang kann auch über mehrere Stufen des XML-Dokumentbaumes vollzogen werden. Hierzu werden die Inhalte aller `xml:base`-Attribute die sich in hierarchisch übergeordneten Elementen des referenzierenden Knotens befinden zusammengefügt.

Das Beispiel der Abbildung 57 formuliert den XHTML-Code des vorangegangenen Beispiels XML Base-konform um. Daher wird zunächst der URI-Anteil der für alle Verweise gleichermaßen gilt auf einem hierarchisch höherstehenden gemeinsamen Element als Belegung des Attributs `xml:base` definiert. Im Beispiel wird daher die Basis-URI `http://www.example.com/` im entsprechenden Attribut des Wurzelelements `html` plaziert.
Die Dokumentinternen Verweise (sie folgen auf den Text Inhaltsverzeichnis) ändern sich daher nicht. In diesem Falle expliziert `xml:base` lediglich die im XHTML durch seine physische Lokation implizit gegebene Information.
Für die Verweise des Dokumentes `Akt1.html` im Pfad `Teil1` hingegen tritt eine Veränderung des durch `href` formulierten Verweises ein. Denn auch in diesem Falle kann der gemeinsame Anteil wieder in das XML-Basis-Attribut ausgelagert werden. In der Konsequenz ergibt sich die vollständige URI des Verweises `Nacht` wieder als `http://www.example.com/Teil1/Akt1.html#Nacht`, die durch Hierarchie-konforme Hintereinanderreichung der `xml:base`-Inhalte entsteht.

Für die Verweise in das Dokument `Akt1.html` im Pfad `Teil2` gilt dasselbe. Auch hier wird die zu dereferenzierende URI durch Konkatenation der XML-Basis-Attribute gebildet.
Durch die Organisation dieses `xml:base`-Attributs auf derselben Hierarchieebene wie das zuvor diskutierte ergibt sich auch in diesem Anwendungsfalle die korrekte URI als `http://www.example.com/Teil2/Akt1.html`.

Beispiel 57: Hypertext-Dokument mit XML-Base-konformen Verweisen

```
(1)<?xml version="1.0" encoding="ISO-8859-1"?>
(2)<html xmlns="http://www.w3.org/1999/xhtml"
(3)        xmlns:xlink="http://www.w3.org/1999/xlink"
(4)        xmlns:book="http://www.example.org/books"
(5)      xml:base="http://www.example.com/">
(6)   <head>
(7)      <title>J. W. von Goethe: FAUST</title>
(8)   </head>
(9)
(10)   <body>
(11)      <h1>Inhaltsverzeichnis</h1>
(12)      <ul>
(13)         <li><book:chapter xlink:type="simple" id="zueignung">Zueignung</book:chapter></li>
(14)         <li><book:chapter xlink:type="simple" id="vat">Vorspiel auf dem Theater</book:
chapter></li>
(15)         <li><book:chapter xlink:type="simple" id="prolog">Prolog im Himmel</book:chapter></li>
(16)      </ul>
(17)
(18)      <ol>
(19)         <li>
(20)            <h2>Der Tragödie erster Teil</h2>
(21)            <ul xml:base="/Teil1/Akt1.html">
(22)               <li><book:chapter xlink:type="simple" id="nacht">Nacht</book:chapter></li>
(23)               <li><book:chapter xlink:type="simple" id="vdTor">Vor dem Tor</book:chapter></li>
(24)               <li><book:chapter xlink:type="simple" id="studZi">Studierzimmer</book:chapter></
li>
(25)               <li>...</li>
(26)            </ul>
(27)         </li>
(28)
(29)         <li>
(30)            <h2>Der Tragödie zweiter Teil</h2>
(31)            <ul xml:base="/Teil2/Akt1.html">
(32)               <li><book:chapter xlink:type="simple" id="anmGegend">Anmutige Gegend</book:
chapter></li>
(33)               <li><book:chapter xlink:type="simple" id="gotZi">Hochgewölbtes enges gotisches
Zimmer</book:chapter></li>
(34)               <li><book:chapter xlink:type="simple" id="palast">Vor dem Palaste des Menelas zu
Sparta</book:chapter></li>
(35)               <li>...</li>
(36)            </ul>
(37)         </li>
(38)      </ol>
(39)   </body>
(40)</html>
```

Download des Beispiels

Web-Referenzen 11: Weiterführende Links

- XLink Spezifikation
- XLink Implementierungen
- Fabio Arciniegas A.: *XLink: An Introductory Example*
- Fabio Arciniegas A.: *What is XLink?*
- XML Base Spezifikation

2.2 Die Lokatorsprache XPath

Zur Extraktion beliebiger Teile eines wohl-geformten XML-Dokuments verabschiedete das W3C 1999 die Sprache *XPath*. Sie bildet eine pfadorientierte *Lokatorsprache*, die das Auffinden von Dokumentteilen (einzelnen Elementen, Attributen, etc.) durch Pfadausdrücke, die sich an der Struktur des XML-Dokuments orientieren, gestattet.
Die Grenze zwischen Lokatorsprache und „echter“ Anfragesprache wie SQL sind fließend. Zwei Unterscheidungsmerkmale sollen jedoch hervorgehoben werden: XPath wird im üblichen Anwendungsfall nicht interaktiv oder in eine Programmiersprache als Wirtssprache eingebettet verwendet, sondern wurde (zunächst) nur für die Nutzung in Kombination mit der Transformationssprache *XSLT* und den erweiterten Verweisen der Sprache *XPointer* konzipiert. Zum zweiten fehlt XPath die üblicherweise mit dem reinen Anfrageteil verwobene Manipulationssprache zur Änderung bereits bestehender Daten; XPath ist allein für den lesenden Zugriff auf XML-Dokumente ausgelegt.
Hinweis: XPath unterscheidet XML-üblich zwischen Groß- und Kleinschreibung. Daher sind Element- und Attributnamen unbedingt in der im Dokument gewählten Schreibweise anzugeben.

Lokalisierungspfade:

Lokalisierungspfade dienen der abstrakten Beschreibung einer Menge von Informationsknoten innerhalb eines Dokuments.
Die einfachste Form eines Lokalisierungspfades beschreibt der *Wurzellokalisierungpfad* (*root location path*), ausgedrückt durch „/“. Er liefert für jedes XML-Dokument den Wurzelknoten. Dieser ist nicht identisch mit dem Wurzelelement eines XML-Dokuments! Der (unbenannte) Wurzelknoten entspricht dem Document Information Item des Information Sets, während das erste benannte Element des Dokuments durch ein Element Information Item dargestellt wird.

Die Navigation zu den einzelnen Elementknoten, oder Knotenmengen, wird durch einen Pfadausdruck realisiert. Die explizite Variante erlaubt die Angabe aller zu traversierenden Knoten bis hin zu den zu extrahierenden. Hierzu werden die Knoten, von der Wurzel absteigend durch „/“-Symbole separiert, notiert. Wegen der Korrespondenz der voneinander abgetrennten Knotennamen und den Baumstufen, werden diese auch als *Lokalisierungsschritte* bezeichnet. Als weitere sprachliche Analoge spiegelt der XPath-Ausdruck, von links nach rechts gelesen, auch die Schritte -- ausgehend vom Wurzelelement des Dokuments -- zur Lokalisierung der gesuchten Knotenmenge wieder. Das Beispiel zeigt eine solche Definition am Beispiel der Projektverwaltung.
Anmerkung: Das Resultat ist in XML-Notation dargestellt, obwohl genaugenommen eine Knotenmenge des Information Sets als Resultat zurückgeliefert wird. Die gewählte XML-Darstellung ist hierbei nur eine der möglichen Varianten zur Ergebnispräsentation.

Beispiel 58: XPath-Ausdruck zur Lokalisierung aller Vornamen

XPath-Ausdruck: `/ProjektVerwaltung/Person/Vorname`
Ergebnis:
```
<Vorname>Hans</Vorname>,
<Vorname>Franz</Vorname>,
<Vorname>Xaver</Vorname>,
<Vorname>Fritz</Vorname>
```

Die Einzelknoten werden entsprechend ihrer Auftrittsreihenfolge im Quelldokument (sog. *document order*) zurückgegeben.

Die expliziten Pfadausdrücke lassen sich in beliebiger Länge fortsetzen, jedoch zeigen sie fundamentale Schwächen in Puncto Flexibilität. Wie im Beispiel der XHTML-Verwendung innerhalb eines eigenen XML-Dokuments gesehen, kann Information desselben Typs (d.h. umschlossen durch denselben Tag) verschiedene Elternknoten besitzen. So im Beispiel, dort ist die `Qualifikation` auf derselben Baumstufe sowohl unterhalb des Elternelements `em` als auch `u` anzutreffen.
Als Lösung erlaubt XPath die Nutzung von Platzhaltern statt der expliziten Elementnamen innerhalb eines Lokalisierungsschrittes. In der Folge entstehen freie Lokalisierungsschritte, die alle Kindknoten einer im direkt vorhergehenden Lokalisierungsschritt selektierten Knotenmenge adressieren.
Der nachfolgende XPath-Ausdruck zeigt dies am Beispiel des `Qualifikationsprofils`.

Beispiel 59: Platzhalter in Lokalisierungsschritten

XPath-Ausdruck: `/ProjektVerwaltung/Person/Qualifikationsprofil/*/Qualifikation`
Ergebnis:
```
<Qualifikation>Programmierung</Qualifikation>
<Qualifikation>Projektleiterfunktion</Qualifikation>
```

Der Pfadausdruck liefert die beiden Kindelemente `Qualifikation` -- unabhängig von der Benennung des Elternknotens -- die direkt unterhalb des Knotens `Qualifikationsprofil` angeordnet sind.
Allerdings enthält die Ausgabe nicht alle Knoten des Typs `Qualifikation`. Der gegebene Pfadausdruck gestattet lediglich das Überspringen einer Hierarchieebene. Daher wird der hierarchisch tieferstehende `Qualifikations`-Knoten mit Inhalt `Entwickler` nicht lokalisiert. Die (zunächst naheliegende) Lösung den Pfadausdruck zu `/ProjektVerwaltung/Person/Qualifikationsprofil/*/*/Qualifikation` zu erweitern liefert nicht das gewünschte Resultat aller `Qualifikations`-Knoten, sondern ausschließlich den zuvor nicht lokalisierbaren, da der modifizierte Ausdruck nun zwingend zwei freie Lokalisierungsschritte vorsieht.
Zur Variierung der Tiefe der freien Schritte sieht XPath die Schreibweise „//“ vor. Sie erlaubt die Lokalisierung der Kindknoten auf einer beliebigen Hierarchiestufe.

Definition 15: Lokalisierungsschritt

Ein Lokalisierungsschritt setzt sich aus dem Namen der Achse gefolgt von zwei Doppelpunkten und einem *Knotentest*, optional ergänzt um ein auszuwertendes Prädikat, zusammen.
Wird keine Achse spezifiziert, so gilt vorgabegemäß die Achse `child`.
Ein *Knotentest* ist syntaktisch ein `QName`, der genau dann erfüllt ist, wenn der Knotenname mit dem Namen des Knotentests übereinstimmt.
Das *Prädikat* filtert die Ergebnismenge hinsichtlich verschiedener Charakteristika wie Existenz von Kindknoten oder Attributen, Position in der Ergebnismenge, etc.

Das Beispiel zeigt die korrekte XPath-Formulierung zur Lokation aller `Qualifikations`-Knoten:

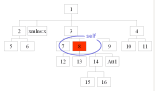
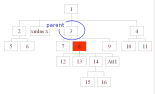
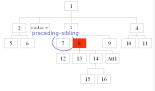
Beispiel 60: Hierarchieunabhängige Knoten-Lokalisierung

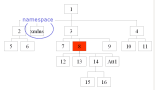
XPath-Ausdruck: `/ProjektVerwaltung/Person/Qualifikationsprofil//Qualifikation`
Ergebnis:
```
<Qualifikation>Programmierung</Qualifikation>

<Qualifikation>Entwickler</Qualifikation>

<Qualifikation>Projektleiterfunktion</Qualifikation>
```

Durch die abkürzende Schreibweise „//“ entsteht ein Muster zur Selektion aller nachfolgenden Knoten. In Verallgemeinerung dieses Konzepts bietet XPath sog. *Achsen* an, um relativ zum aktuellen Knoten beliebige Teilbäume zu lokalisieren.
Die Abbildung zeigt die verschiedenen durch Achsen zugänglichen Knotenmengen relativ zum rot hervorgehobenen aktuellen Knoten.

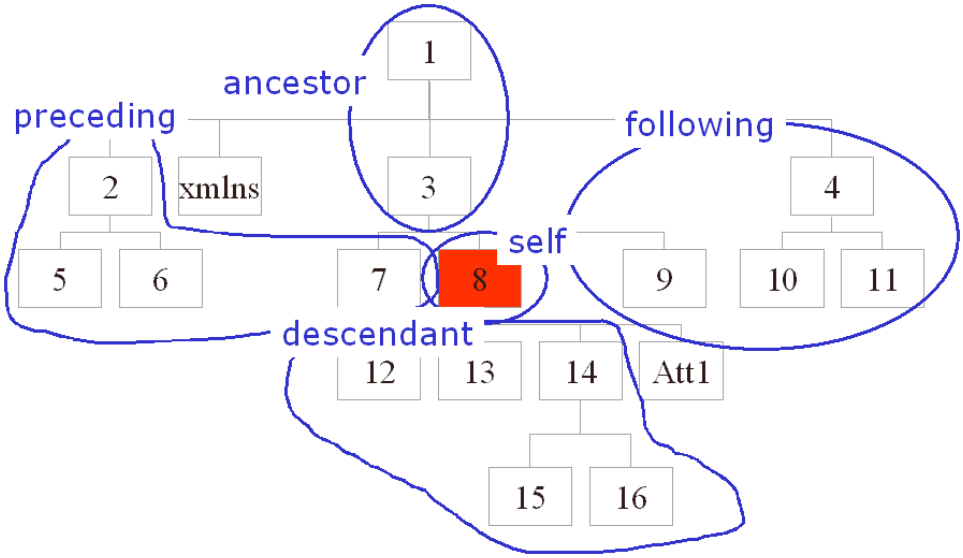
Download der XML-Datei mit dem Beispiel der Graphik

Tabelle 18: XPath-Achsen und ihre Bedeutung

Achse	Semantik	Im Beispiel selektierte Knoten	Graphik
`self`	Lokalisiert den aktuellen Knoten Als abkürzende Schreibweise kann der Punkt „.“ verwendet werden.	**XPath-Ausdruck:** `/node1/node3/node8/self::node8` **Ergebnisknotenmenge:** {8}	
`child`	Lokalisiert die (direkten) Kindknoten des aktuellen Knotens	**XPath-Ausdruck:** `/node1/node3/node8/child::*` **Ergebnisknotenmenge:** {12, 13, 14}	
`descendant`	Lokalisiert transitiv alle Kindknoten des aktuellen Knotens, außer Attribut- und Namensraumknoten	**XPath-Ausdruck:** `/node1/node3/node8/descendant::*` **Ergebnisknotenmenge:** {12, 13, 14, 15, 16}	
`descendant-or-self`	Lokalisiert transitiv alle Kindknoten des aktuellen Knotens (außer Attribut- und Namensraumknoten), sowie den Knoten selbst	**XPath-Ausdruck:** `/node1/node3/node8/descendant-or-self::*` **Ergebnisknotenmenge:** {8, 12, 13, 14, 15, 16}	
`parent`	Lokalisiert den Elternknoten des aktuellen Knotes, falls existent	**XPath-Ausdruck:** `/node1/node3/node8/parent::*` **Ergebnisknotenmenge:** {3}	
`ancestor`	Lokalisiert transitiv alle Elternknoten des aktuellen Knotes. Die ancestor-Achse enthält daher immer den Wurzelknoten, außer der aktuelle Knoten ist es selbst; in diesem Falle liefert die Achse die leere Menge	**XPath-Ausdruck:** `/node1/node3/node8/ancestor::*` **Ergebnisknotenmenge:** {1, 3}	
`ancestor-or-self`	Lokalisiert transitiv alle Elternknoten des aktuellen Knotes, sowie den aktuellen Knoten. Diese Achse enthält immer den Wurzelknoten des Dokuments.	**XPath-Ausdruck:** `/node1/node3/node8/ancestor-or-self::*` **Ergebnisknotenmenge:** {1, 3, 8}	
`preceding`	Lokalisiert alle dem aktuellen Knoten vorausgehenden Knoten, ohne seine Vorfahren sowie Attribut- und Namensraumknoten	**XPath-Ausdruck:** `/node1/node3/node8/preceding::*` **Ergebnisknotenmenge:** {2, 5, 6, 7}	
`preceding-sibling`	Lokalisiert die im Dokument vor dem aktuellen Knoten auftretenden Geschwisterknoten	**XPath-Ausdruck:** `/node1/node3/node8/preceding-sibling::*` **Ergebnisknotenmenge:** {7}	
`following`	Lokalisiert alle dem aktuellen Knoten nachfolgenden Knoten ohne dessen Kind-, Attribut und Namensraumknoten	**XPath-Ausdruck:** `/node1/node3/node8/following::*` **Ergebnisknotenmenge:** {9, 4, 10, 11}	
`following-sibling`	Lokalisiert alle „Geschwister“ des aktuellen Knotens, d. h. Knoten auf derselben Hierarchieebene.	**XPath-Ausdruck:** `/node1/node3/node8/following-sibling::*` **Ergebnisknotenmenge:** {9}	
`attribute`	Lokalisiert Attribut(e) eines Knotens	**XPath-Ausdruck:** `/node1/node3/node8/attribute::*` **Ergebnisknotenmenge:** {Att1}	

namespace	Lokalisiert Namensraum-Attribut eines Knotens	**XPath-Ausdruck:** /node1/node3/node8/namespace::* **Ergebnisknotenmenge:** {xmlns:xml="http://www.w3.org/XML/1998/namespace", xmlns:x="namespace:www.jeckle.de/vorlesung/xml"}	

Anmerkung:
Die Achsen `ancestor`, `descendant`, `following`, `preceding` und `self` partitionieren ein Dokument (unter Auslassung der Attribut- und Namensraumknoten): sie überschneiden sich nicht und enthalten alle Elementknoten des Dokuments.



Filterung durch Prädikate:
Ein -- durch eckige Klammern abgegrenztes -- Prädikat kann innerhalb jedes Lokalisierungsschrittes eines XPath-Ausdrucks angegeben werden. Fehlt es, wird die bisher ermittelte Knotenmenge nicht modifiziert.
Das Prädikat kann selbst ein gültiger XPath-Ausdruck sein.
Das prinzipielle Vorgehen kann folgendermaßen beschrieben werden:
Beginnend von links nach rechts für jeden Lokalisierungsschritt: (1) Ermittlung der zur Anfrage passenden Knotenmenge
(2) Reduzierung der Ergebnismenge um diejenigen Knoten, für die das Prädikat `false` liefert.
Befinden sich rechts vom aktuell bearbeiteten Lokalisierungsschritt weitere Ausdrücke, so wird die Resultatmenge als Eingabe eines weiteren Schritts (1) übergeben.

Beispiel 61: Selektion unter Anwendung eines Prädikats

XPath-Ausdruck: `//Person[Qualifikationsprofil]/Nachname`
Ergebnis:

```
<Nachname>Obermüller</Nachname>
```

Der Ausdruck selektiert an beliebiger Stelle des Dokuments („//“) alle Knoten des Typs `Person`. Die Knotenmenge wird um diejenigen `Personen` vermindert, zu denen kein `Qualifikationsprofil` angelegt ist. D.h. Es werden nur diejenigen Knoten selektiert, die über einen Kindknoten des Typs `Qualifikationsprofil` verfügen. Von dieser Knotenmenge (des Typs `Person`!) werden anschließend im zweiten Lokalisierungsschritt die Kindknoten des Typs `Nachname` selektiert.
Mithin liefert der XPath-Ausdruck alle `Nachnamen` von `Personen`, zu denen ein `Qualifikationsprofil` abgelegt ist.
Anmerkung: Das Beispiel nutzt im Prädikat die abkürzende Schreibweise zur Angabe der Vorgabeachse `child`. Die ausführliche Schreibweise -- mit unveränderter Semantik -- des XPath-Ausdruckes lautet daher: `//Person[child::Qualifikationsprofil]/Nachname`

Durch die zusätzliche Definition eines Prädikats für den zweiten Lokalisierungsschritt kann eine weitere Filterung der Ergebnismenge realisiert werden. Zusätzlich können innerhalb eines Prädikats neben XPath-Ausdrücken auch einige vordefinierte Funktionen verwendet werden.
Das Beispiel zeigt die Selektion der `Vornamen` als Kind eines `Personen`-Knotens (Test der Elternschaft durch erstes Prädikat), wenn dieser mit „O“ beginnt (Test durch *starts-with*-Funktion innerhalb des zweiten Prädikats). Die Struktur der Eingabedatei zwingt zusätzlich zur Anwendung der `following`-Achse, da Knoten des Typs `Nachname` in der Dokumentreihenfolge nach Knoten des Types `Vornamen` auftreten.

Beispiel 62: Schrittweise Berechnung einer Selektion unter Verwendung mehrerer Prädikate

XPath-Ausdruck: `//Person[parent::ProjektVerwaltung]/Vorname[starts-with (following::Nachname,'O')]`

Ausgewerteter XPath:`//Person`
Ergebnis:
```
<Person PersID="Pers01" mitarbeitInProjekt="Prj01"> ... </Person>
<Person PersID="Pers02" mitarbeitInProjekt="Prj02"> ... </Person>
<Person PersID="Pers03" mitarbeitInProjekt="Prj02"> ... </Person>
```

Ausgewerteter XPath:`//Person[parent::ProjektVerwaltung]`
Ergebnis:
```
<Person PersID="Pers01" mitarbeitInProjekt="Prj01"> ... </Person>
<Person PersID="Pers02" mitarbeitInProjekt="Prj02"> ... </Person>
<Person PersID="Pers03" mitarbeitInProjekt="Prj02"> ... </Person>
```

Ausgewerteter XPath:`//Person[parent::ProjektVerwaltung]/Vorname`
Ergebnis:
```
<Vorname>Hans</Vorname>
<Vorname>Franz</Vorname>
<Vorname>Xaver</Vorname>
<Vorname>Fritz</Vorname>
```

Ausgewerteter XPath:`//Person[parent::ProjektVerwaltung]/Vorname[following::Nachname]`
Ergebnis:
```
<Vorname>Hans</Vorname>
<Vorname>Franz</Vorname>
<Vorname>Xaver</Vorname>
<Vorname>Fritz</Vorname>
```

Ausgewerteter XPath:
`//Person[parent::ProjektVerwaltung]/Vorname[starts-with(following::Nachname,'O')]`
Ergebnis:
```
<Vorname>Franz</Vorname>
<Vorname>Xaver</Vorname>
```

Die durch die XPath-Spezifikation vordefinierten Funktionen lauten in der Übersicht:

Tabelle 19: XPath-Funktionen für Knotenmengen (node-sets)

Funktionsprototyp	Funktionalität
number last()	Liefert die Größe der aktuellen Knotenmenge; damit den Index des letzten Elements
number position()	Liefert die Position des aktuellen Knotens innerhalb der Knotenmenge. Die erste Knoten trägt die Positionsnummer 1.
number count(*node-set*)	Liefert Elementzahl der übergebenen Knotenmenge
node-set id(*object*)	Liefert denjenigen Knoten, dessen ID-typisiertes Attribut den Argumentwert aufweist. *Anmerkung*: Zur Nutzung dieser Funktion muß zwingend eine Dokument-Grammatik (DTD oder Schema) zum Eingangsdokument vorliegen.
string local-name (*node-set*?)	Liefert den local name (oder die Menge der Namen) der übergebenen Knotenmenge. Wird keine Knotenmenge übergeben, dann wird der aktuelle Knoten als Argument genutzt.
string namespace-uri (*node-set*?)	Liefert die Namensraum-URI der übergebenen Knotenmenge. Wird keine Knotenmenge übergeben, dann wird der aktuelle Knoten als Argument genutzt. *Anmerkung*: Handelt es sich nicht um einen Element- oder Attributknoten, so ist die retournierte Zeichenkette leer.
string name (*node-set*?)	Liefert die QName(n) (=qualifizierte(r) Name(n) aus Namensraumkürzel und local name) der übergebenen Knotenmenge, oder des aktuellen Knotens bei leerer Knotenmenge. *Anmerkung*: Nur für Element- und Attributknoten liefert name andere Resultate als local-name.

Tabelle 20: XPath-Funktionen für Zeichenketten

Funktionsprototyp	Funktionalität
string string (*object*)?	Liefert Zeichenkettenrepräsentation einer Knotenmenge. Dabei wird der Zeichenkettenwert des ersten Knotens in der Dokumentreihenfolge zurückgegeben, andernfalls die leere Zeichenkette.
string concat (*string*, *string*, *string**)	Verkettet mindestens zwei Zeichenketten.
boolean starts-with (*string1*, *string2*)	Liefert true falls string1 das zweite Argument string2 als Präfix enthält; andernfalls false
boolean contains (*string1*, *string2*)	Liefert true falls string1 die Zeichenkette aus string2 enthält; andernfalls false.
string substring-before (*string1*, *string2*)	Liefert denjenigen Teil der Zeichenkette *string1*, der sich vor dem ersten Auftreten der Zeichenkette *string2* befindet.
string substring-after (*string1*, *string2*)	Liefert denjenigen Teil der Zeichenkette *string1*, der sich nach dem ersten Auftreten der Zeichenkette *string2* befindet.
string substring (*string*, *number1*, *number2*?)	Liefert eine Zeichenkette der Länge *number2* aus *string*, beginnend mit der Position *number1*. Fehlt das dritte Argument, so wird der Teilstring bis zum Ende der Zeichenkette *string* zurückgegeben. *Anmerkung*: Das erste Zeichen trägt die Indexnummer 1, nicht 0 wie in Java und C üblich.
number string-length(*string*?)	Liefert die Länge der übergebenen Zeichenkette. Wird kein Argument übergeben, so wird die Länge des zuvor in eine Zeichenkette konvertierten aktuellen Knotens zurückgegeben.
string normalize-space (*string*?)	Liefert die übergebene Zeichenkette unter Entfernung führender, schließender und mehrfacher Leerzeichen zurück. Ferner werden noch evtl. in der Argumentzeichenkette enthaltenen Entitätsreferenzen aufgelöst. *Anmerkung*: Der Normalisierungsvorgang entspricht damit der Attributwertenormalisierung nach Abschnitt 3.3.3 der XML-Spezifikation.
string translate (*string1*, *string2*, *string3*)	Liefert die Zeichenkette *string1* wobei jedes Zeichen aus *string2* durch das Zeichen an derselben Position aus *string3* ersetzt wurde.

Tabelle 21: Boole'sche XPath-Funktionen

Funktionsprototyp	Funktionalität
boolean boolean (*object*)	Liefert die Boole'sche Repräsentation des übergebenen Arguments. Hierbei gilt: •Eine Zahl wird genau dann nach true konvertiert, wenn sie weder Null (unbeachtlich ihres Vorzeichens) noch eine nicht darstellbare Zahl (NaN) ist. •Eine Knotenmenge ergibt true, wenn sie nicht leer ist. •Eine Zeichenkette ergibt true, wenn sie nicht leer (d.h. Länge größer Null) ist. •Die Konvertierung anderer Typen ist typabhängig, und nicht durch den Standard festgelegt
boolean not (*boolean*)	Negiert das übergebene Argument
boolean true()	Liefert statisch den Wert true
boolean false()	Liefert statisch den Wert false
boolean lang (*string*)	Liefert true wenn der aktuelle Knoten ein xml:lang-Attribut gemäß der als Argument übergebenen Sprache besitzt

Tabelle 22: Zahlenorientierte XPath-Funktionen

Funktionsprototyp	Funktionalität
number number (*object*?)	Konvertiert ein Objekt in eine Zahl gemäß folgender Regeln: •Eine Zeichenkette wird in eine Fließkommazahl gemäß IEEE 754 konvertiert, wenn sie aus einem optionalen Leerzeichen, gefolgt durch ein optionales Minuszeichen, gefolgt von einem optionalen Leerzeichen und einer Ziffernfolge besteht. •Der Boole'sche Wert true wird zu 1, der Wert false zu 0 konvertiert. •Eine Knotenmenge wird zunächst in eine Zeichenkette übersetzt, und dann gemäß der oben definierten Regeln umgesetzt. •Die Konvertierung anderer Typen erfolgt typabhängig, und ist nicht durch den Standard geregelt. Wird kein Argument übergeben, so wird stattdessen der aktuelle Knoten als einziges Element einer Knotenmenge interpretiert.
number sum (*node-set*)	Liefert die Summe aller Elemente der übergebenen Knotenmenge, die zuvor in eine Zahl konvertiert werden.
number floor (*number*)	Liefert die größte ganze Zahl, die nicht größer als das Argument ist. *Anmerkung*: Entspricht dem Abschneiden beliebiger Nachkommastellen
number ceiling (*number1*)	Liefert die kleinste ganze Zahl, die nicht kleiner als das Argument ist. *Anmerkung*: Entspricht floor(number1+0.999...)
number round (*number*)	Liefert das Argument auf die nächste ganze Zahl gerundet. Gibt es zwei solche -- wie bei Nachkommastelle gleich 0.5 immer der Fall -- so wird die größere zurückgeliefert.

Für mathematische Berechnungen auf zahlenartigen Knoten stehen folgende Operatoren zur Verfügung.

Tabelle 23: Mathematische Operatoren

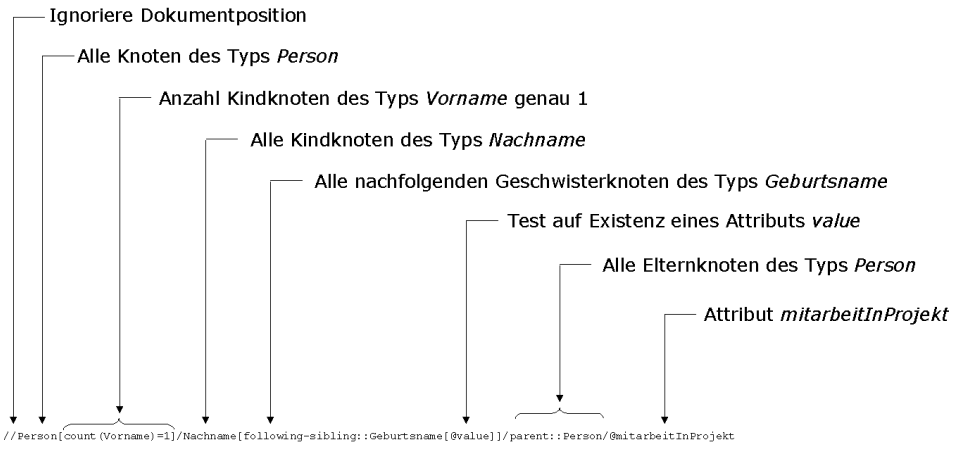
Operator	Funktionalität
+	Addition
-	Subtraktion als zweistelliger Operator. Der einstellige Operator - ist nicht spezifiziert, er liefert üblicherweise die negative Zahlendarstellung.
*	Multiplikation. Außer wenn innerhalb von XPath-Ausdrücken als Knotentest eingesetzt.
div	Division Achtung: Das Symbol / dient ausschließlich als Trennzeichen zur Separierung von Lokalisierungspfaden!
mod	Rest einer ganzzahligen Division

Ein umfangreiches Beispiel: Für das nachfolgende Beispiel wird das Projektverwaltungsdokument erweitert zu:

Beispiel 63: Erweiterte Projektverwaltung

```
(1)<?xml version="1.0" encoding="ISO-8859-15"?>
(2)<ProjektVerwaltung xmlns:xhtml="http://www.w3.org/1999/xhtml" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:noNamespaceSchemaLocation="L:\vorlesung\xml\examples\projektverwaltung3.
xsd">
(3)     <Person PersID="Pers01" mitarbeitInProjekt="Prj01">
(4)             <Vorname>Hans</Vorname>
(5)             <Nachname>Hinterhuber</Nachname>
(6)     </Person>
(7)     <Person PersID="Pers02" mitarbeitInProjekt="Prj02">
(8)             <Vorname>Franz</Vorname>
(9)             <Vorname>Xaver</Vorname>
(10)            <Nachname>Obermüller</Nachname>
(11)            <Qualifikationsprofil>
(12)                    <xhtml:u>IT-Kompetenz</xhtml:u>
(13)                    <xhtml:em>verschiedene</xhtml:em> Betriebssysteme und
(14)         <Leistungsstufe>professionelle</Leistungsstufe>
(15)                    <xhtml:em>
(16)                            <Qualifikation>Programmierung</Qualifikation>
(17)                    </xhtml:em>
(18)         verschiedener Programmiersprachen
(19)         <xhtml:em>
(20)                            <xhtml:u>
(21)                                    <Qualifikation>Entwickler</Qualifikation>
(22)                            </xhtml:u>
(23)                    </xhtml:em> von 1988-1990
(24)         <xhtml:u>
(25)                            <Qualifikation>Projektleiterfunktion</Qualifikation>
(26)                    </xhtml:u>
(27)         von <xhtml:b>1990-93</xhtml:b> im X42-Projekt in Abteilung AB&amp;amp;C
(28)       </Qualifikationsprofil>
(29)    </Person>
(30)    <Person PersID="Pers03" mitarbeitInProjekt="Prj02">
(31)            <Vorname>Fritz</Vorname>
(32)            <Nachname>Meier</Nachname>
(33)            <Geburtsname value="Huber"/>
(34)    </Person>
(35)    <Projekt ID="Prj01" Projektleiter="Pers01" Mitarbeiter="Pers01"/>
(36)    <Projekt ID="Prj02" Projektleiter="Pers02" Mitarbeiter="Pers03"/>
(37)</ProjektVerwaltung>
```

Download des Beispiels



Der XPath-Ausdruck der Abbildung 31 lokalisiert den Attributknoten des Inhalts `Prj02`.

Übung 2: Einige Übungen

Welches Ergebnis liefern folgende XPath-Ausdrücke?
(a) `//Person[//child::Qualifikationsprofil]/Nachname`
(b) `//Person[parent::ProjektVerwaltung]/Vorname[following-sibling::Vorname]`
(c) `/ProjektVerwaltung/Person[attribute::PersID='Pers01']//Nachname`

Wie muß ein XPath-Ausdruck lauten, um folgendes zu selektieren?
(d) Selektion aller Personen mit Nachnamen „Obermüller“.
(e) Selektion aller Nachnamen von Personen die über mehr als eine Qualifikation verfügen.
(f) Selektion der Nachnamen aller Projektleiter.

Anwendungsbeispiel: Integritätsbedingungen in XML-Schema

Die bei der Diskussion des Referenzierungsmechanismus für DTDs geäußerte Kritik gilt prinzipiell auch für die Verwendung von XML-Schema fort. Zwar steht mit XLink ein deutlich flexiblerer Verknüpfungsmechanismus zur Verfügung, jedoch verbleibt das Problem der Gültigkeit von Referenzen. Sie bleibt auch bei der Verwendung von XML-Schema, durch die unverändert von den DTDs übernommene Semantik der Datentypen `ID` und `IDREF`, `IDREFS`, auf das aktuelle Dokument beschränkt. Zusätzlich ist eine Einschränkung, etwa auf den aktuellen Namensraum, nicht möglich. Daher müssen alle Belegungen aller ID-typisierten Attribute zwingend dokumentweit eindeutig sein.

Über die Möglichkeiten der Datentypen hinausgehend bietet XML-Schema das Element `unique` zur Definition eindeutiger Wertbelegungen an. Hierbei wird auf die Lokatorsprache XPath zurückgegriffen um die abzuprüfenden Knoten innerhalb des Dokuments zu bezeichnen.

Die Syntax verwendet XPath-Ausdrücke eingeschränkter Mächtigkeit sowohl zur Festlegung des der Knotenmenge, auf die sich die Einschränkung bezieht (`selector`), als auch zur Angabe der eingeschränkten Knoten (`field`) selbst.

```
(1)<xsd:unique name="aName">
(2)     <xsd:selector xpath="aValidXPath"/>
(3)     <xsd:field xpath="aFieldStatement"/>
(4)     ...
(5)</xsd:unique>
```

Die Mächtigkeit der XPath-Ausdrücke ist dahingehend eingeschränkt, daß für das `selector`-Element ausschließlich Ausdrücke erlaubt sind, die Kindelemente des Knotens liefern, in dessen Kontext die durch `unique` formulierte Einschränkung angegeben wird. Als Konsequenz ist die Nutzung der verfügbaren XPath-Achsen auf diejenigen beschränkt, die Element-Knotenmengen zurückliefern.
Die Lokationsausdrücke in den -- möglicherweise mehrfach auftretenden -- `field`-Elementen werden relativ zum Pfad des `selector`-Knotens interpretiert. Hintereinandergesetzt muß der Pfad eines `selector`-Elements, gefolgt von einem Pfad eines `field`-Elements, einen gültigen Lokationsausdruck ergeben, der genau einen Knoten oder genau ein Attribut in der Ergebnismenge liefert. Sind mehrere `field`-Elemente zu einem `selector`-Element gegeben, so werden diese als durch logisches *und* verknüpft interpretiert. Mithin entspricht diese Semantik einem *concatenated primary key* aus den relationalen Datenbanken.

Das Beispiel zeigt die Nutzung des `unique`-Konstrukts zur Angabe der Eindeutigkeitsbedingung für das Attribut `PersID` des Elements `Person`.
Zunächst selektiert der Pfad `/Person` alle Knoten des gleichnamigen Typs; durch das `field`-Element wird die Eindeutigkeitsbedingung auf alle Attribut-Kindnoten des Typs `PersID` der Knoten in der selektierten Knotenmenge angewendet.
Die Semantik ist damit zur bisherigen `ID`-Typisierung identisch.

Beispiel 64: Unique-Einschränkung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema
(3)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)<xsd:element name="ProjektVerwaltung">
(7)     <xsd:complexType>
(8)             <xsd:sequence>
(9)                     <xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/>
(10)                    <xsd:element name="Projekt" type="ProjektType" maxOccurs="unbounded"/>
(11)            </xsd:sequence>
(12)            <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(13)    </xsd:complexType>
(14)    <xsd:unique name="uniquenessPersID">
(15)            <xsd:selector xpath="Person"/>
(16)            <xsd:field xpath="@PersID"/>
(17)    </xsd:unique>
(18)</xsd:element>
(19)
(20)<xsd:complexType name="PersonType">
(21)    <xsd:attribute name="PersID" type="xsd:token"/>
(22)</xsd:complexType>
(23)
(24)<xsd:complexType name="ProjektType"/>
(25)
(26)</xsd:schema>
```

Download des Beispiels

Das nächste Beispiel zeigt die Verwendung mehrerer `field`-Elemente zur Realisierung zusammengesetzter Schlüssel. Hierzu wird die Kombination aus dem Inhalt des `Nachnamen`- und des `Vornamen`-Elements zusammen als eindeutig deklariert.
Überdies zeigt das Beispiel die Anwendung des Schlüsselmechanismus auf Elemente ohne Änderung der Basissyntax, abgesehen von der geänderten XPath-Achse.

Beispiel 65: Zusammengesetzter Schlüssel innerhalb eines unique-Elements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema
(3)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)<xsd:element name="ProjektVerwaltung">
(7)     <xsd:complexType>
(8)             <xsd:sequence>
(9)                     <xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/>
(10)                    <xsd:element name="Projekt" type="ProjektType" maxOccurs="unbounded"/>
(11)            </xsd:sequence>
(12)            <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(13)    </xsd:complexType>
(14)    <xsd:unique name="uniquenessPersID">
(15)            <xsd:selector xpath="Person"/>
(16)            <xsd:field xpath="Vorname"/>
(17)            <xsd:field xpath="Nachname"/>
(18)    </xsd:unique>
(19)</xsd:element>
(20)
(21)<xsd:complexType name="PersonType">
(22)    <xsd:sequence>
(23)            <xsd:element name="Vorname" type="xsd:token" minOccurs="1" maxOccurs="unbounded"/>
(24)            <xsd:element name="Nachname" type="xsd:token" maxOccurs="1"/>
(25)    </xsd:sequence>
(26)</xsd:complexType>
(27)
(28)<xsd:complexType name="ProjektType"/>
(29)
(30)</xsd:schema>
```

Download des Beispiels

Das Pendant des `IDREF(S)`-Attributtyps bildet die Kombination der XSD-Elemente `key` und `keyref`.
Hierzu lokalisiert key auf der Basis eines XPath-Ausdruckes eine Referenzmenge, während keyref diejenige Knotenmenge lokalisiert, in der ausschließlich Elemente der Referenzmenge enthalten sein dürfen.
Das Beispiel zeigt die Anwendung auf das Element `ProjektVerwaltung`. Der mit *projectKey* benannte Schlüssel definiert die Referenzmenge als das Ergebnis der Anfrage `Projekt/@ID`, worauf die *projectReference* Bezug nimmt.

Beispiel 66: Schlüsselbasierte Referenzierung

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsd:schema
(3)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(4)     elementFormDefault="qualified"
(5)     attributeFormDefault="unqualified">
(6)     <xsd:element name="ProjektVerwaltung">
(7)             <xsd:complexType>
(8)                     <xsd:sequence>
(9)                             <xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/>
(10)                            <xsd:element name="Projekt" type="ProjektType"
maxOccurs="unbounded"/>
(11)                    </xsd:sequence>
(12)                    <xsd:attribute name="version" type="xsd:string" fixed="1.0"/>
(13)            </xsd:complexType>
(14)
(15)            <xsd:key name="projectKey">
(16)                    <xsd:selector xpath="Projekt"/>
(17)                    <xsd:field xpath="@ID"/>
(18)            </xsd:key>
(19)            <xsd:keyref name="projectReference" refer="projectKey">
(20)                    <xsd:selector xpath="Person"/>
(21)                    <xsd:field xpath="@mitarbeitInProjekt"/>
(22)            </xsd:keyref>
(23)    </xsd:element>
(24)
(25)    <xsd:complexType name="PersonType">
(26)            <xsd:attribute name="mitarbeitInProjekt" type="xsd:token"/>
(27)    </xsd:complexType>
(28)    <xsd:complexType name="ProjektType">
(29)            <xsd:attribute name="ID" type="xsd:token"/>
(30)    </xsd:complexType>
```

```
(31)</xsd:schema>
```

Download des Beispiels

Zusammenfassend lassen sich folgende Vorteile gegenüber dem `ID/IDREF`-Mechanismus festhalten:

- Neben Attributen können auch Elemente und ihre Inhalte Beschränkungen unterworfen werden.
- Typisierung des Elements oder Attributs nicht mehr berührt.
  Trennung zwischen Inhaltsmodell und Eindeutigkeitsbedingung erlaubt Beschränkung beliebig typisierter Elemente und Attribute.
- Eindeutigkeitsbeschränkungen sind auf Gültigkeitsbereich des definierenden Elements beschränkt.
- Zusammengesetzte Schlüssel möglich.
- Schlüsselbestandteile müssen im XML-Dokument zwingend mit Werten (ungleich `nil`) belegt sein.
- Gleichheitsvergleich erfolgt typbasiert.

Web-Referenzen 12: Weiterführende Links

•XPath Spezifikation
•Deutsche Übersetzung der XPath-Spezifikation
•XPath Visualisierer (Java-basiert)
•Visual XPath (.NET Windows-Applikation)
Originalbezugsquelle
•XPath Explorer (Java-basiert)
Originalbezugsquelle
•Online Experimentieren mit XPath

2.3 Transformation von XML-Dokumenten: XSL Transformations

Die prominenteste Verwendung der XPath-Pfadausdrücke und eine der in jüngerer Zeit am weitesten beachteten XML-Vokabulare dürfte XSLT -- die Sprache der *XSL Transformations* -- sein. Sie wurde im Verlauf der Standardisierung der Stylesheetsprache XSL von dieser abgetrennt und seither durch eine eigene W3C-Arbeitsgruppe vorangetrieben.
In einem Satz zusammengefaßt stellt sie eine Turing-vollständige funktionale Programmiersprache zur Transformation wohlgeformter XML-Dokumente in beliebige Unicode-Streams -- und damit im speziellen wiederum in XML-Dokumente -- dar.
Der Begriff *Transformationen* bezeichnet hierbei die Selektion einzelner Bestandteile des Quelldokuments, deren Umordnung sowie die Ableitung neuer Inhalte aus den bereits bestehenden.
Derzeit aktuell ist die Recommendation zur Version 1.0 von Herbst 1999. Intern arbeitet das W3C bereits an der Nachfolgerversion XSLT v2.0, welche auch die absehbaren Entwicklungen des Umfeldes, wie XPath v2.0 oder XML-Schema, berücksichtigen wird.

Jedes XSLT-Stylesheet ist ein gültiges XML-Dokument, in dem alle Elemente der Sprache XSLT im Namensraum `http://www.w3.org/1999/XSL/Transform` plaziert sind.
Üblicherweise wird der Namensraum an das Präfix `xsl` gebunden, welches allen XSLT-Sprachelementen explizit vorangestellt wird.
Das Wuzelelement eines XSLT-Dokuments bildet der Knoten `stylesheet`. Alternativ kann auch `transform` angegeben werden. Zusätzlich verfügt jedes Stylesheet über ein Versionsattribut zur Bezeichnung der verwendeten XSLT-Version.
Das Beispiel zeigt ein minimales Stylesheet

Beispiel 67: Ein minimales Stylesheet

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```

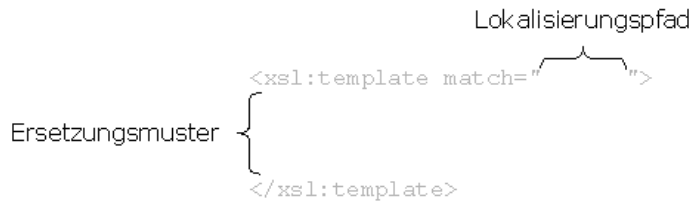
Download des Beispiels

Angewendet auf das Beispieldokument der Projektverwaltung liefert es folgende Ausgabe:

```
(1)<?xml version="1.0" encoding="utf-8"?>
(2)Hans Hinterhuber
(3)Franz Xaver Obermüller
(4)IT-Kompetenz verschiedene Betriebssysteme und professionelle Programmierung verschiedener
Programmiersprachen
(5)Entwickler von 1988-1990 Projektleiterfunktion von 1990-93 im X42-Projekt in Abteilung AB&amp;amp;
C
(6)Fritz Meier
```

Dies mag zunächst verwundern, definiert doch das Beispiel-Stylesheet keinerlei Transformationsregeln oder Ausgabefunktionen.
Das Ergebnis des minimal-Beispiels wird durch die eingebauten Vorgaberegeln erzeugt. Diese geben, sofern nicht anders angegeben alle Character Information Items unverändert aus.
Durch Überschreiben dieser Vorgaben und die Definition eigener Regeln lassen sich komplexe Transformationen auf dem Eingabedokument verwirklichen.

Transformationsschablonen



Die Graphik zeigt den Aufbau einer Transformationsschablone. Ihre beiden Hauptbestandteile sind der *Lokalisierungspfad* und das *Ersetzungsmuster*.
Der Lokalisierungspfad (in der Spezifikation als *pattern* bezeichnet) liefert eine Knotenmenge. Als Syntax wird eine eingeschränkte Variante der Lokatorsprache XPath verwendet.
Augenfälligster Unterschied zur bisherigen Notation ist die Optionalität der descendant-Achse (zumeist zu `//` verkürzt) zur hierarchiebenenunabhängigen Lokalisierung eines Knotens.
Im Rumpf des Musters legt das Ersetzungsmuster diejenige Zeichenfolge fest, die statt jedem Element der lokalisierten Knotenmenge ausgegeben werden soll.
Das nachfolgende Transformationssheet liefert angewandt auf das Projektverwaltungsbeispiel dreimal die Ausgabe `Person gefunden!`; für jedes Auftreten des Knotens `Person` in der Eingabe.

Beispiel 68: Eine einfache Transformation

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)     <xsl:template match="Person">
(6)             <xsl:text>Person gefunden!</xsl:text>
(7)     </xsl:template>
(8)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Innerhalb des Ersetzungsmusters können im Allgemeinen beliebige Textsequenzen angegeben werden. Insbesondere ist die Verwendung wohlgeformter XML-Fragmente zugelassen.
Textsequenzen werden hierbei durch direktes Anschreiben, oder umschlossen durch das Element `xsl:text` definiert.
Anmerkung: Die Anforderungen an die Wohlgeformtheit sind hierbei auf die korrekte Terminierung der Elemente (damit einhergehend ihre korrekte Schachtelung), sowie die Quotierung der Attribute beschränkt.
Im folgenden Beispiel wird jedes Element des Typs `Person` durch ein leeres `Mitarbeiter`-Element ersetzt.
Das Beispiel erklärt auch die übliche Anwendungspraxis, alle XSLT-Elemente durch Namensraumpräfix zu qualifizieren, statt der -- weniger schreibaufwendigen -- Überschreibung des Vorgabenamensraumes. Würde der Vorgabenamensraum mit der Namensraum-URI der XSL-Transformations belegt, so befände sich auch jedes XML-Element und -Attribut innerhalb des Ersetzungsmusters in diesem Namensraum. Als Konsequenz würde der XSLT-Prozessor die Transformation wegen des Auftretens ungültiger (d.h. nicht in der XSLT-Sprache enthaltener) Elemente ablehnen. Bei Redefinition des Vorgabenamensraumes müßte daher für alle Elemente, die nicht Bestandteil von XSLT sind, eine explizite Namensraumdefinition im Element erfolgen, wodurch die Lesbarkeit stark herabgesetzt würde.

Beispiel 69: Erzeugung einer XML-Ausgabe

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter/>
(7)</xsl:template>
(8)
(9)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Erwartungsgemäß liefert das Beispiel dreifach das leere Element `Mitarbeiter` anstelle der `Personen`-Elemente der Eingabe.
Die bisher genutzte Form der Ersetzung ist jedoch für die praktische Anwendung in nur äußerst wenigen Fällen geeignet, da sie keine Übernahme von Daten der Eingabedatei in die Ausgabe erlaubt.
Dieses Manko wird durch das XSLT-Sprachelement `value-of` behoben.

Das Element `value-of` übernimmt als Teil des Ersetzungsmusters frei selektierbare Text-artige Informationsknoten aus dem Quelldokument. Die Lokalisierung der zu übernehmenden Knoten erfolgt durch XPath-Syntax innerhalb des `select`-Attributs.
Im folgenden Beispiel wird der Inhalt der `Personen`-Knoten in den neuen Knotentyp `Mitarbeiter` übernommen.

Beispiel 70: Übernahme bestehender Information aus dem Quelldokument

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter>
(7)             <xsl:value-of select="self::*"/>
(8)     </Mitarbeiter>
(9)</xsl:template>
(10)
(11)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Das Resultat liefert jedoch nicht das Quelldokument, unter Umbenennung der Personen-Knoten in Mitarbeiter, sondern die dargestellte Textsequenz.

```
(1)<?xml version="1.0" encoding="utf-8"?>
(2)<Mitarbeiter>Hans Hinterhuber</Mitarbeiter>
(3)<Mitarbeiter>Franz Xaver Obermüller IT-Kompetenzverschiedene Betriebssysteme und professionelle
(4)Programmierung verschiedener Programmiersprachen Entwickler von 1988-1990
Projektleiterfunktion
(5)von 1990-93 im X42-Projekt in Abteilung AB&amp;amp;C</Mitarbeiter>
(6)<Mitarbeiter>Fritz Meier</Mitarbeiter>
```

Die Lösung liegt in der Definition des value-of-Elements. Es konvertiert alle durch den im select-Attribut bezeichneten XPath lokalisierten Knoten in ihre Textrepräsentation. Im vorliegenden Beispiel ist dies der Inhalt der Knoten des Typs Person, der durch den Elementinhalt gebildet wird. Der Elementinhalt wird hierbei durch alle Kindknoten und deren Attribute gebildet.

Zur unveränderten Übernahme eines vollständigen Elements einschließlich der Auszeichnungssymbole wird das XSLT-Element copy-of angeboten.
Das nachfolgende Beispiel modifiziert das vorhergend diskutierte Stylesheet dergestalt, daß für alle Personen-Knoten zunächst ein öffnender Mitarbeiter-Tag gesetzt wird. Im Rumpf des so begonnenen Elements werden durch das copy-of-Element alle Kindknoten der aktuellen Knotenmenge unverändert kopiert.
Als zusätzliche Veränderung gegenüber dem vorigen Beispiel fällt die Nutzung der child-Achse im select-Attribut des copy-of-Elements auf. Dies ist notwendig, da durch den Lokalisierungspfad der Schablone alle Personen-Knoten zu einer Ergebnisknotenmenge zusammengefaßt werden. Die Anwendung der self-Achse innerhalb des copy-of-Elements würde daher auch die Personen-Knoten selbst übernehmen.
Zusammenfassend läßt sich die Funktionalität des Beispiels mit *Umbenennung aller Elemente des Types Person in Mitarbeiter* wiedergeben.

Beispiel 71: Kopieren vollständiger Elementknoten

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="//Person">
(6)     <Mitarbeiter>
(7)             <xsl:copy-of select="child::*"/>
(8)     </Mitarbeiter>
(9)</xsl:template>
(10)
(11)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Die vorgestellte Transformationsvorschrift ist hinreichend flexibel für Knoten des Typs Person auf beliebiger Hierarchiestufe des Eingabedokuments. Jedoch versagt sie bereits beim Versuch einen weiteren Transformationsschritt einzubauen, der auf einem der unverändert kopierten Kindelemente von Person operiert.
Das folgende Stylesheet stellt einen flexibleren Ansatz zur Umbenennung von einzelnen Knoten vor.

Gegenüber der vorhergehenden Fassung ist der Umfang um zwei weitere template-Elemente erweitert. Eines, das für alle Qualifikationsprofil-Knoten angewendet wird und eines für alle anderen Knoten. Der dort eingesetzte Lokalisierungspfad liefert als Ergebnismenge die Vereinigung aller Attributknoten (attribute::*) mit allen Knoten außer dem Wurzelknoten (node()-Funktion). Im Rumpf des Elements wird das copy-Element zur Kopie jedes Elements verwendet. Anders als das bisher herangezogene copy-of-Element übernimmt diese Variante ausschließlich das aktuelle Element in das Ergebnisdokument und läßt eventuell vorhandene Kindelemente unberücksichtigt.
Das template-Element mit dem Lokalisierungspfad Qualifikationsprofil verfügt über kein Ersetzungsmuster. Es bewirkt damit die Unterdrückung des Teilbaumes unterhalb aller Knoten des Typs Qualifikationsprofil.

Die korrekte Anwendung der verschiedenen Schablonen wird durch den ausführenden XSLT-Prozessor gewährleistet, er ermittelt anhand der Lokalisierungspfade das best-passendste Template und bringt es zur Ausführung. Wenn, wie im vorliegenden Beispiel, mehrere Pfadausdrücke einen Knoten des Eingabedokuments lokalisieren, so wird das am weitesten spezifizierte Muster ausgewählt. Im untenstehenden Beispiel gilt dies für alle Elemente des Typs Person. Jeder dieser Knoten ist sowohl durch den XPath-Ausdruck der ersten Schablone als auch durch den allgemeineren

Ausdruck node() zugänglich. Der explizite Pfad Person des ersten Lokalisierungsmusters ist jedoch gegenüber der Ergebnismenge der node-Funktion (deutlich) spezifischer.

Da jeder Knoten, außer denen des Typs Person und Qualifikationsprofil, unverändert in den Ausgabestrom übernommen werden soll, wird im Beispiel wieder ein copy-Element eingesetzt. Im Unterschied zum bisher verwendeten copy-of jedoch mit der Einschränkung, daß copy nur den aktuellen Knoten kopiert und eventuell existierende Kindknoten unberücksichtigt läßt. Dieser Vorgang wird auch als *shallow copy* bezeichnet.

Steuerung der Transformationsreihenfolge durch apply-templates-Elemente

Standardmäßig durchläuft ein XSLT-Prozessor den aus dem Eingabedokument erzeugten Baum ausgehend vom Wurzelknoten in Preorder Reihenfolge. Während des Traversierungsvorganges werden die zum jeweiligen Knoten „passenden" Schablonen ausgewertet. „Passend" deutet hierbei auf das Enthaltensein des besuchten Knotens in der Ergebnismenge eines XPath-Ausdrucks innerhalb eines match-Attributes hin.
Jedoch ist auch die anwenderdefinierte Beeinflussung der vorgegebenen Abarbeitungsreihenfolge möglich. Hierzu enthält das Beispiel zwei apply-templates-Elemente. Diese lösen einen Rekursionsschritt aus, dergestalt, daß an jeder Stelle, an der sich ein apply-templates-Aufruf findet, der Prozessor versucht, weitere passende Schablonen anhand der angegebenen Lokalisierungspfade zu ermitteln. Diese können an der gegebenen Stelle ausgewertet werden. Der Vorgang entspricht damit der Substitution in funktionalen Programmiersprachen.
Im vorliegenden Fall wird nach Ausgabe des öffnenden Tags Mitarbeiter -- nachdem ein Personen-Knoten im Eingabedokument ermittelt wurde -- nach weiteren Knoten im Eingabedokument gesucht, zu denen Lokalisierungspfade in der XSLT-Transformationsvorschrift existieren. Dies ist für alle Attribute und Kindknoten von Person der Fall, da sie durch den Lokalisierungspfad attribute::*|node() zugänglich sind. So wird innerhalb des neu erzeugten Elements Mitarbeiter des Ausgabestroms das Ersetzungsmuster ausgeführt, das die Elemente und Attribute mit ihren Inhalten unverändert übernimmt.
Als Besonderheit nutzt das apply-templates-Element im „allgemeinen" (dritten) template das Attribut select. Es erlaubt die anwenderdefinierte Steuerung der Menge, innerhalb der nach weiteren *passenden* Lokalisierungspfaden gesucht werden soll. Standardmäßig ist diese Menge mit allen dem aktuellen Element nachfolgenden (following-Achse) Elementknoten gefüllt. Im vorliegenden Beispiel wird sie durch den XPath des Attributwertes um alle Attributknoten erweitert.

Beispiel 72: Flexible Umbenennung und Löschung von Elementen

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter>
(7)             <xsl:apply-templates/>
(8)     </Mitarbeiter>
(9)</xsl:template>
(10)
(11)<xsl:template match="Qualifikationsprofil"/>
(12)
(13)<xsl:template match="attribute::*|node()">
(14)    <xsl:copy>
(15)            <xsl:apply-templates select="attribute::*|node()"/>
(16)    </xsl:copy>
(17)</xsl:template>
(18)
(19)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

Übung 3: Verfolgung der Aufrufreihenfolge
Lassen Sie sich mit von einem XSLT-Prozessor die Aufrufreihenfolge der einzelnen Templates ausgeben. Beispielsweise mit dem Prozessor Xalan aus dem Apache-Projekt.

Benannte Ersetzungsschablonen und Parameterübergabe

Neben den Lokationspfad-gesteuerten Schablonen kann der Anwender auch benannte Schablonen, ohne match-Attribut, definieren. Diese werden während der Abarbeitung des Eingabebaumes nicht berücksichtigt, sondern müssen manuell durch call-template aufgerufen werden.
Konzeptionell entsprechen sie Funktionsaufrufen herkömmlicher Programmiersprachen.
(In Spezifikation nachschlagen).
Die Definition erfolgt analog den bisher genutzten Schablonen, mit der Ausnahme, daß statt dem match-Attribut ein eindeutiger Name (Attribute name) angegeben wird.

Als neuer Freiheitsgrad beim nun notwendigen manuellen Aufruf tritt die Möglichkeit der Parameterübergabe hinzu.
Als Parameter können beliebige Dokumentbestandteile als Knotenmenge, Ergebnisse von Funktionsausdrücken oder Konstanten übergeben werden.
Eine Parameterrückgabe ist nicht möglich, sie wird durch den Anteil der Schablone an der Ausgabe realisiert.
Die Aufrufsyntax lautet:

```
(1)<xsl:call-template name="QName">
```

```
(2)     <!-- Content: xsl:with-param* -->
(3)</xsl:call-template>
(4)<xsl:with-param name="QName" select="eingeschränkter XPath-Ausdruck">
(5)     <!-- Content: template -->
(6)</xsl:with-param>
```

Anmerkung: Wie in allen funktionalen Sprachen kommt den Funktionen dieses Typs besondere Bedeutung, als Ausgangspunkt rekursiver Aufrufe, zu.

Die Vorgabe-Transformationsregeln

Das einführende Beispiel dieses Kapitels griff bereits auf die in den XSLT-Prozessor „eingebauten" Standard-Transformationsvorschriften (*built-in templates*) zurück.
So lautet die Definition, welche für alle Text- und Attributknoten der Eingabe den Inhalt in die Ausgabe kopiert:

```
(1)<xsl:template match="text()|@*">
(2)     <xsl:value-of select="."/>
(3)</xsl:template>
```

Ferner existiert ein `template` zur rekursiven Abarbeitung des Eingabebaumes, welches immer dann Anwendung findet, wenn sich keine spezialisiertere Transformationsvorschrift findet.

```
(1)<xsl:template match="*|/">
(2)     <xsl:apply-templates/>
(3)</xsl:template>
```

Ergänzt wird diese Zusammenstellung durch eine Schablone zur Eliminierung von Namensraum- und Kommentarknoten.

Elemente der Ablaufsteuerung

Ähnlich zu herkömmlichen Programmiersprachen bietet auch XSLT Sprachmittel zur Selektion und bedingten Verarbeitung, abhängig von den Eingabedaten.
So erlaubt das `if`-Element die Bearbeitung der umschlossenen Elemente nur, wenn die im `test`-Attribut formulierte Boole'sche Bedingung wahr ist.
(In Spezifikation nachschlagen)
Die Syntax lautet:

```
<xsl:if test="Boole'scher Ausdruck">   <!-- Content: template --> </xsl:if>
```

Das Beispiel zeigt die Nutzung des `if`-Elements. Verfügt ein Knoten des Typs `Person` über mehr als einen Kindknoten des Typs `Vorname` so wird der Inhalt des `if`-Elements abgearbeitet, der durch das XSLT-Element `message` während des Transformationsvorganges eine Bildschirmausgabe erzeugt.
Diese gibt zunächst den Inhalt des `Nachnamen` Knotens gefolgt von einem statischen Text aus.
Angewandt auf das Beispieldokument liefert es auf Kommandozeile die Ausgabe: `Obermüller hat mehr als einen Vornamen!`. Der Inhalt des Eingabedokuments wird unverändert kopiert.

Beispiel 73: Bedingte Verarbeitung durch Verwendung des if-Elements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:template match="Person">
(6)     <Mitarbeiter>
(7)             <xsl:if test="count(Vorname)&gt;1">
(8)                     <xsl:message><xsl:value-of select="Nachname"/> hat mehr als einen Vornamen!
</xsl:message>
(9)             </xsl:if>
(10)            <xsl:apply-templates/>
(11)    </Mitarbeiter>
(12)</xsl:template>
(13)
(14)<xsl:template match="Qualifikationsprofil"/>
(15)
(16)<xsl:template match="attribute::*|node()">
(17)    <xsl:copy>
(18)            <xsl:apply-templates select="attribute::*|node()"/>
(19)    </xsl:copy>
(20)</xsl:template>
(21)
(22)</xsl:transform>
```

Download des Beispiels
Download der Ergebnisdatei

In Erweiterung der simplen Selektion bildet choose die Möglichkeiten einer Mehrfachselektion, oder einer simplen if-then-else-Struktur ab.
(In Spezifikation nachschlagen)

Die Syntax lautet:

```
(1)<xsl:choose>
(2)     <!-- Content: (xsl:when+, xsl:otherwise?) -->
(3)</xsl:choose>
(4)<xsl:when test="Boole'scher Ausdruck">
(5)     <!-- Content: template -->
(6)</xsl:when>
(7)<xsl:otherwise>
(8)     <!-- Content: template -->
(9)</xsl:otherwise>
```

Das Beispiel zeigt die Transformation des Beispieldokuments in eine XHTML-Ausgabe.
Hierbei wird zunächst der XHTML-Dokumentrahmen bei Auftreten des Dokumentknotens (Suchmuster /) erzeugt. Nach dem öffnenden XHTML-Rumpfelement body werden innerhalb des Quelldokuments wahlfrei weitere, auf eines der angegebenen Suchmuster passende, Knoten gesucht (apply-templates-Aufruf).
Bei Auftreten des Knotens ProjektVerwaltung wird -- nach einigen Überschriftszeilen -- der Kopf einer XHTML-Tabelle, bestehend aus den Elementen table und der Kopfzeile (eingeschlossen durch tr), erzeugt. Den Rumpf der Tabelle schreibt ein anderes Template. Dieses wird jedoch nicht direkt aufgerufen, sondern der Prozeß der Ermittlung neuer „passender“ Knoten neu initiiert; jedoch diesmal, durch das select-Attribut des apply-templates-Elements, nur auf Knoten des Typs Person beschränkt.
Die Ersetzungsregel für Person speichert zunächst den Inhalt des Attributs PersID in einer Variable. Anschließend werden die Tabellenelemente der in der Ersetzungsregel für ProjektVerwaltung geöffneten Tabelle geschrieben. Hierzu werden nacheinander die Ersetzungsmuster für Knoten des Typs Vorname bzw. Nachname, die Kindknoten des aktuellen Knotens sind (die PersID des aktuellen Personen-Knotens findet sich in der zuvor belegten Variable), aktiviert.
Ein zweites Tabellenelement enthält einen XHTML-Hyperlink zu den durch den Mitarbeiter bearbeiteten Projekten. Als Sprungziel wird hierbei der Inhalt des Attributs mitarbeitInProjekt eingetragen.
Das Ersetzungsmuster Vorname übernimmt durch value-of die in einen Zeichenkettenwert gewandelten Inhalte des Elements. Zusätzlich existiert ein zweites Ersetzungsmuster für Knoten des Typs Vorname, das jedoch durch ein Prädikat nur auf das zweite und alle folgenden Elemente dieses Typs angewendet wird. Es gibt vor der Übernahme des Elementinhaltes ein Leerzeichen aus.
Das Element Nachname bettet den Elementinhalt in eine benannte Sprungreferenz ein. Zur Erzeugung der dokumentweiten eindeutigen Benennung wird die Funktion generate-id herangezogen, die für jedes Element des Information Sets des Eingabedokuments einen eindeutigen Bezeichner liefert.
Nach Abschluß der Tabellengenerierung im durch den Lokalisierungspfad ProjektVerwaltung gekennzeichneten Template werden durch den Aufruf des benannten Ersetzungsmusters wasteSpace 25 Leerzeilen, jeweils gefüllt mit der Zeichenkette „...“, erzeugt. Als Besonderheit ist in diesem Muster sehr deutlich die Nutzung der Rekursion zur Modellierung einer Schleife zu sehen.
Zum Abschluß wird nochmals eine Tabelle, nun mit den Mitarbeitern und ihren Projekten, erzeugt.

Beispiel 74: Erzeugung eines XHTML-Reports

```
(1)<?xml version="1.0"?>
(2)<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(3)
(4)<xsl:output method="xml" encoding="ISO-8859-1" omit-xml-declaration="no"  indent="yes"
(5)             xmlns="http://www.w3.org/1999/xhtml" />
(6)
(7)<xsl:template match="/">
(8)     <html>
(9)             <head>
(10)                    <title>XML-Vorlesung Sommersemester 2001 -- Projektverwaltung</title>
(11)            </head>
(12)            <body>
(13)                    <xsl:apply-templates/>
(14)            </body>
(15)    </html>
(16)</xsl:template>
(17)
(18)<xsl:template match="ProjektVerwaltung">
(19)    <center><h1>Projektverwaltung</h1></center>
(20)
(21)    <h2>Projekte</h2>
(22)    <table border="1">
(23)            <tr>
(24)                    <td><b>Projektnummer</b></td>
(25)                    <td><b>Projektleiter</b></td>
(26)            </tr>
(27)            <xsl:apply-templates select="Projekt"/>
(28)    </table>
(29)
(30)    <pre>
(31)            <xsl:call-template name="wasteSpace">
(32)                    <xsl:with-param name="maxLines">25</xsl:with-param>
(33)                    <xsl:with-param name="curLines">0</xsl:with-param>
(34)            </xsl:call-template>
(35)    </pre>
(36)
(37)    <h2>Mitarbeiter</h2>
(38)    <table border="1">
(39)            <tr>
```

```
(40)                    <td><b>Name</b></td>
(41)                    <td><b>Projekt</b></td>
(42)            </tr>
(43)            <xsl:apply-templates select="Person"/>
(44)    </table>
(45)</xsl:template>
(46)
(47)    <xsl:template name="wasteSpace">
(48)            <xsl:param name="maxLines"/>
(49)            <xsl:param name="curLines"/>
(50)            <xsl:if test="number($curLines) &lt; number($maxLines)">
(51)<xsl:text>...
(52)</xsl:text>
(53)                    <xsl:call-template name="wasteSpace">
(54)                            <xsl:with-param name="maxLines"><xsl:value-of select="$maxLines"/></
xsl:with-param>
(55)                            <xsl:with-param name="curLines"><xsl:value-of select="$curLines +
1"/></xsl:with-param>
(56)                    </xsl:call-template>
(57)            </xsl:if>
(58)    </xsl:template>
(59)
(60)<xsl:template match="Projekt">
(61)    <xsl:variable name="prjLeiter" select="@Projektleiter"/>
(62)    <tr>
(63)            <td>
(64)                    <xsl:element name="a">
(65)                            <xsl:attribute name="name"><xsl:value-of select="@ID"/></xsl:
attribute>
(66)                            <xsl:value-of select="@ID"/>
(67)                    </xsl:element>
(68)            </td>
(69)            <td>
(70)                    <xsl:element name="a">
(71)                            <xsl:attribute name="href">#<xsl:value-of select="generate-id(//
Person[@PersID=$prjLeiter]/Nachname)"/></xsl:attribute>
(72)                            <xsl:value-of select="//Person[@PersID=$prjLeiter]/Nachname"/>
(73)                    </xsl:element>
(74)            </td>
(75)    </tr>
(76)</xsl:template>
(77)
(78)<xsl:template match="Person">
(79)    <xsl:variable name="persNr" select="@PersID"/>
(80)    <tr>
(81)            <td><xsl:apply-templates select="Vorname[parent::Person/@PersID=$persNr]"/> 
(82)                     <xsl:apply-templates select="Nachname[parent::Person/@PersID=$persNr]"/></
td>
(83)            <td>
(84)                    <xsl:element name="a">
(85)                            <xsl:attribute name="href">#<xsl:value-of
select="@mitarbeitInProjekt"/></xsl:attribute>
(86)                            <xsl:value-of select="@mitarbeitInProjekt"/>
(87)                    </xsl:element>
(88)            </td>
(89)    </tr>
(90)</xsl:template>
(91)
(92)<xsl:template match="Vorname">
(93)    <xsl:value-of select="."/>
(94)</xsl:template>
(95)
(96)<xsl:template match="Vorname[position() &gt; 1]">
(97)    <xsl:text> </xsl:text><xsl:value-of select="."/>
(98)</xsl:template>
(99)
(100)<xsl:template match="Nachname">
(101)   <xsl:element name="a">
(102)           <xsl:attribute name="name"><xsl:value-of select="generate-id()"/></xsl:attribute>
(103)           <xsl:value-of select="."/>
(104)   </xsl:element>
(105)</xsl:template>
(106)
(107)<xsl:template match="text()"/>
(108)
(109)</xsl:stylesheet>
```

Download des Beispiels
Download der Ergebnisdatei

Die nachfolgende XSLT-Transformation ermittelt zu jedem beliebigen Element- und Attributknoten (Muster: * bzw.

@*) die zugehörige Namensraum-URI (Funktion namespace-uri) und gibt sie gemeinsam mit dem Namen des Knotens (Funktion name) in einer XML-formatierten Ausgabe aus.
Zur Neuselektion aller weiteren Element- und Attributknoten wird apply-templates mit dem Musterausdruck @*|node() ausgeführt, um in die Prüfung nach weiteren passenden Knoten (node()) auch explizit alle beliebigen Attribute (@*) einzubeziehen. Standardmäßig ermittelt apply-templates weitere passende Knoten nur innerhalb der Elemente eines Dokuments.

Beispiel 75: Ausgabe Namensräume jedes Elements und Attributs eines beliebigen XML-Dokuments

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<xsl:transform version="1.0"
(3)   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(4)
(5)<xsl:output method="xml" encoding="ISO-8859-1" omit-xml-declaration="no"  indent="yes" />
(6)
(7)<xsl:template match="*">
(8)     <xsl:element name="element">
(9)             <xsl:attribute name="name"><xsl:value-of select="name()"/></xsl:attribute>
(10)            <xsl:attribute name="namespace"><xsl:value-of select="namespace-uri()"/></xsl:
attribute>
(11)       <xsl:apply-templates select="@*|node()"/>
(12)    </xsl:element>
(13)</xsl:template>
(14)
(15)<xsl:template match="@*">
(16)    <xsl:element name="attribute">
(17)            <xsl:attribute name="name"><xsl:value-of select="name()"/></xsl:attribute>
(18)            <xsl:attribute name="namespace"><xsl:value-of select="namespace-uri()"/></xsl:
attribute>
(19)       <xsl:apply-templates select="@*|node()"/>
(20)    </xsl:element>
(21)</xsl:template>
(22)
(23)<xsl:template match="text()"/>
(24)
(25)</xsl:transform>
```

Download des Beispiels

Web-Referenzen 13: Weiterführende Links

- XSLT v1.0 Spezifikation
- XSLT @ jeckle.de
- Holman, K. G.: *What is XSLT?*
- Saxon -- ein freier XSLT-Prozessor
- Apache Xalan -- ein Open Source XSLT-Prozessor
- Literatur zum Thema

Übung 4: Textuelle Aufbereitung der Baumstruktur eines XML-Dokuments

Schreiben Sie eine XSLT-Transformation, die es erlaubt die Elementstruktur beliebiger wohlgeformter XML-Dokumente in Form eines „ASCII-Baums“ (siehe Beispiel) am Bildschirm auszugeben.
Beispiel:
Eingabe:

```
(1)<root>
(2)     <childX>
(3)             <childY1/>
(4)             <childY2/>
(5)     </childX>
(6)     <childX2/>
(7)</root>
```

Ausgabe:

```
(1)root
(2)+--childX
(3)     +--childY1
(4)     +--childY2
(5)+--childX2
```

2.4 Erzeugung von Präsentationssichten: XML Stylesheets

Mit XML können zwar beliebige Inhalte durch eigene Vokabulare ausgedrückt werden, jedoch bleibt die Präsentationssicht -- zumeist -- unberücksichtigt.
Lediglich für die standardisierte XML HyperText Markup Language (XHTML) ist eine Präsentationssicht eindeutig durch die Semantik der Sprachelemente definiert. Für alle anderen XML-Sprachen gibt es eine solche zunächst nicht, auch

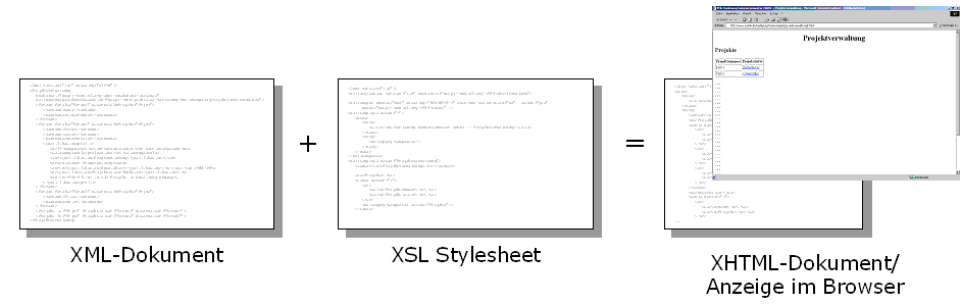
wenn einzelne Browser und Anzeigewerkzeuge eine zumeist baumartig orientierte graphische Ansicht bieten.

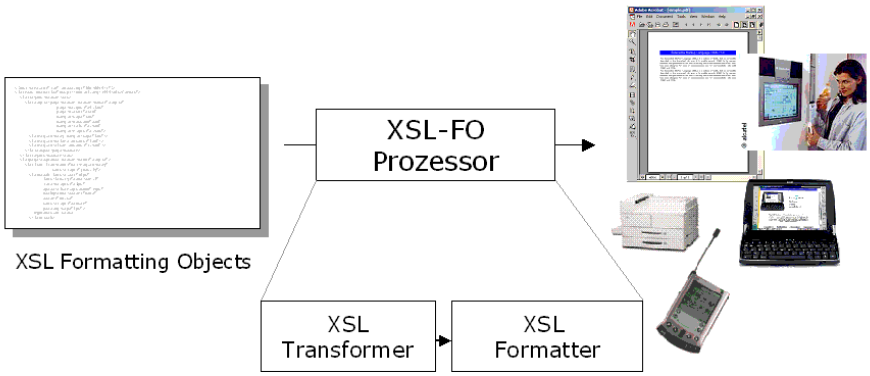
Zur Erzeugung einer Ansicht eines XML-Dokuments definiert das W3C derzeit die Extensible Stylesheet Language (XSL), eine XML-Sprache zur Formulierung beliebiger Präsentationssichten auf XML-Dokumente.
Dieser Ansatz verfolgt die Linie, Präsentationsinformation und Inhaltsinformation nicht in einem Dokument zu mischen, sondern beide Informationsarten physisch getrennt zu verwalten. Intuitiv wird der Vorzug dieser Idee klar; so können auf denselben Inhalt verschiedene Präsentationssichten angewendet werden, ebenso erlaubt die Abkopplung der Darstellungsinformation deren Wiederverwendung für verschiedenste Inhalte.

Der Gedanke der Trennung von Darstellung und inhaltlicher Information wurde bereits in „späteren“ HTML-Versionen (ab 1996) durch die *Cascading Style Sheets* (CSS) realisiert. Sie sind, als Ergänzung der HTML, in einer proprietären (d.h. nicht XML) Syntax realisiert um einerseits die Layoutmöglichkeiten von HTML zu erweitern, und zusätzlich wiederkehrende Präsentationsinformation an einem zentralen Ort ablegen zu können.

Konzeptionell teilt sich die XML Stylesheet Language in zwei Bereiche: Eine Transformationssprache und ein XML-Vokabular zur Darstellung von medienunabhängigen Formatierungsanweisungen (*formatting objects*).
Per Anwendungsszenario sind die beiden Teile eng verknüpft. Die durch XSL definierten Formatierungsobjekte sind ohne Nutzdaten (die zu formatierende Information) weitestgehend nutzlos; die notwendigen Nutzdaten müssen naturgemäß auch als XML-Dokument vorliegen. Die Anreicherung der Nutzdaten um XML-formulierte Formatierungsobjekte reduziert sich somit auf eine Transformation der XML-Nutzdaten in ein XML-Dokument, das neben den (evtl. modifizierten) Nutzdaten auch Formatierungsobjekte enthält.
Wegen ihrer allgemeinen Einsetzbarkeit wurde die Transformationssprache im Laufe des Standardisierungsprozesses vom Formatierungsvokabular abgetrennt, und seither als eigenständiger Standard (*XSL Transformations*) weiterentwickelt, der im folgenden Kapitel betrachtet wird.
Dieses Kapitel beschränkt sich daher ausschließlich auf die Betrachtung der *XSL formatting objects*.

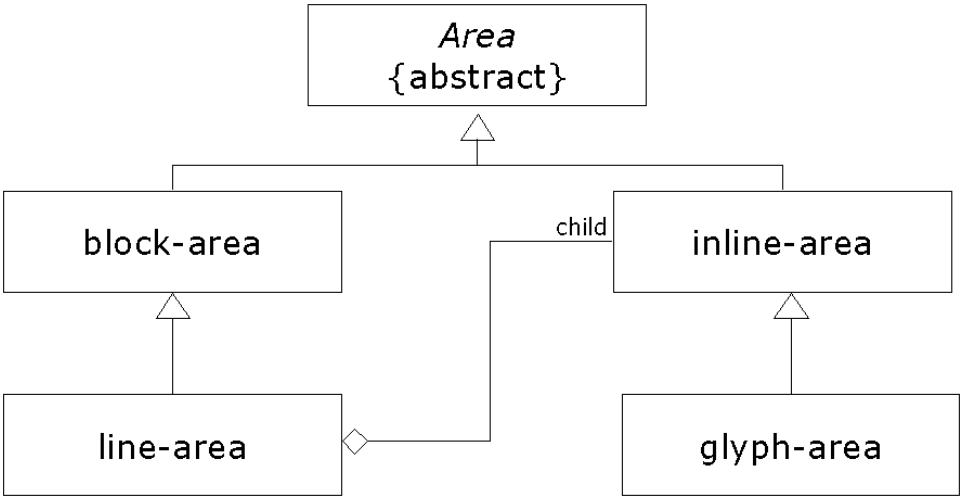
Die Abbildung faßt die beiden prinzipiellen Anwendungsszenarien zusammen. Im oberen Teilbild ist die Nutzung von XSL in Verbindung mit einem Dokument einer beliebigen XML-Sprache dargestellt. Als Resultat erzeugt der Browser-interne XSL-Prozessor eine XHTML-Repräsentation, die direkt zur Anzeige gebracht wird. Dieses Szenario wird im direkt anschließenden Kapitel vertieft.
Die untere Bildhälfte zeigt einige der Möglichkeiten der formatting objects für beliebige Zielrepräsentationen, beispielsweise Adobes *portable document format* (PDF) oder LaTeX. Der in den eingesetzten Prozessor integrierte Transformationsteil erzeugt zunächst die Baumrepräsentation der Zielstruktur, die anschließend zusätzlich durch den *XSL formatter* in das gewünschte physische Format umgesetzt wird. Diese Komponente übernimmt hierbei die Funktion einer *rendering engine*.





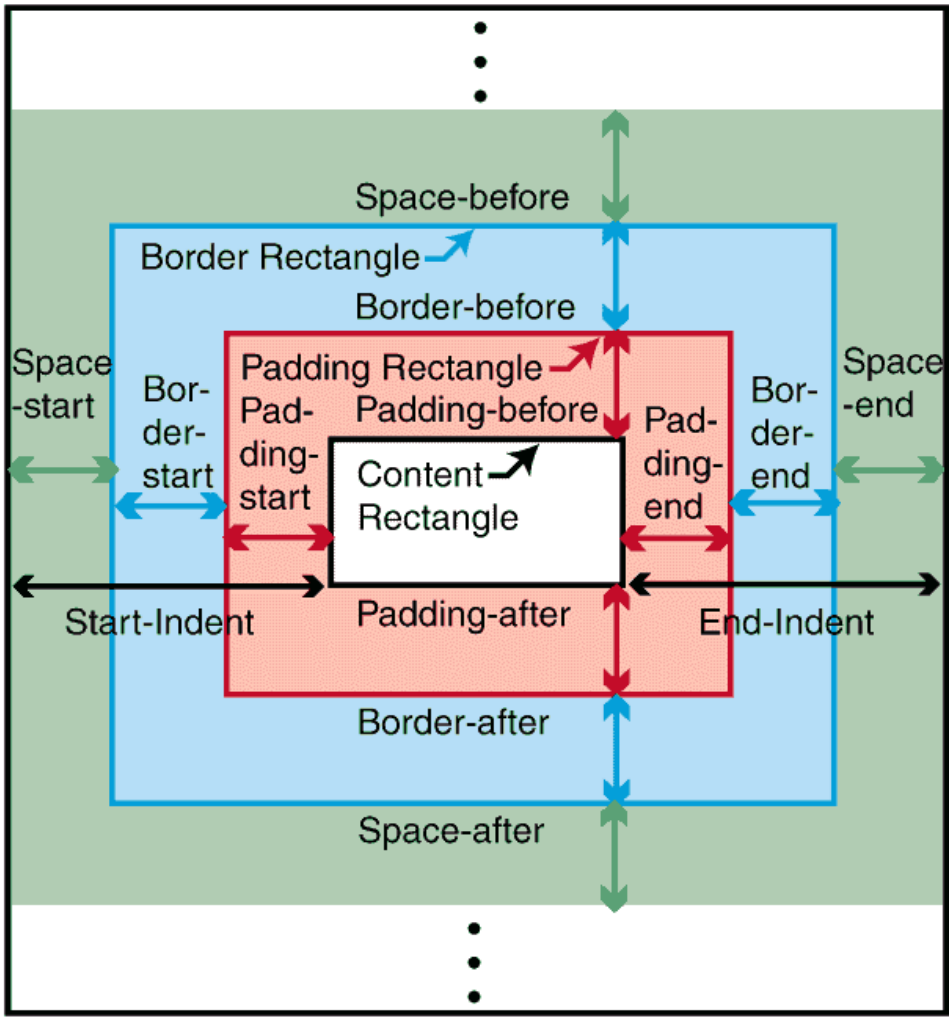
Für XSL-FO existiert keinerlei direkte Browserunterstützung. Dies erklärt sich einerseits in der Komplexität der Spezifikation, deren Umfang die üblichen Darstellungsmöglichkeiten gängiger Web-Browser bei weitem übersteigt und andererseits in dem Hauptanwendungsgebiet des Vokabulars: der (rechnerunabhängigen) Publikation von Texten. Konzeptionell orientiert sich XSL-FO daher eher am klassischem Buchdruck, als an den Cascading Style Sheets.

Geprägt durch das Anwendungsfeld arbeitet XSL-FO generell seitenorientiert. Daher werden die darzustellenden Inhalte gemäß anwenderdefinierter Vorgaben auf die erzeugten Seiten verteilt. Hierzu werden die verschiedenen graphischen Inhalte durch *Bereiche* (*areas*) umschlossen.
Die Abbildung zeigt die vier verschiedenen Grundtypen und ihr Verhältnis zueinander.

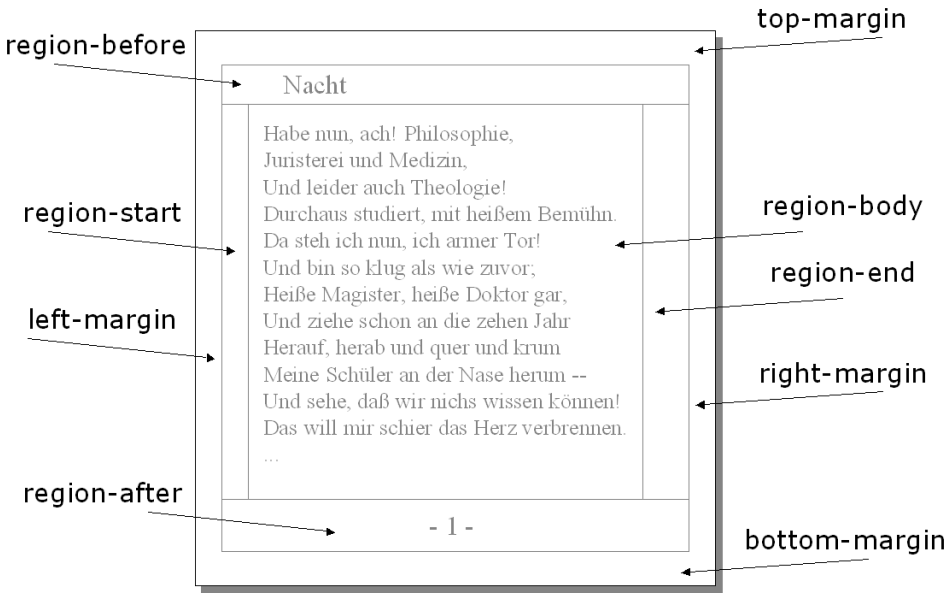


Ein Bereich kann entweder block- oder inline-Bereiche enthalten, jedoch nicht beide gleichzeitig. Block-Bereiche stellen die allgemeinste Variante der Inhaltsdarstellung zur Verfügung. Sie definieren einen rechteckigen durch Ränder umschlossenen Bereich zur Aufnahme von Textelementen wie: Absätzen, Überschriften oder Bildunterschriften (in XSL-Spezifikation nachschlagen).
Inline-Bereiche können zur graphischen Hervorhebung der umschlossenen Textbereiche, beispielsweise durch Farbunterlegung, verwendet werden. (in XSL-Spezifikation nachschlagen).
Als Spezialisierung der Block-Bereiche erlauben Line-Bereiche die Darstellung von Texten ohne umgebende Ränder.
Die feinste Darstellungsgranularität offerieren die Glyph-Bereiche, welche genau einen Buchstaben oder ein Zeichen kapseln.

Die nachfolgende Abbildung zeigt einen Bereich, mit einer Anzahl möglicher Attribute, zur Steuerung seiner Positions- und Abstandseigenschaften.
Der mit *content rectangle* bezeichnete Bereich nimmt die Nutzdaten (einzelner Buchstabe, einzelne Zeile oder mehrzeiliger Text) auf.
Direkt umschließend folgen *padding*- und *border*-Bereich.
Durch die Dimensionen von *padding* kann eine genaue Positionierung des Rahmenbereichs (border) erreicht werden.
Optional kann der Rahmenbereich gefüllt oder die Rahmenlinie in verschiedensten Varianten angezeigt werden.
Die *space*-area definiert einzuhaltende nichtbedruckbare Abstandsflächen zum nächstliegenden plazierten Bereich.
Hierdurch lassen sich zu enge Positionierungen oder gar Überschneidungen Verhindern.



Struktur eines XSL-FO-Dokuments: Spezifikationsgemäß befinden sich alle Elemente und Attribute eines XSL-FO-Dokuments im Namensraum `http://www.w3.org/1999/XSL/Format`.
Das Wurzelelement jedes XSL-FO-Dokuments bildet das Element `root`, es umschließt die einzelnen Definitionen und zu formatierenden Nutzdaten.
Die Spezifikation legt weiter fest, daß im `root`-Element zwingend die Kindknoten `layout-master-set` und `page-sequence` anzugeben sind.
Hierbei enthält das Element `layout-master-set` eine Art „Schablone“ aller Seiten der zu erzeugenden Ausgabe. Innerhalb dieses Elements werden die Seitenmaße festgelegt, sowie weitere wiederkehrende Regionen dimensioniert. Die nachfolgende Abbildung zeigt diese im Überblick.



Innerhalb des zweiten zwingend als Kindelement von `root` zu spezifizierenden Elements *page-sequence* werden die einzelnen zu plazierenden Bereiche definiert. Das Element *page-sequence* nimmt eine Reihe von Seiten auf, beispielsweise ein Kapitel. Die graphische Aufbereitung der Seite wird durch einen *page-master* oder einen *page-sequence-master*, der durch das `master-name`-Attribut referenziert wird, festgelegt. (Informationen zur *page-sequence* in der XSL-Spezifikation)
Die darzustellenden Inhalte (Text, Tabellen, Graphik, etc.) werden durch *flow*-Elemente umschlossen. Innerhalb eines *flow*-Elements sind die verschiedenen Inhaltselemente plaziert.
Das Beispiel zeigt ein erstes formatting objects-Dokument. Der *page-master* definiert eine DIN A4 Seite mit den bekannten Abmessungen. Ferner wird die Größe des Kopfzeilenbereichs (*region-before*) auf mindestens (`extent`-Attribut) 3 cm festgelegt. Dergleichen geschieht für den Fußzeilenbereich (*region-after*).
Innerhalb der *page-sequence*, welche die im Seiten-Master festgelegten Dimensionscharakteristika übernimmt (das `master-name`-Attribut enthält die Identifikation des *page-master*s), wird innerhalb eines *flow*-Elements ein Text-`block` definiert.
Der zentriert gesetzte Text (`text-align="center"`) wird in einer serifenlosen Schrifttype (`font-family="sans-serif"`) der Größe 12 Punkt (`font-size="12pt"`) gesetzt. Zusätzlich wird eine Zeilenhöhe von 15 Punkten (`line-height="15pt"`) mit einem zusätzlichen Abstand von 3 Punkten (`space-after="3pt"`) zum nächst folgenden Element festgelegt.
Als Inhalt des `block`-Elements ist der darzustellende Text plaziert.

Beispiel 76: Ein erstes XSL-FO-Dokument

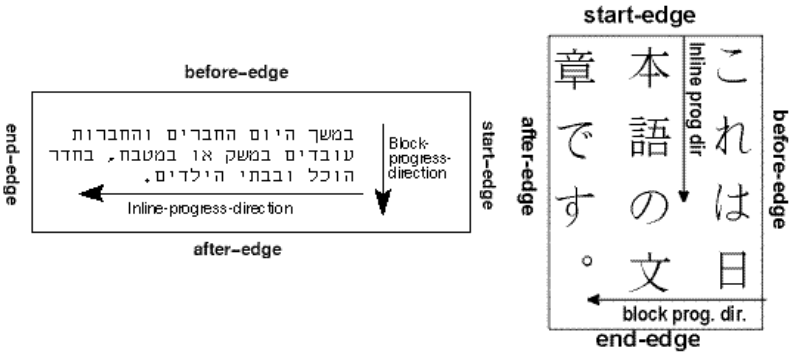
```
 <?xml
version="1.0" encoding="utf-8"?> <fo:root
xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <fo:layout-master-set>
      <fo:simple-page-master master-name="simple"
            page-height="29.7cm"
            page-width="21cm"
            margin-top="1cm"
            margin-bottom="2cm"
            margin-left="2.5cm"
            margin-right="2.5cm">
         <fo:region-body margin-top="3cm"/>
         <fo:region-before extent="3cm"/>
         <fo:region-after extent="1.5cm"/>
      </fo:simple-page-master>
   </fo:layout-master-set>
   <fo:page-sequence master-name="simple">
      <fo:flow flow-name="xsl-region-body">
         <fo:block font-size="12pt"
            font-family="sans-serif"
            line-height="15pt"
            space-after="3pt"
            text-align="center">
               Hello World - mein erstes XSL-FO-Dokument!
         </fo:block>
      </fo:flow>
   </fo:page-sequence>
</fo:root>
```

Download des Beispiels
Download der Ergebnisdatei

Textfluß und Objektplazierung
Zur Unterstützung der Erzeugung von Dokumenten mit geänderter Leserichtung, wie im nahen Osten und arabischen Sprachraum üblich, erlaubt XSL-FO ihre explizite Definition durch das Element `writing-mode`.
Darüberhinaus kann dieses Element auch die Anordnungsreihenfolge der einzelnen Symbole steuern, auf diesem Wege lassen sich auch asiatische Texte -- von „oben nach unten“ -- erzeugen.
Die gültigen Belegungen des `writing-mode`s lauten:

- lr-tb: Von links nach rechts, darin von oben nach unten
  Vorgabebelegung und übliches westliches Anordnungsschema
- rl-tb: Von rechts nach links, darin von oben nach unten
- tb-rl: Von oben nach unten, darin von rechts nach links (Japanisch)
- lr: Nur von links nach rechts; keine weitere Angabe
- rl: Nur von rechts nach links; keine weitere Angabe
- tb: Nur von oben nach unten; keine weitere Angabe

Die drei letzten Belegungen übernehmen den fehlenden Wert von ihrem direkt umgebenden Element.
Die nachfolgende Abbildung zeigt Beispiele der Anwendung des `writing-mode`-Elements (zum `writing-mode` in der XSL-Spezifikation nachschlagen).
Man beachte die geänderten Plazierungen der *edges*! So wird im hebräischen Dokument -- gemäß der getauschten Lesrichtung der Zeilen -- auch die dem Block folgende *end-edge* links ans Ende(!) des Absatzes verschoben. Dasselbe gilt spiegelsymmetrisch für die *start-edge*.
Im japanischen Dokument wird die Positionen der *start*- und *after-edge* ebenfalls verschoben, um wieder die bekannte Semantik (*start-edge* geht in Lesrichtung dem Block voraus, *after-edge* folgt ihm) zu erhalten.



Erzeugung von Tabellen
Tabellen werden als blockartige Objekte innerhalb eines `flow`-Bereichs definiert. Sie stellen ein eigenes Element dar und bedürfen daher keiner Kapselung durch ein `block`-Element.
Der prinzipiell Aufbau lautet:

```
<table>
  <table-column>...</table-column>
  <!-- weitere table-column-Elemente; für jede Tabellenspalte -->
  <table-body>
     <table-row>
        <table-cell>...</table-cell>
        <!-- weitere Tabelleneinträge -->
     </table-row>
  </table-body>
</table>
```

`table-column` legt spaltenspezifische Charakteristika für alle Tabellenzellen der Spalte fest. Hierzu zählt beispielsweise die im angegebenen Code verwende Spaltenbreite; definiert durch das Attribut `column-width`.
Innerhalb jeder Tabellenzelle stehen wieder alle `block`-Elemente (Textblöcke, Listen, Tabellen) zur Verfügung.
Das Beispiel nutzt ferner die beiden Attribute `border-width` und `border-style` zur Definition der Rahmenlinienstärke bzw. ihres Aussehens.

Beispiel 77: Erzeugung einer Tabelle

```
 <?xml
version="1.0" encoding="utf-8"?>

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="myMaster"
                           page-height="29.7cm"
                           page-width="21cm"
                           margin-top="1cm"
                           margin-bottom="2cm"
                           margin-left="2.5cm"
                           margin-right="2.5cm">
      <fo:region-body margin-top="3cm"/>
      <fo:region-before extent="3cm"/>
```

```
      <fo:region-after extent="1.5cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-name="myMaster">
  <fo:flow flow-name="xsl-region-body">
      <fo:block font-size="16pt"
            font-family="sans-serif"
            space-after.optimum="15pt"
            text-align="center">
         Eine Tabelle ...
      </fo:block>

     <fo:table>
       <fo:table-column column-width="50mm"/>
       <fo:table-column column-width="50mm"/>
       <fo:table-column column-width="50mm"/>
       <fo:table-body>
         <fo:table-row>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x11</fo:
block></fo:table-cell>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x12</fo:
block></fo:table-cell>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x13</fo:
block></fo:table-cell>
         </fo:table-row>
         <fo:table-row>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x21</fo:
block></fo:table-cell>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x22</fo:
block></fo:table-cell>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x23</fo:
block></fo:table-cell>
         </fo:table-row>
         <fo:table-row>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x31</fo:
block></fo:table-cell>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x32</fo:
block></fo:table-cell>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x33</fo:
block></fo:table-cell>
         </fo:table-row>
         <fo:table-row>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x41</fo:
block></fo:table-cell>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x42</fo:
block></fo:table-cell>
           <fo:table-cell border-width="0.5mm"          border-style="solid"><fo:block>x43</fo:
block></fo:table-cell>
         </fo:table-row>
       </fo:table-body>
     </fo:table>
   </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Download des Beispiels
Download der Ergebnisdatei

Einbindung von Graphiken

XSL-FO erlaubt die Einbindung von Graphiken in beliebigen Formaten; die Unterstützung konkreter Formate ist jedoch von der gewählten *rendering engine* abhängig.
Das nachfolgende Beispiel zeigt die Einbindung einer GIF-Graphik durch das Element `external-graphic` welches über eine URI die physische Lokation der einzubindenden Bilddatei angibt (in XSL-Spezifikation nachschlagen). Ferner zeigt das Beispiel die Nutzung des `inline`-Elements um innerhalb eines Blockes Formatierungsinformation (im Beispiel auf Wortebene) anzubringen.

Beispiel 78: Einbinden einer Graphik

```
 <?xml
version="1.0" encoding="utf-8"?> <fo:root
xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <fo:layout-master-set>
      <fo:simple-page-master master-name="simple"
            page-height="29.7cm"
            page-width="21cm"
            margin-top="1cm"
            margin-bottom="2cm"
            margin-left="2.5cm"
            margin-right="2.5cm">
        <fo:region-body margin-top="3cm"/>
        <fo:region-before extent="3cm"/>
```

```
            <fo:region-after extent="1.5cm"/>
         </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-name="simple">
         <fo:flow flow-name="xsl-region-body">
            <fo:block font-size="12pt"
               font-family="Helvetica"
               line-height="15pt"
               space-after="3pt"
               text-align="center">
                  Text <fo:inline text-decoration="underline">vor</fo:inline> der Graphik ...
            </fo:block>

            <fo:block text-align="center">
               <fo:external-graphic src="file:sblumen.jpg"/>
            </fo:block>

            <fo:block font-size="12pt"
               font-family="Times"
               line-height="15pt"
               space-after="3pt"
               text-align="center">
                  Vincent van Gogh: <fo:inline font-style="italic">Sonnenblumen</fo:inline>
            </fo:block>

         </fo:flow>
      </fo:page-sequence>
   </fo:root>
```

Download des Beispiels
Download der Ergebnisdatei

XSL-FO offenbart sich als mächtiger und flexibler Mechanismus zur Erzeugung auch komplexer Dokumente mithilfe eines XML-Vokabulars. Jedoch zeigen die vorgestellten Beispiele, daß sich die Erstellung der notwendigen XSL-FO-Eingabedokumente mitunter sehr aufwendig gestaltet. Aus diesem Grunde haben sich XML-Sprachen wie das bekannte DocBook entwickelt, durch die diese Komplexität nochmals gekapselt wird. Zusätzlich zu einem vom Schriftsatz abstrahierten Vokabular zur Erzeugung von beliebigen Dokumenten enthält DocBook auch eine Reihe von Transformationsbeschreibungen zur automatisierten Ableitung der XSL-FO-Dokumente. So operiert der DocBook-Verwender mit den Primitiven *Kapitel*, *Überschrift* oder *Bildunterschrift*, die dann durch eine XSLT-Datei nach XSL-FO zur Weiterverarbeitung übersetzt werden.

Web-Referenzen 14: Weiterführende Links
- Extensible Stylesheet Language (XSL) @ W3C
- Apache FOP -- eine freie XSL-FO-Implementierung
- XEP -- eine kommerzielle XSL-FO-Implementierung
- XSL-FO Tutorial @ RenderX
- Eisenberg, D. J.: *Using XSL Formatting Objects*, Part 1
- Eisenberg, D. J.: *Using XSL Formatting Objects*, Part 2
- XSL-FO Referenz @ Zvon.org
- DocBook
- Hintergrundinformation zu DocBook
- XSL @ Coverpages

2.5 Anfragesprachen

Mit der Sprache XPath wurde im Abschnitt 2.2 ein mächtiger Ansatz zur Lokalisierung beliebiger Knotenmengen innerhalb von XML-Dokumenten vorstellt. Konzeptionell stellt XPath eine durch Pfadausdrücke navigierende Sprache dar. Neben den Vorteilen, insbesondere der vergleichsweise einfachen Formulierung kleiner Anfragen, treten jedoch einige bemerkenswerte Nachteile zu Tage:

1. Das Ergebnis ist immer eine Element- oder Attributmenge,
   die Konstruktion neuer Ergebnistypen ist nicht möglich.
2. Dem Anwender muß die Struktur des Eingabedokuments bekannt sein.
3. Anfragen nach derselben Information müssen für strukturell verschiedene Eingaben i.A. unterschiedlich formuliert werden.
4. Entstehende Pfadausdrücke gewinnen schnell an Komplexität und damit Unübersichtlichkeit.

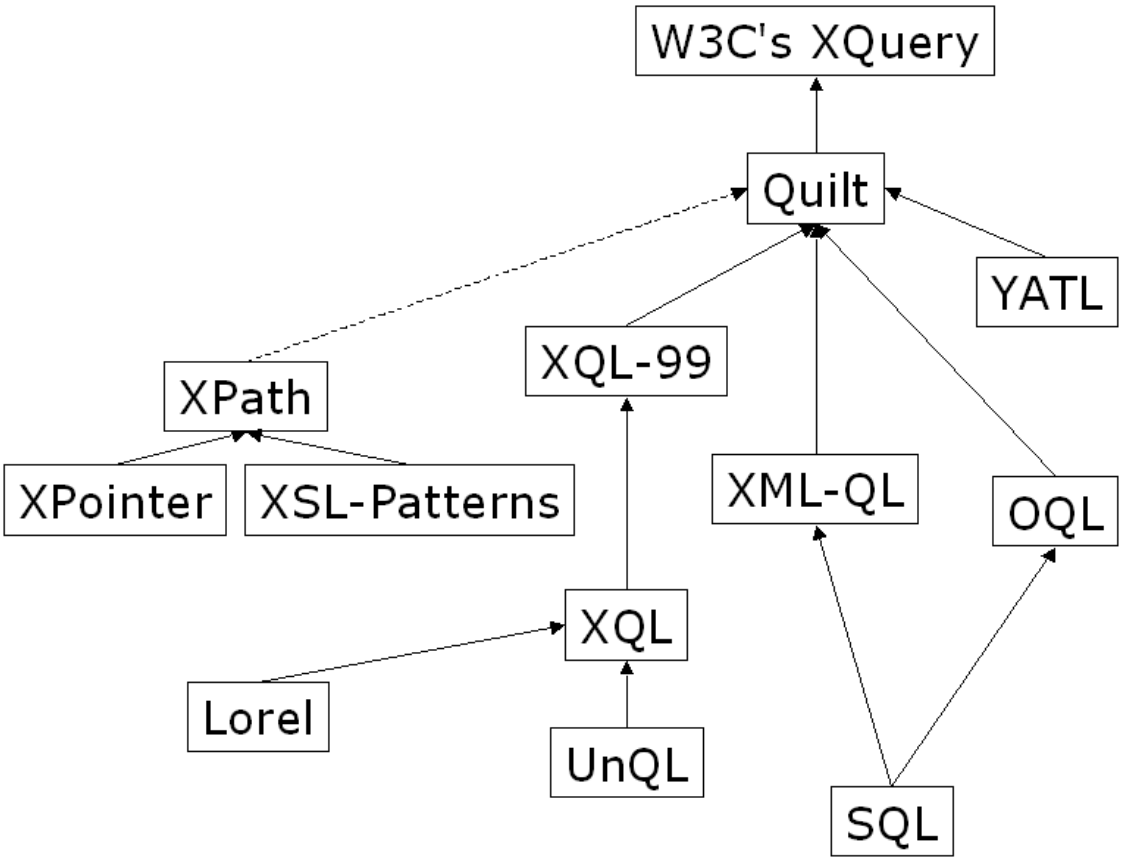
Aus diesen Gründen wurden im Laufe der XML-Entwicklung einige Anfragesprachen vorgeschlagen, welche diese Beschränkungen nicht aufweisen. Sammelbecken der verschiedensten Ideen zur Definition einer eigenständigen Anfragesprache für XML-Dokumente war der *Tandberg XML Query Workshop* 1998. Zahlreiche Hersteller, Forschungsinstitute und Hochschulen legten in Positionspapieren ihre Ideen zur Thematik dar. Hierbei reichte das Spektrum von halbseitigen Anforderungsdefinitionen (eher Wunschzetteln) bis hin zu ausgearbeiteten Sprachvorschlägen (wie XQL). Verständlichermaßen handelte es sich bei den meisten beitragenden Firmen um (bekannte) Hersteller von Datenbankmanagementsystemen, welche eine Anfragesprache für XML als natürliche Erweiterung ihrer (inzwischen durch XML angereicherten) „universellen“ Systeme sehen.
Prinzipiell kann jedoch eine XML-Anfragekomponente auf beliebigen XML-Eingaben, also auch auf reinen Dokumenten, operieren.

Anders als die in der XPath-Recommendation definierten Pfadausdrücke legen die vorgeschlagenen Anfragesprachen keine navigierenden Zugriffe zugrunde, sondern orientieren sich an den klassischen deskriptiven Anfragesprachen wie SQL.
Durch die unerwartet große Resonanz des Tandberg-Workshops motiviert initiierte das W3C die XML Query Working Group und beauftragte diese mit der Erarbeitung eines Sprachstandards, der mittlerweile als „XQuery“ in einem ersten Arbeitsstand (*working draft*) vorliegt.
Nachfolgend soll diesem zukünftigen Sprachstandard die Hauptaufmerksamkeit gewidmet werden. Betrachtungen anderer Anfragesprachen finden nur insofern statt, wie sie für das Verständnis von XQuery notwendig und dienlich sind.

Neben der Behebung der erwähnten Nachteile einer Pfad-basierten Anfrage wurden folgende Anforderungen an XQuery definiert:

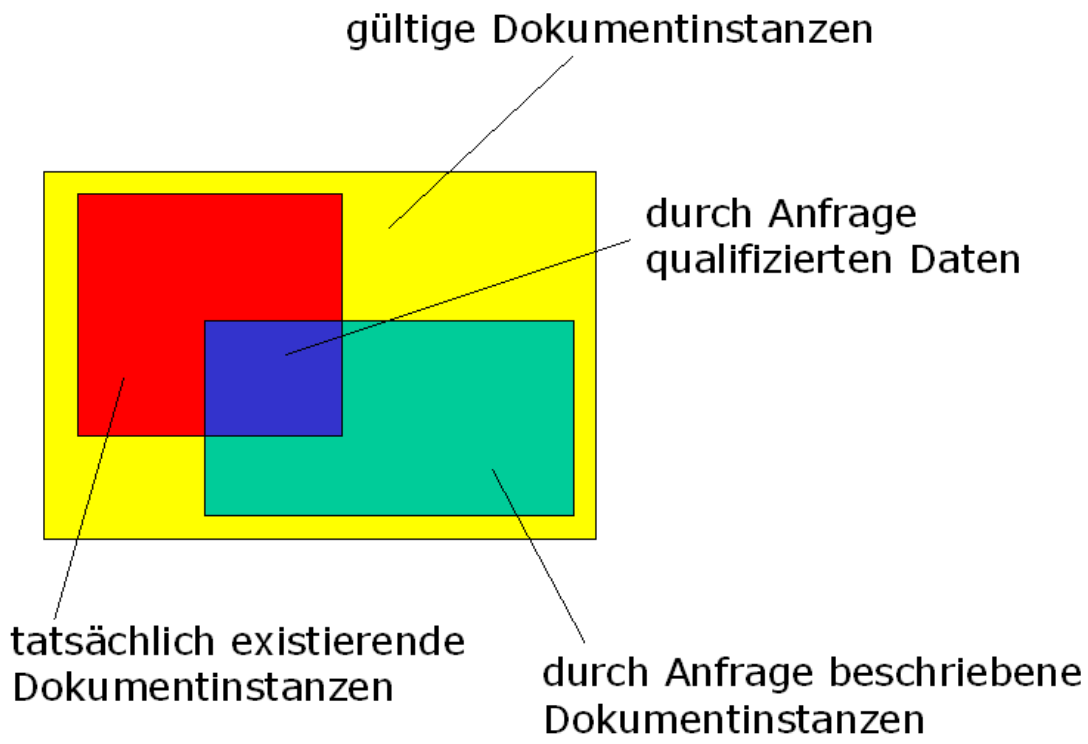
- Abgeschlossenheit.
  Jede XQuery-Anfrage liefert als Ergebnis ein XML-Dokument das im Bedarfsfalle als Grundlage neuer Anfragen herangezogen werden kann.
- Kombinierbarkeit von Funktionen.
  Durch die Anfragesprache angebotene Funktionen sind ohne Einschränkung miteinander kombinierbar.
- Seiteneffektfreiheit.
  Eine XQuery-Anfrage darf keine Seiteneffekte auf nicht in der Anfrage spezifizierte Datenbereiche aufweisen.
- Strenge Typisierung.
  Das Schema des XML-Ergebnisdokuments ist zur Anfrageanlysezeit ermittelbar.
- Schemaberücksichtigung.
  Operation auf allen Datentypen, die durch XML-Schema definiert werden.
- Relationale Vollständigkeit.
  Für jede durch einen Konstruktor erzeugbare Struktur existiert eine Zerlegungsmöglichkeit.

Die Abbildung stellt die Vorläufer, Entwicklungen und Einflüsse, die in die W3C-Initiative mündeten, dar.
Konzeptionell stellt sich XQuery, als deklarative Sprache und auch syntaktisch, in die Tradition der in den relationalen Datenbanken implementierten Sprache SQL; genaugenommen deren Anfrageteil der bekannten `SELECT ... FROM ... WHERE ...`-Struktur.
Die Berücksichtigung von SQL geht auf den Sprachvorschlag XML-QL zurück. Neben der Deklarativität weist XQuery eine funktionale Struktur auf, welche die beliebige Kombination von Teilausdrücken zu einer Anfrage erlaubt. Hierfür stand der ODMG-Sprachstandard OQL zur Anfrage auf objektorientierte Datenbanken Pate. Auch er hat sich als deklarative Anfragesprache aus SQL heraus entwickelt.
XQL markiert eine alternative Entwicklungslinie zur Formulierung der XML-Anfragesprache, sie entwickelt im wesentlichen die Pfadausdrücke der XPath-Syntax fort. Damit greift XQL frühere Entwicklungen aus dem Umfeld semi-strukturierter Datenverwaltung -- wie Lorel oder UnQL -- auf.
Beide Sprachstämme, sowie experimentelle Prototypen wie „YATL“, flossen in den Sprachvorschlag Quilt ein, der durch die W3C-Arbeitsgruppe zu XQuery weiterentwickelt wird.



Nicht zuletzt wegen der besseren Optimierbarkeit algebraisch fundierter Ausdrücke ist XQuery im Stile SQLs als deskriptive Anfragesprache realisiert. Dieser Sprachtyp definiert Bedingungen, denen ein gültiges Ergebnis genügen muß, läßt jedoch die Realisierung der Anfrageverarbeitung durch das System vollkommen offen. Hierunter sind insbesondere Auswertungsreihenfolge, Zugriffspfade und optimierende Techniken wie Indexstrukturen, etc. zu verstehen.
Die Abbildung 39 zeigt (angeleht an K. Dittrich) das Prinzip deskriptiver Anfragen. Gelb dargestellt ist die Menge der

möglichen XML-Dokumentinstanzen. Diese werden durch das zugrundeliegende Schema bestimmt. In der Praxis existiert eine -- rot symbolisierte -- Untermenge tatsächlich verwalteter Dokumente, die konform zum Schema formuliert sind.
Jede Anfrage (grünes Rechteck) beschreibt eine zweite Untermenge der gültigen Dokumentinstanzen. Anschaulich handelt es sich hierbei um diejenigen Dokumente, die die in der Anfrage formulierten Bedingungen erfüllen. Innerhalb der Schnittmenge aus existierenden Dokumenten und prinzipiell „passenden" Dokumentinstanzen befindet sich das Anfrageergebnis. Mithin diejenigen verwalteten Daten, die den Anfragebedingungen genügen.
Da die Menge der durch die Anfrage beschriebenen Dokumentinstanzen leer sein kann -- dies ist bei a priori „unmöglichen" Anfragen (etwa: „Zeige alle Dokumente, deren Element mit dem Namen *X* gleichzeitig *Y* heißt") der Fall -- muß sich nicht zwingend eine (nichtleere) Ergebnismenge ergeben. Die tatsächlich existierenden Dokumentinstanzen bilden lediglich bei Anfrageausführung auf XML-Dokumenten eine nichtleere Menge, da leere XML-Dokumente gemäß den Anforderungen an die Wohlgeformtheit nicht zulässig sind. Im Falle der Anfrageauswertung gegen die Inhalte einer XML-Datenbank -- XQuery spricht hierbei von *virtuellen Dokumenten* -- kann auch diese Eingangsmenge (bei unbefüllter Datenbank) leer sein.



Syntax einer XQuery-Anweisung:
Die Struktur einer XQuery Anweisung wird durch die FLWR- (sprich *flower*) Syntax beschrieben. Sie entspricht in ihrer Mächtigkeit der XQuery-Algebra. Syntaktisch verkörpert sie jedoch keineswegs die einzige Möglichkeit XQuery-Anfragen zu formulieren. Genaugenommen stellt die FLWR-Syntax nur eine Empfehlung einer möglichen Algebradarstellung vor. Weitere Syntaxen können jederzeit durch den Anwender definiert werden. Das W3C-Dokument XML Syntax for XQuery definiert beispielhaft eine XML-Syntax zu XQuery.

```
FLWRExpr    ::=  (ForClause | LetClause)+ WhereClause? "return" Expr

                 ForClause   ::=  "for" Variable "in" Expr ("," Variable "in" Expr)*

                 LetClause   ::=  "let" Variable ":=" Expr ("," Variable ":=" Expr)*

                 WhereClause ::=  "where" Expr Expr        ::=  extended XPath
```

Die entstehenden Ausdrücke sind durch das Auftreten der definierten Schlüsselworte `FOR`, `LET`, `WHERE` und `RETURN` strukturiert, die namensgebend für den Gesamtausdruck sind.
Nachfolgend werden ausgewählte Sprachbestandteile an Beispielen vorgestellt.

Beispiel 79 zeigt einen ersten XQuery-Ausdruck, der eine Menge vorgegebener Elemente umstrukturiert. Mittels der FOR-Klausel werden alle in der durch runde Klammern begrenzten Aufzählungsmenge angegebenen Elemente des Namens `Nachname` an die Variable `name` gebunden. Die RETURN-Klausel gibt die gebundenen Werte einzeln aus und bettet sie statisch in ein Element `Mitarbeiter` ein.
Im Ergebnis treten daher nicht nur die textuellen Wertinhalte der `Nachnamen`-Elemente innerhalb der durch die Abfrage erzeugten `Mitarbeiter`-Elemente auf, sondern die gesamten `Nachname`-Elemente einschließlich ihrer Start- und Endemarken.

Beispiel 79: Selektion mit der FOR-Klausel aus einer Menge (explizit) vorgegebener Werte

```
(1)FOR $name IN
(2)     (<Nachname>Hinterhuber</Nachname>, <Nachname>Obermüller</Nachname>,
(3)     <Nachname>Meier</Nachname>)
(4)RETURN
(5)<Mitarbeiter>
(6){$name}
(7)</Mitarbeiter>
```

Download des Beispiels
Download der Ergebnisdatei

Üblicherweise werden jedoch die abzufragenden Werte nicht explizit durch den Anfragenden vorgegeben, sondern werden im Verlauf der Anfrageauswertung durch den XQuery-Prozessor aus einem oder mehreren Dokumenten extrahiert.
Daher liegt den nachfolgenden Anfragebeispielen, sofern nicht anders angegeben das Projektverwaltungsdokument zugrunde.

Beispiel 80: XQuery-Anfrage (FR) die alle Personen liefert

```
(1)FOR $x IN document("projektverwaltung5.xml")//Person
(2)RETURN $x
```

Download des Beispiels
Download der Ergebnisdatei

Der Ausdruck des Beispiels 80 liefert alle Knoten des Typs `Person`. Hierzu werden eine oder mehrere durch XPath definierte Knotenmengen an Variablen gebunden. Im Beispiel ist dies die durch den erweiterten XPath `//Person` erreichbare Knotenmenge, die an die neu definierte Variable `$x` gebunden und im Anschluß durch das `RETURN`-Schlüsselwort ausgegeben wird.

Beispiel 81: XQuery-Anfrage der Form FWR

```
(1)FOR $y IN document("projektverwaltung5.xml")//
Person
(2)WHERE $y/Vorname = 'Hans'
(3)RETURN $y/Nachname
```

Download des Beispiels
Download der Ergebnisdatei

Die Anfrage des Beispiels 81 erweitert den vorhergehenden Ausdruck zunächst um eine `WHERE`-Bedingung. Sie prüft innerhalb der durch die Variablen `$y` repräsentierten Knotenmenge die Elemente des Typs `Vorname` auf den Inhalt `Hans`.
Für alle Knoten aus `$y` die diese Bedingung erfüllen, liefert die Anfrage das Element `Nachname`.

Ist eine sortierte Ausgabe der Resultate gewünscht, so kann der `RETURN`-Klausel ein Sortierausdruck der Form `SORTBY (Knotenmenge ascending|descending)` nachgestellt werten.
Beispiel 82 zeigt dies als modifizierte Erweiterung der vorhergehend diskutierten Anfrage.

Beispiel 82: Sortierung der Resultatmenge

```
(1)FOR $y IN document("projektverwaltung5.xml")//
Person
(2)RETURN $y/Nachname
(3)SORTBY (Nachname ascending)
```

Download des Beispiels
Download der Ergebnisdatei

Beispiel 83: XQuery-Anfrage der Form FLWR

```
(1)FOR $personen IN document("projektverwaltung5.xml")//Person
(2)LET $projekte := document("projektverwaltung5.xml")//
Projekt
(3)WHERE $projekte/@Projektleiter = $personen/@PersID
(4)RETURN $personen/Vorname
```

Download des Beispiels
Download der Ergebnisdatei

Beispiel 83 zeigt eine Anfrage unter Ausprägung aller vier Strukturelemente. Die Anfrage ermittelt die Vornamen aller Projektleiter.
Hierbei wird zunächst die zu durchsuchende Knotenmenge `//Person` an die Variable `$personen` gebunden. Innerhalb des `LET`-Blockes wird zusätzlich die Variable `$projekte` mit der durch den XPath erreichbaren Knotenmenge `//Projekt` identifiziert. Durch die `WHERE`-Klausel werden alle diejenigen Knoten in `$personen` selektiert, deren `PersID`-Attribut mit dem Inhalt eines `Projektleiter`-Attributs in `$personen` übereinstimmt.
Zurückgeliefert werden durch `RETURN` alle `Vorname`-Elemente aus `$personen`.
Der wesentliche Unterschied zwischen den in `FOR`- und den in `LET`-Strukturen auftretenden Knotenmengen liegt in ihrer Behandlung während der iterativen Abarbeitung der Anfrage. Während alle in `FOR` referenzierten Knoten

durchlaufen werden, wird die LET-Knotenmenge zu Beginn der Auswertung statisch ermittelt und keine Iteration über sie durchgeführt.
Beispiel 84 zeigt nochmals gesondert die verschiedene Auswertungssemantik von FOR und LET ohne dabei auf die Projektverwaltung als Eingabedokument zurückzugreifen.

Beispiel 84: Auswertung der FOR- und LET-Klausel

```
(1)FOR $NName in (<Nachname>Huber</Nachname>, <Nachname>Müller</Nachname>, <Nachname>Meier</
Nachname>)
(2)LET $VName := (<Vorname>Hans</Vorname>, <Vorname>Franz</Vorname>, <Vorname>Fritz</Vorname>)
(3)RETURN
(4)<Person>
(5)     <Name>
(6)             {$VName}
(7)             {$NName}
(8)     </Name>
(9)</Person>
```

Download des Beispiels
Download der Ergebnisdatei

Im Verlauf der Anfrageauswertung werden zunächst die drei Nachnamen-Elemente iterativ an die Variable NName gebunden, d.h. in jedem Durchlauf der Auswertungsschleife wird genau ein Wert NName zugewiesen. Die Berechnung eines Ergebnisses für NName erfolgt also in jedem Teildurchlauf der Auswertung. Anders gelagert ist das Verhalten von LET. Das Ergebnis dieser Klausel wird nur einmal innerhalb der Abfrage bestimmt und an die Variable VName gebunden, daher enthält VName auch nicht einen Einzelwert, sondern die Menge aller zugewiesenen Werte.
Das Ergebnis spiegelt dieses Verhalten wieder, es enthält nicht, wie intuitiv zu vermuten die paarweise Zuordnung der Werte, sondern das Kartesische Produkt der Menge der Nachnamen mit der als einelementiger Menge betrachteten Nachnamensammlung.

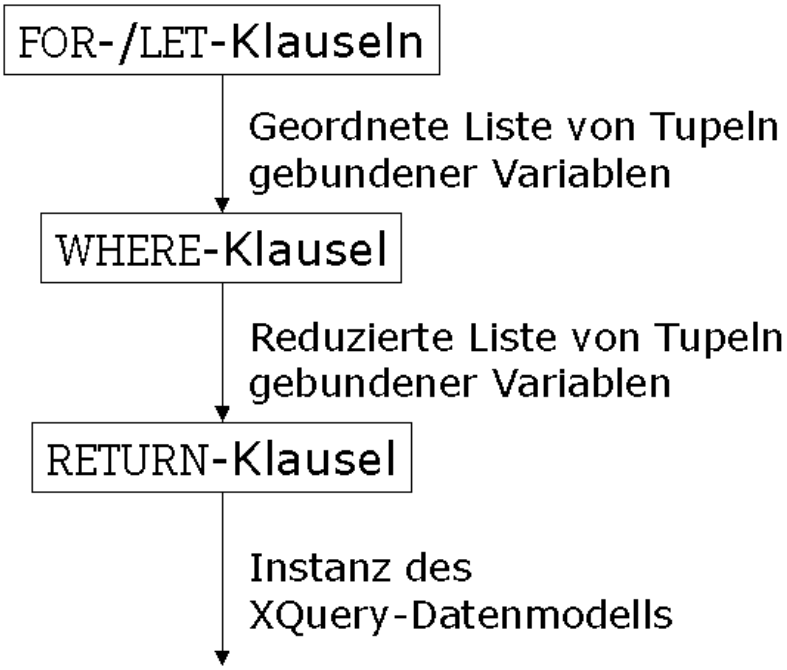
Um mehr als eine Knotenmenge iterativ bearbeiten zu können darf ein XQuery-Ausdruck mehrere FOR-Klauseln beinhalten. Das folgende Beispiel zeigt auf diesem Wege die (korrekte) Berechnung des kartesischen Produktes der in Beispiel 84 dargestellten Namen.

Beispiel 85: Berechnung des kartesischen Produktes zweier Mengen

```
(1)FOR $NName in (<Nachname>Huber</Nachname>, <Nachname>Müller</Nachname>, <Nachname>Meier</
Nachname>)
(2)FOR $VName in (<Vorname>Hans</Vorname>, <Vorname>Franz</Vorname>, <Vorname>Fritz</Vorname>)
(3)RETURN
(4)<Person>
(5)     <Name>
(6)             {$VName}
(7)             {$NName}
(8)     </Name>
(9)</Person>
```

Download des Beispiels
Download der Ergebnisdatei

Auswertungsreihenfolge einer XQuery-Anfrage: In den Beispielen klingt bereits die Auswertungsreihenfolge der einzelnen Anweisungsteile an, sowie die zwischen den Einzelschritten transportierte Information.
Die Berechnung des Ergebnisdokuments erfolgt in Reihenfolge der verschiedenen Klauseln. In einem ersten Schritt werden die Knotenmengen der innerhalb von FOR- und WHERE-Klauseln auftretenden erweiterten XPath-Ausdrücke ermittelt. Die an Variablen gebundenen Knoten werden anschließend um diejenige Teilmenge vermindert, welche die in der WHERE-Klausel formulierte Bedingung nicht erfüllen.
Nach Errechnung des Typs des Zielschemas wird das Anfrageergebnis als XML-Dokument zurückgeliefert.
Abbildung 40 veranschaulicht dies.



Einige Beispiele:
Die Anfrage aus Beispiel 86 entspricht in Aufgabe und Ergebnis dem XPath-Ausdruck aus Beispiel 58. Geliefert werden -- unter Erhalt der Reihenfolge ihres Auftretens im Eingabedokument -- alle Vornamen zu jedem Knoten des Typs Person.

Beispiel 86: XQuery-Anfrage zur Ermittlung aller Vornamen

```
(1)FOR $x IN document("projektverwaltung5.xml")//
Person
(2)RETURN $x/Vorname
```

Download des Beispiels

Die Anfrage aus Beispiel 87 entspricht dem XPath-Ausdruck des Beispiels 59 bzw. der hierarchieebenenunabhängigen Formulierung aus Beispiel 60.
Demnach ergibt sich die vermeintlich kompaktere Darstellung der XQuery-Anfrage gegenüber dem Pendant aus Beispiel 59 allein aus der Nutzung der descendant-Achse aus XPath.

Beispiel 87: XQuery-Anfrage zur Hierarchiebenenunabhängigen Anfrage

```
(1)FOR $x IN document("projektverwaltung5.xml")//
Qualifikation
(2)RETURN $x
```

Download des Beispiels
Download der Ergebnisdatei

Die in Beispiel 88 formulierte Anfrage gewinnt hingegen durch die erfolgte Separierung von Bedingung und Lokalisierungspfad deutlich an Übersichtlichkeit gegenüber der XPath-Formulierung gleicher Funktion.
Zusätzlich entsteht durch die Konzentration aller Bedingungen in die WHERE-Klausel Optimierungspotential vor Durchführung der Anfrage, das in der durch die nagivierende Schreibweise der XPath-Syntax nicht gegeben war.

Beispiel 88: Bedingte Selektion in XQuery

```
(1)FOR $x IN document("projektverwaltung5.xml")//
Person
(2)WHERE $x//Qualifikationsprofil
(3)RETURN $x//Nachname
```

Download des Beispiels
Download der Ergebnisdatei

Noch deutlicher zeigt sich gewonnene Übersichtlichkeit bei der Reformulierung des Beispiels 62.
Darüberhinaus erweist sich die Möglichkeit, Kontexte durch LET-Klauseln explizit zu bilden und zu benennen, deutlich der impliziten Formulierung in XPath überlegen:

Beispiel 89: Selektion hinsichtlich mehrerer Bedingungen in XQuery

```
(1)FOR $x IN document("projektverwaltung5.xml")//Person
(2)LET $y := $x/Vorname/following::Nachname
(3)WHERE $x/parent::ProjektVerwaltung and starts-with
($y,'O')
(4)RETURN $x/Vorname
```

Download des Beispiels

Dasselbe Bild gibt auch Beispiel 90 wieder; die mehrschrittige, mit Prädikaten zweiter Ordnung formulierte, XPath-Formulierung läßt sich durch eine äquivalente XQuery-Anfrage ersetzen.
Auch in diesem Beispiel erweist sich die Möglichkeit der gleichzeitigen Operation auf mehreren benannten Knotenmengen als sehr hilfreich.

Beispiel 90: XQuery des umfangreichen XPath-Beispiels

```
(1)FOR $x IN document("projektverwaltung5.xml")//Person
(2)LET $y := $x//Nachname
(3)WHERE count($x/Vorname) = 1 and $y//Geburtsname/@value !=
''
(4)RETURN $x/@mitarbeitInProjekt
```

Download des Beispiels
Download der Ergebnisdatei

Eine der abzusehenden Besonderheiten im noch nicht verabschiedeten XQuery-Standard ist die Nutzung von Schemainformation, falls vorhanden, zur Steigerung der Mächtigkeit der Anfragesprache.
Hervorstechendstes Beispiel dürfte die Auswertung der ID-IDERF(S)-Beziehungen sein, die innerhalb XPath gleichgestellt den CDATA-typisierten Attributen behandelt wurden.
Beispiel 91 zeigt dies anhand der Beziehung zwischen Projekt und den dort eingesetzten Mitarbeitern. Inhaltlich läßt sich diese Beziehung an der Korrespondenz der Belegung des Attributs Mitarbeiter mit Werten, die zuvor bereits innerhalb eines Person-Elements im Attribut PersID angegeben wurden, festmachen.
XQuery definiert einen Operator =>, der Entsprechungen zwischen IDREF(S) und ID-typisierten Attributen auswertet.
Formal kann dieser Operator beschrieben werden als:
=> := (e_1 : xsd:IDREF(S), e_2 : xsd:ID) --> xsd:boolean
Er liefert true wenn für die linksstehende Referenz(-menge) ein entsprechend belegtes ID-typsiertes Attribut existiert, andernfalls false. Die verwendeten Typen entsprechen denen der XML-Schema Part 2 Recommendation.

Beispiel 91: XQuery mit Dereferenzierung von IDREF(S)-Attributen

```
(1)FOR $x IN document("projektverwaltung5.xml")//Person
(2)LET $y := document("projektverwaltung5.xml")//
Projekt
(3)WHERE $y/@Mitarbeiter => $x/@PersID
(4)RETURN $x/Vorname
```

Download des Beispiels

Erweiterung der XQuery-Basisfunktionalität:
Durch die Definition eigener Funktionen kann der Ausgangssprachumfang von XQuery durch den Anwender erweitert werden. Diese Funktionen stehen ab ihrer Definition für die Verwendung in Abfragen zur Verfügung.
Beispiel 92 zeigt die Definition der Funktion get_vorname die zu einem gegebenen textuellen Elementinhalt die im Dokument abgelegten Vornamen-Elemente liefert.

Beispiel 92: Erweiterung der XQuery-Mächtigkeit um eigene Funktionen

```
(1)DEFINE FUNCTION get_vorname (ELEMENT $nachname) RETURNS ELEMENT
(2){
(3)     FOR $x IN document("projektverwaltung5.xml")//Person
(4)     WHERE $x/Nachname = $nachname
(5)     RETURN $x/Vorname
(6)}
(7)
(8)<Namensliste>
(9){ get_vorname(document("projektverwaltung5.xml")//Nachname) }
(10)</Namensliste>
```

Download des Beispiels
Download der Ergebnisdatei

Innerhalb einer XQuery-Funktion können vollständige XQuery-Ausdrücke aber auch einfache prozedurale Berechnungsvorschriften angegeben sein. Vor diesem Hintergrund führt Beispiel 93 die Berechnung der Fakultät der Zahl 42 als XQuery-Funktion ein.

Beispiel 93: Berechnung der Fakultät einer Zahl als XQueryfunktion

```
(1)Define FUNCTION fac (INTEGER $n) RETURNS INTEGER
(2){
(3)IF ($n = 0)
(4)THEN 1
(5)ELSE $n * fac($n - 1)
(6)}
(7)<p>
(8){ fac(42) }
(9)</p>
```

Download des Beispiels
Download der Ergebnisdatei

Berechnung des Zielschemas:
Wie anhand des =>-Operators im vorhergehenden Beispiel veranschaulicht, definiert XQuery für alle Operationen eine formale Semantik -- sie entspricht der Algebra --, die zur Berechnung des Zielschemas verwendet wird. Das zugrundeliegende Typsystem ist an das aus XML-Schema bekannte angelehnt, wenngleich es ihm nicht vollständig entspricht. Die verwendete mathematische Formalisierung des XSD-Typsystems ist im Dokument XML Schema: Formal Description ausführlich beschrieben und soll hier nicht vertieft werden.
Als Beispiel sei die formale Semantik der *conditional expression* herausgegriffen. Sie ist in Abschnitt 3.3 des XQuery-Dokuments beschrieben als:

$$\frac{\Gamma \vdash e_1 : \texttt{xsd:boolean}\ \Gamma \vdash e_2 : t_1\ \Gamma \vdash e_3 : t_2}{\Gamma \vdash \ \text{if}\ (e_1)\ \text{then}\ e_2\ \text{else}\ e_3 : (t_1|t_2)}$$

Hieraus ergibt sich der Ergebnistyp eines solchen Ausdrucks, wie intuitiv erwartet, als Typ einer der beiden Alternativen (im Formalismus der Abbildung 41 als t_1 und t_2).

Insgesamt zeigt sich XQuery als mächtige und, insbesondere für Anwender mit Vorwissen aus dem Bereich der Anfragesprachen für relationale oder objektorientierte Datenbanken, leicht zu erlernende Sprache.
Hervorgehoben sei abschließend, daß XQuery keineswegs die Pfadausdrücke nach XPath ersetzen wird, sondern vielmehr XPath im praktischen Einsatz auf seinen eigentlichen Zweck -- die Formulierung von navigierenden Pfadausdrücken -- reduziert werden kann.
Die Tabelle 24 stellt die Charakteristika der XML-Anfragespache, klassischer Anfragesprachen und der XML-Pfadausdrücke zusammen. Die Übersicht wurde inspiriert durch P. Fankhauser.

Tabelle 24: Übersicht verschiedener Anfragesprachen hinsichtlich ihrer Einsatzbereiche

Sprache	Eingangsdaten	Ausgangsdaten	Schema	Anwendungsbereich
XPath und Muster aus XSLT	XML	Unicode Text oder XML	XML-DTDs oder -Schema (wird jedoch nicht ausgewertet)	Dokumenttransformationen
SQL, OQL	strukturierte typisierte Information	(Re-)Strukturierte typisierte Information	SQL-DDL, ODL	Auswertung von Datenbankinhalten (relational oder postrelational)
XQuery	XML	XML	XML-Schema	XML-Datenbanken, Anfrage an Dokumente

Web-Referenzen 15: Weiterführende Informationen
- Buch mit einer Sammlung von Aufsätzen zum Thema
- Tandberg-Workshop zu Anfragesprachen
- Rein, L.: *The Quest for an XML Query Standard*
- XQuery @ W3C
- *Fatdog* -- ein XQuery-Implementierung
- Rainbow -- eine weitere XQuery-Implementierung
- *KWEELT* -- eine Open-Source Implementierung von XQuery
- *Qizx/Open* -- Eine XQuery Implementierung
- *Derby* -- eine kommerzielle XQuery Implementierung
- QuiP -- eine XQuery-Implementierung für Datenbanken und XML-Dokumente

Einige Anfragesprachen:
- XML-QL
- XQL
- QUILT

Übung 5: Einige Übungen
Welches Ergebnis liefern folgende XQuery-Anfragen?
(a)
```
FOR $x IN //Person
WHERE $x//Qualifikationsprofil
RETURN $X//Nachname
```
(b)
```
FOR $x IN //Person
WHERE count($x/Vorname) > 1
RETURN $x/Vorname
```
(c)
```
FOR $X IN //Person
WHERE $x/@PersID = 'Pers01'
RETURN $x/Nachname
```

Wie muß eine XQuery-Anfrage lauten, die folgendes liefert?
(d) Selektion aller Personen mit Nachnamen „Obermüller“.
(e) Selektion aller Nachnamen von Personen, die über mehr als eine Qualifikation verfügen.
(f) Selektion der Nachnamen aller Projektleiter.

2.6 Der erste Schritt zum Semantic Web: Das Resource Description Framework

Der nächste Evolutionsschritt des Internets soll durch die Weiterentwicklung des heutigen sichtbaren Webs zu einem allgegenwärtigen und „unsichtbaren“ Kommunikationsmedium gekennzeichnet sein. Das zukünftige Web soll daher

neben dem Informationsaustausch von Mensch zu Mensch vor allem der Interkation zwischen Maschinen dienen. Notwendige Voraussetzung hierfür ist die Erweiterung der heute verfügbaren Web-Inhalte um semantische Information, die ausdrücken was Inhalte bedeuten (sollen).

Tim Berners-Lee, James Hendler und Ora Lassila formulieren in die Zielsetzung des Semantischen Webs lakonisch als eine Erweiterung des gegenwärtigen Webs, in der Information über eine wohldefinierte Bedeutung verfügt, um eine bessere Zusammenarbeit zwischen Mensch und Computer zu ermöglichen. Unterstellt man der Formulierung gegenwärtiges Web die Bedeutung eines zusammenfassenden Begriffs als Gesamtheit der aktuell eingesetzten Techniken zum Auffinden, Darstellen, Übertragen und Präsentieren beliebiger Daten, so wird offenbar, daß der Autor _ immerhin Direktor des World Wide Web Konsortiums und „Erfinder" des größten Teils der gegenwärtig eingesetzten Techniken wie URL, HTTP und HTML _, diesen eine Verfehlung des Ziels einer zumindest gut zu nennenden Zusammenarbeit von Mensch und Maschine zuschreibt.
Im wesentlichen speist sich die formulierte Kritik aus dem bisher mit automatisierten technischen Mitteln nicht vollziehbaren Übergang der durch das weltweite Netz angebotenen Daten zu Informationen im Shannon'schen Sinne. Diese Interpretation bleibt für alle aus dem Web gewonnenen Daten dem kognitiven Prozeß des Empfängers überlassen.

Dasselbe gilt umgekehrt für die von einem menschlichen Nutzer zu Daten reduzierten Informationen, die an das Netz -- etwa an Suchmaschinen -- übermittelt werden. Das vorherrschende Codierungsformat für Daten beider Transferrichtungen ist hierbei nahezu ausschließlich in einer natürlichen Sprache formulierter Text.

In der Konsequenz dieses Interaktionsparadigmas ergeben sich Nutzungsmuster, die jedem WWW-Anwender vertraut sind:

- Datenproduktion: Zur Publikation beliebiger Inhalte im weltweiten Netz werden Textsprachen verwendet, deren überwiegendes Einsatzfeld die Festlegung graphischer Eigenschaften der anzuzeigenden Daten ist.
- Um diese Daten für Dritte auffindbar abzulegen ist Kenntnis der Funktion sog. „Suchmaschinen" notwendig.
- Datensuche: Recherchevorgänge im Web lassen sich in zwei Kategorien gliedern:
  Zunächst geführte Suchen durch manuell erzeugte und befüllte Verzeichnisse, die Hypertextseiten gemäß vorgegebener Kategorien rubrizieren, zum anderen automatisiert mit aus den indizierten Seiten extrahierten textuellen Daten gefüllten Datenbanken.
  Während der erste Ansatz das Ergebnis eines manuell durchgeführten Interpretationsvorganges festschreibt läßt der zweite keinerlei Interaktion beim Aufbau des Datenbestandes zu.
  Seitens des Anfragers offenbaren sich in der Praxis jedoch beide Ansätze zumeist als inadäquat. Unterdrückt der erste potentielle Treffer, wenn keine Einordnung in die angefragte Rubrik vorliegt, so liefern Umsetzungen des letzteren Ansatzes häufig eine Fülle von Ergebnisseiten, die zwar den gesuchten Begriff lexikalisch enthalten, hierbei jedoch den Auftretenskontext außer Acht lassen und die Ergebnismenge daher auch auf unpassende oder gar unerwünschte Inhalte unnötig aufblähen.
  Um Suchmaschinen effizient zur Identifikation relevanter Ergebnisseiten heranziehen zu können ist die manuelle Auswertung der Suchergebnisse unabdingbar. Zusätzlich offenbart sich die anwenderformulierte Anfrage der Suchmaschinen im zweiten Realisierungsmuster bzw. die Auswertung der natürlichsprachlich benannten Kategorien im ersten als unzureichend, da auch sie eines manuellen Nutzereingriffs bedarf. Dieser geschieht entweder durch direkte Formulierung der Anfrage als Zeichenkette oder durch Auswertung der angebotenen Kategorien, welche wieder durch verzeichnisdienstspezifische Benennungen identifiziert werden.
  Um mit den aktuell vorherrschenden Mechanismen effiziente Suchen hinsichtlich des zu erbringenden Zeitaufwandes in Relation der als relevant erachteten Resultatseiten zu den tatsächlichen Suchergebnissen zu erzielen, ist daher die Anreicherung der durch Suchmaschinen verwalteten Basisinformation um zusätzliche deskriptive Anteile notwendig.
- Datenkonsumption: Die menschliche oder maschinelle Verarbeitung von Web-verfügbaren Inhalten kann ausschließlich auf den verfügbaren textuellen Inhalten erfolgen.
  Jegliche Bedeutung der empfangenen Daten muß durch eine manuelle Interpretation (re-)konstruiert werden, sofern dies überhaupt möglich ist.
  Um aus dem Web bezogene Daten maschinell weiterverarbeiten zu können ist daher im Rahmen der Datenkonsumption die manuelle Anreicherung mit Kontext- und Anwendungssemantik notwendig.

Die skizzierten mehrfach notwendigen Interpretationsvorgänge kennzeichnen den aktuellen Umgang mit Web-verfügbaren Inhalten und motivieren gleichermaßen die Vision des Semantischen Webs.
Im Spiegel der diskutierten Nutzungsmuster ergibt sich die Zielsetzung der Semantic Web Initiative des World Wide Web Konsortium im Stile der durch Berners-Lee erhobenen Forderung: Techniken zu entwickeln und zu etablieren, welche die Interaktion zwischen menschlichem Nutzer und Computer vereinfachen.

Die apostrophierte Anreicherung der im Web publizierten Daten kann jedoch kaum durch den Konsumenten geschehen, da er im allgemeinen Falle nicht über die notwendigen (Zusatz-)Informationen verfügt um diesen Vorgang durchführen zu können. Daher muß diese Anreicherung bereits vor der Publikation im Web, idealerweise schon zum Erstellungszeitpunkt durch den Verfasser selbst, erfolgen. Dieser ergänzt die erzeugten Daten um zusätzliche beschreibende Anteile, sog. *Metadaten*.
Metadaten -- im einfachsten Wortsinne also Daten über Daten -- zu XML codierten Inhalten lassen sich prinzipiell in frei wählbaren Formaten verfassen und ablegen. Vielmehr noch, im weitesten Sinne erfüllt jegliche Art eines deskriptiven Datums zu einem gegebenen Datum die eingeführte Sinngebung.

Wesentlich für die Vision des Semantic Web ist jedoch nicht die Metadatenexistenz an sich, sondern der Wunsch nach Maschinenverarbeitbarkeit zur Unterstützung der Benutzerinteraktion mit den durch die Metadaten beschriebenen Daten.

Aus diesem Wunsche heraus wird die enge Verbindung des Technikgebietes XML mit der Idee des Semantic Web offenbar. Eignet sich doch die selbst als Metasprache angelegte XML in besonderer Weise zur Formalisierung der für das Sematic Web existenznotwendigen Metainformation; wenngleich diese Formalisierung selbst nur auf der deskriptiven Metaebene der syntaktischen Beschreibung anzusiedlen ist, wie einschränkend angemerkt sei. Vielmehr noch, verschiebt dieser -- an sich naheliegende und valide -- Formalisierungsansatz die notwendige Semantische Vereinheitlichung zu Gunsten der durch die XML erfolgten Syntaktischen auf die Meta-Metaebene. Durch die

Selbstbeschreibungsfähigkeit der XML kollidiert jedoch auf dieser Ebene der Semantikbegriff XML Schemas mit der dort zu etablierenden Semantikbeschreibung der durch das Schema definierten Sprache. Gänzlich unlösbar wird das Ansiedelungsproblem durch die Organisation des XML-Schemastandards als wiederum selbstbeschreibende Sprache, die den XML-Schemastandard selbst zu seiner syntaktischen und semantischen Beschreibung heranzieht ...

Zu dieser syntaktisch induzierten Semantikansiedelungsproblematik tritt eine semantisch induzierte Syntaxansiedelungsproblematik bei der strukturellen Definition der deskriptiven Metainformation selbst. Konkret stellt sich die Frage der zu wählenden Abstraktionsebene bei der Sprachformulierung einer allgemein einsetzbaren Metasprache zur Codierung beschreibender Daten zu XML codierten Daten.

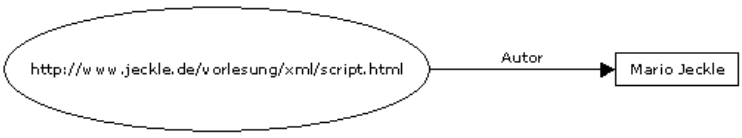
Wird der Abstraktionsgrad zu hoch gewählt, so geht dies zwar mit einer prinzipiellen Verbreiterung des Einsatzgebietes, d.h. einer größeren Menge beschreibbarer Entitäten einher, schränkt jedoch in gleicher Weise die Spezifität der so formulierbaren Aussagen ein.

Wird der Abstraktionsgrad hingegen möglichst niedrig gewählt, so gewinnt man zwar einerseits ausdetaillierte Entitätsbeschreibungen, diese jedoch nur für sehr wenige Entitäten.

Vor dem Hintergrund dieser Problematik finden im Kontext des Semantic Web zwei Ansätze Verwendung, die in ihrer Kombination den Zielsetzungen beider konkurrierender Ansprüche zu genügen suchen.
So definiert das Resource Description Framework (RDF) ein Vokabular zur Darstellung beliebiger Aussage über Ressourcen. Zur Beschreibung von Syntax kann für RDF XML Schema Anwendung finden.
Zur Sematikdefinition greift RDF jedoch nicht auf XML-Schema zurück, sondern die eigens zum Zwecke der RDF-Beschreibung entwickelte Metasprache RDF Schema welche wiederum selbstbeschreibend organisiert ist. Dieser Schritt fundiert die RDF zugedachte Rolle eines formalisierten Metadatendefinitionsvokabulars gegenüber XML Schema, dessen Einsatzzweck in der syntaktischen Beschreibung beliebiger Vokabulare und der Semantikdefinition der XML-Schemasprache erschöpft ist.

RDF selbst bietet eine einfache XML-Syntax zur Formulierung von Aussagen über Ressourcen an. Der Begriff der *Ressource* ist dabei eng an den der eineindeutigen Identifizierbarkeit gekoppelt. Diese wird hierzu mit der Technik der Universal Resource Identification synonymisiert. Inhaltlich orientieren sich die mittels RDF formulierbaren Aussagen an der Satzstruktur der englischen Sprache und gestatten Zuordnungen von prädikativ angebundenen Werten (Objekt) zu Ressourcen (Subjekt).

Abbildung 42 zeigt die typische graphische Visualisierung einer RDF-Information. Die Aussage „Mario Jeckle ist der Autor der Ressource http://www.jeckle.de/vorlesung/xml/script.html“ wird durch zwei mittels einer annotierten gerichteten Kante verbundene Knoten dargestellt.
Im Beispiel wird durch RDF eine Aussage über die die als Ellipse dargestellte Ressource `http://www.jeckle.de/vorlesung/xml/script.html` formuliert. Über diese Ressource wird eine durch das Prädikat, es ist als Beschriftung (*Autor*) auf der Pfeil dargestellt, formulierte Aussage getroffen. Das Prädikat ist daher das bestimmende Element der Beziehung zwischen Ressource und Objekt, welches als benanntes Rechteck (*Mario Jeckle*) dargestellt wird.
Aus dem Tripel Ressource--Property--Literal (Subjekt--Prädikat--Objekt) erschließt sich der Sinn der Gesamtaussage als „Die durch `http://www.jeckle.de/vorlesung/xml/script.html` eineindeutig identifizierte Ressource besitzt einen Autor der Mario Jeckle genannt wird“.



Komplementär zur graphischen Notation definiert die RDF-Spezifikation ein XML-Vokabular zur textuellen Repräsentation der beschreibenden Metadaten einer Ressource.
Es definiert im Namensraum `http://www.w3.org/1999/02/22-rdf-syntax-ns#` u.a. die Basielemente `RDF`, `Description`, `type` sowie deren Attribute.

Unter Verwendung des XML-Vokabulars für RDF kann die in Abbildung 42 graphisch formulierte Aussage auch codiert werden als:

Beispiel 94: XML/RDF-Darstellung

```
(1)<?xml version="1.0"?>
(2)<rdf:RDF
(3)     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
(4)     xmlns:myNS="http://www.jeckle.de">
(5)     <rdf:Description rdf:about="http://www.jeckle.de/vorlesung/xml/script.html">
(6)             <myNS:Autor>Mario Jeckle</myNS:Autor>
(7)     </rdf:Description>
(8)</rdf:RDF>
```

Download des Beispiels

Hierbei folgt die Umsetzung des Graphen in die XML-Syntax folgendem Schema:

```
<RDF
    xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:myNS="eigener Namensraum">
    <Description rdf:about="Ressource/Subjekt">
```

```
            <myNS:Eigenschaft/Prädikat>Objekt/Literal</myNS:Eigenschaft/Prädikat>

        </Description>

    </RDF>
```

Im Beispiel wird daher das durch den RDF-Anwender definierte Element `Autor` verwendet, welches das Prädikat ausdrückt. Dieses Element ist nicht durch das standardisierte RDF-Vokabular vorgegeben, sondern kann durch den Ersteller des RDF-Tripels frei in seiner Syntax und Semantik festgelegt werden. Aus diesem Grunde ist das Element `Autor` auch nicht dem RDF-Standardnamensraum zugeordnet, sondern einem anwenderdefinierten.

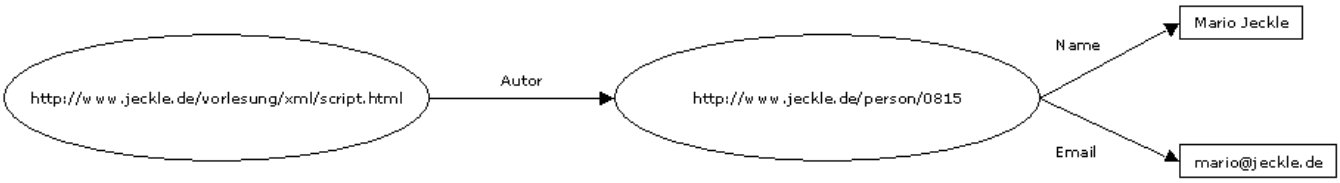
Oftmals besteht der Wunsch mehr als eine Aussage über eine Ressource zu treffen. Dieser Fall kann sowohl in Ausprägung aufteten, daß verschiedene Prädikate diese verschiedenen Aussagen identifizieren, als auch, daß das Prädikat mehrteilig ist, d.h. mehr als ein Einzelobjekt durch das Prädikat angebunden wird.
Der erste Fall läßt sich durch Definition mehrerer separater aus den Prädikaten resultierender Elemente im Rumpf des `Description`-Elements realisieren, was Beispiel 95 an der Aussage „Mario Jeckle ist der Autor der Ressource http://www.jeckle.de/vorlesung/xml/script.html und besitzt die Mailadresse mario@jeckle.de“ zeigt.

Beispiel 95: XML/RDF-Darstellung mit mehreren Prädikaten

```
(1)<?xml version="1.0"?>
(2)<rdf:RDF
(3)     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
(4)     xmlns:myNS1="http://www.jeckle.de"
(5)     xmlns:myNS2="http://www.example.com">
(6)     <rdf:Description rdf:about="http://www.jeckle.de/vorlesung/xml/script.html">
(7)             <myNS1:Autor>Mario Jeckle</myNS1:Autor>
(8)             <myNS2:Mail>mario@jeckle.de</myNS2:Mail>
(9)     </rdf:Description>
(10)</rdf:RDF>
```

Download des Beispiels

Für den Fall, daß dasselbe Prädikat mehrere Objekte referenziert verlangt RDF die Einführung eines „Stellvertreterobjektes“ welches die Verweise auf die angebundenen Objekte kapselt.
Abbildung 43 stellt diesen Sachverhalt graphisch dar und Beispiel 96 zeigt die korrekte XML-Serialisierung.



Beispiel 96: XML/RDF-Darstellung eines mehrteiligen Prädikats

```
(1)<rdf:RDF
(2)     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
(3)     xmlns:s="http://www.example.com"
(4)     xmlns:t="http://www.examples.org">
(5)  <rdf:Description rdf:about="http://www.jeckle.de">
(6)    <s:Autor rdf:resource="http://www.jeckle.de/person/0815"/>
(7)  </rdf:Description>
(8)
(9)  <rdf:Description rdf:about="http://www.jeckle.de/person/0815">
(10)    <t:Name>Mario Jeckle</t:Name>
(11)    <t:Email>mario@jeckle.de</t:Email>
(12)  </rdf:Description>
(13)</rdf:RDF>
```

Download des Beispiels

Anwendung des Resource Description Frameworks

RDF-basierte Beschreibungen finden heute vielfältige Anwendung zur standardisierten Ablage von strukturierten Metadaten. Insbesondere im Hypertextumfeld stellt RDF unter Nutzung des Metadatenvokabulars *Dublin Core* einen interessanten und vielversprechenden Ansatz dar.
Hierbei werden die RDF-codierten XML-Daten allerdings nicht direkt in das XHTML-Dokument eingefügt, da verfügbare Browser diese bei der Erzeugung der Präsentationsansicht nicht ignorieren und daher die RDF-Inhalte im Klartext auslesen, sondern durch das XHTML `link`-Element extern referenziert.

Auf der Basis des RDF-Standards ist es somit möglich, in ihrem Detaillierungsgrad frei variierende deskriptive Daten zu einer beliebigen Web Ressource so abzulegen, daß sie auf der Basis von XML maschinell weiterverarbeitet werden können. Syntax und Struktursemantik werden hierbei durch die W3C-Empfehlungsdokumente spezifiziert. Ungeklärt ist bisher lediglich noch die semantische Charakterisierung der deskriptiven Dateninhalte selbst. Durch die Etablierung des Resource Description Frameworks wird zwar ein wesentlicher Schritt hin zu einer semantikbasierten Interoperabilität unternommen, jedoch kann wirkliche Semantik auch durch RDF nicht ausgedrückt werden. Im Grunde handelt es sich bei RDF um einen sehr flexibel einsetzbaren und leistungsfähigen Mechanismus zur freien Ressourcennotation, welcher letztlich die deskriptiven Daten auf eine textuelle Repräsentation in einer natürlichen

Sprache zurückführt.

Web-Referenzen 16: Weiterführende Links

- Resource Description Framework
- RDF Vocabulary Description Language: RDF Schema
- Semantic Web Activity des W3C
- Berners-Lee, Hendler, and Lassila: *The Semantic Web*
- Berners-Lee, Miller: *The Semantic Web lifts off*

---

3 Anwendungen der XML im praktischen Einsatz ...

Im nachfolgenden Kapitel sind die Grundlagen der Anwendung des Technikgebietes XML in Applikationen und Architekturen betrachtet.
Hierzu werden zunächst die gängigen Schnittstellen zum Zugriff auf XML-strukturierte Daten aus Hochsprachenapplikationen betrachtet. Neben der *Simple API for XML*, einem einfach zu implementierenden Mechanismus, wird mit dem *Document Object Model* des World Wide Web Konsortiums eine direkte Abbildung von XML-Dokumenten in Hauptspeicherstrukturen eingeführt. Dieses Modell hat in der Vergangenheit als Datenstruktur in Web-Browsern und weiteren XML-verarbeitenden Softwarelösungen einige Bedeutung erlangt.
Mit SUNs Vorschlag einer Anbindung von XML-Schema-beschriebenen Datenstrukturen und Applikationsobjekten der Programmiersprache Java wird ein neuer Ansatz zur engen Kopplung zwischen Hochsprache und XML-Serialisierungsformat der Laufzeitobjekte vorgestellt.
Den Abschluß des Kapitels bildet die Diskussion einiger Grundlagen zur persistenten Speicherung von XML-Dokumenten in Datenbanksystemen, sowie die Vorstellung der *SOAP*-Spezifikation, welche den Transport entfernter Methodenaufrufe mittels XML einführt.

3.1 Die *Simple API for XML*

Die *Simple API for XML* (abgekürzt zu: SAX) entstand aus einer durch David Megginson initiierten Aktivität der XML-Mailingliste `xml-dev`. Sie stellt einen einfachen leichtgewichtigen Mechanismus für die Ereignis-basierte Verarbeitung von XML-Dokumenten dar. Die Charakterisierung als *leichtgewichtiger* Ansatz bezieht sich sowohl auf den Implementierungsaufwand der API selbst, als auch ihren Integrationsaufwand in eigene Applikationen.
Ursprünglich entstand die SAX-API aus einer Sammlung generischer Java-Schnittstellen für XML-Parser. Inzwischen hat sie sich jedoch als eigenständige Möglichkeit zur Verarbeitung von XML-Dokumenten in verschiedenen Hochsprachen entwickelt. So existieren neben den Umsetzungen für Java auch Implementierungen für C++, Python, Perl und Eiffel. Im folgenden wird aus Gründen der weitesten Verbreitung die Java-Umsetzung der SAX2-Schnittstelle behandelt.
SAX2 treibt den Vorgängeransatz hinsichtlich der jüngeren Entwicklungen im XML-Umfeld (wie Namensräume) voran, behält jedoch die Grundidee bei.
Die nachfolgenden Beispiele beziehen sich auf die SAX2-Implementierung, die durch SUN für Java unter dem Namen *Java APIs for XML Processing (JAXP)* veröffentlicht wurde und seit Version 1.4 Bestandteil des JDK ist.

Struktur der SAX-API

SAX-Implementierungen weisen üblicherweise drei erkennbare Blöcke auf:

- **ParserFactory**
  Sie dient zur Erzeugung verschiedener Ausprägungen (z.B. validierend/nicht-validierend, Namespace berücksichtigend, etc.) SAX-basierter Parser.
- **Parser**
  Der Parser selbst. SAX definiert lediglich eine Sammlung abstrakter Schnittstellen, jedoch keine Implementierung. Diese Schnittstellen werden durch den jeweiligen eingesetzten Parser bedient und die definierten Call-back-Methoden entsprechend eingebunden.
- Die Schnittstellen:
  - **`ContentHandler`**
    Versammelt Operationen zur Reaktion auf Dokumentereignisse (wie `startDocument`, `startElement`, `processingInstruction` ...)
    Hinweis: In älterer Literatur findet sich häufig statt des `ContentHandlers` eine mit `DocumentHandler` benannte Schnittstelle. Sie stellt die Vorgängerversion (SAX1), die u.a. keine Namensraumintegration bietet, des `ContentHandlers` dar, und wurde durch diesen ersetzt.
    Achtung: Die Java-Standardklassenbibliothek bietet im Paket `java.net` eine mit `ContentHandler` benannte Klasse an. Diese kann durch Importanweisungen mit der gleichnamigen SAX-Schnittstelle aus dem Paket `org.xml.sax` kollidieren!
  - **`ErrorHandler`**
    Versammelt Operationen zur Reaktion auf die drei in der XML-Spezifikation definierten Fehlerklassen:
    - `error` (behebbarer Fehler)
      z.B.: Anderer Wert als `1.0` im XML-Prolog oder ein nichtdeterministisches Inhaltsmodell.
    - `fatalError` (Fehler, der üblicherweise zum Abbruch der Verarbeitung führt)
      z.B.: Unbekanntes Codierungsschema
    - `warning` (Warnung)
      z.B.: Referenzierung eines nicht deklarierten Elements als Kindelement.
  - **`DTDHandler`**
    Möglichkeit zur Implementierung eigener Behandlungsroutinen für `Notation`-Deklarationen und ungeparste Entitäten.

- **EntityResolver**
  Möglichkeit zur Implementierung von Auflösungsmechanismen für Entitäten gemäß ihres `system identifiers` und, falls vorhanden, des `public identifiers`.

Das SAX-Ausführungsmodell

Jeder SAX-basierte Parser arbeitet nach demselben Ausführungsmodell. Es definiert eine Reihe von Ereignissen, die durch Operationen (sog. *Call-Backs*) behandelt werden. Der Aufruf der Operationen zum Zeitpunkt des Ereigniseintritts erfolgt immer durch den Parser. Ein expliziter Aufruf durch den Programmierer sollte in jedem Falle unterbleiben!
Der resultierende Programmcode weist daher keinen erkennbaren durchgängigen Kontrollfluß auf, vielmehr ergibt sich dieser durch die serielle Aktivierung der verschiedenen Ereignisbehandlungsroutinen aus dem Eingabedokument. SAX impliziert jedoch keinerlei Speicherstruktur zur Laufzeit. Als Konsequenz ist sein Ansatz -- mit nur geringen Modifikationen -- auf nahezu beliebige Programmiersprachen übertragbar. Darüberhinaus stellt SAX nur minimale Anforderungen an den verfügbaren Hauptspeicherausbau; der sich nach Art und Umfang der Übergabeparameter einer call-back-Operation richtet. Diese Art der ereignisgetriebenen Verarbeitung eines XML-Dokuments eignet sich daher in besonderer Weise für Geräte mit geringem Hauptspeicherausbau, bzw. generell zur Verarbeitung von XML-Dokumenten die den verfügbaren oder adressierbaren Hauptspeicher übersteigen.
Da sich eine SAX-Applikation immer passiv verhält und auf Fremdaktivierung -- durch Auswerfen der entsprechenden Ereignisse -- wartet, wird dieses Ausführungsmodell auch als *Push Model* charakterisiert.
Aus der Anlage der SAX-Verarbeitung folgt direkt, daß es keinerlei Modifikationen am Eingangsdokument erlaubt. Lediglich durch veränderte Ausgabe des Eingabedokuments lassen sich einfache Transformationen realisieren.

Der Code des Beispiels 97 zeigt eine Implementierung der vordefinierten call-back Funktionen `startDocument` und `endDocument`, die durch den Parser zu Lese-Beginn der XML-Eingabe und nach Abschluß des Lesevorganges aufgerufen werden.
Die Signatur der beiden Operationen wird durch die Klasse `DefaultHandler` (definiert im Paket `org.xml.sax.helpers`) vorgegeben. Deshalb erbt die Beispielklasse (`SAXExample1`) von dieser Klasse.
Bei SAX handelt es sich um eine passive Schnittstelle, die des Aufrufs durch den lesenden Parser bedarf. Hierzu wird zunächst durch den Aufruf `SAXParserFactory.newInstance()` ein Objekt des Typs `SAXParserFactory` erzeugt. Instanzen dieser Klasse stellen verschiedene konfigurierbare SAX-Parser Implementierungen zur Verfügung. Unter Nutzung der aktuellen Konfiguration wird mittels `newSAXParser` ein neuer Parser erzeugt. Im Beispiel wird die Vorgabekonfiguration verwendet.
Die Methode `parse` des Parser-Objekts führt auf dem über Kommandozeilenparameter (`args[0]`) übergebenen Dokument den Lesevorgang durch. Der zweite Aufrufparameter der Methode bezeichnet dasjenige Objekt, welches Implementierungen der in `DefaultHandler` definierten Operationen anbietet. Im Beispiel ist dies die Klasse `SAXExample1` selbst.

Nach der Übersetzung durch `javac SAXExample1.java` kann die Anwendung mit beliebigen XML-Dokumenten zur Ausführung gebracht werden.
Für das Beispiel der Projektverwaltung liefert der Aufruf `java SAXExample1 projektverwaltung5.xml` folgende Ausgabe:

```
document started
document ended
```

Beispiel 97: Ein einfache SAX-basierte Applikation

```
(1)import org.xml.sax.helpers.DefaultHandler;
(2)import javax.xml.parsers.SAXParser;
(3)import javax.xml.parsers.SAXParserFactory;
(4)
(5)public class SAXExample1 extends DefaultHandler {
(6)    public void startDocument() {
(7)        System.out.println("document started");
(8)    } //startDocument()
(9)
(10)    public void endDocument() {
(11)        System.out.println("document ended");
(12)    } //endDocument()
(13)
(14)
(15)    public static void main(String args[]) throws Exception {
(16)        SAXParserFactory spf = SAXParserFactory.newInstance();
(17)
(18)        SAXParser sp = spf.newSAXParser();
(19)        sp.parse(args[0], new SAXExample1());
(20)    } //main()
(21)}//class SAXExample1
```

Download des Beispiels

Die SAX2-Schnittstellen

Die grundlegende Schnittstelle der SAX2-API wird mit `ContentHandler` bezeichnet, sie versammelt Operationen zur Abbildung des logischen Inhaltes eines XML-Dokumentes. Die im Beispiel 97 verwendete Basisklasse `DefaultHandler` implementiert u.a. diese Schnittstelle und stellt sie so der Beispielapplikation zur Verfügung.
Die Übersicht der Tabelle 25 stellt die einzelnen Operationen der Schnittstelle mit ihrer Signatur und Funktionalität zusammen.

Tabelle 25: Die Schnittstelle ContentHandler

Operation	Funktionalität
startDocument()	Aufruf zu Beginn eines Dokuments. Naturgemäß wird diese Methode nur einmal ausgeführt.
processingInstruction(String target, String data)	Liefert die deklarierten Verarbeitungsanweisungen. Diese können vor und nach Eintritt des startDocument-Ereignisses auftreten. Spezifikationgemäß wird der XML-Prolog nicht als Processing Instruction behandelt und daher auch kein Ereignis dieses Typs ausgelöst.
startElement(String namespaceURI, String localName, String qName, Attributes atts)	Aufruf zu Beginn (öffnender Tag) eines Elements. Neben dem Elementnamen (ohne Namensraumpräfix: localName, oder als qualifizierter Name in qName) *Anmerkung*: Namensrauminformationen werden nur erzeugt, wenn der Parser entsprechend konfiguriert wurde. Objekte des Type Attributes enthalten die Attribute eines Elements in der Reihenfolge ihrer Definition. Zu jedem Attribut sind neben dem lokalen und dem qualifizierten Namen die Namensraum URI sowie Inhalt und Typ gemäß XML v1.0 abrufbar.
characters(char[] ch, int start, int length)	Aufruf bei der Verarbeitung von Zeichenkettendaten innerhalb eines Elements. ch enthält ab der Position start Zeichenketten-artige Daten der Länge length.
ignorableWhitespace(char[] ch, int start, int length)	Aufruf beim Auftreten ignorierbarer Leerzeichen im Sinne der XML-Spezifikation.
endElement(String namespaceURI, String localName, String qName)	Aufruf bei Erreichen eines Elementendes. Die Übergabeparameter entsprechen denen des zugehörigen startElement-Aufrufs. *Anmerkung*: Auch für leere Elemente wird das Ergeignispaar aus startElement und endElement erzeugt.
startPrefixMapping(String prefix, String uri)	Signalisiert den Beginn des Gültigkeitsbereits des Namensraumkürzels prefix. uri enthält die vollständige URI des gebundenen Namensraums. Ist prefix leer, so handelt es sich um die Redefinition des Vorgabenamensraums. Das Ereignis wird vor dem ersten Element des Namensraums (d.h. dem Element, das die Präfixbindung enthält oder den Vorgabenamensraum überschreibt) ausgelöst.
endPrefixMapping(String prefix)	Signalisiert das Ende des Gültigkeitsbereichs eines Namensraums. Ist der Übergabeparameter leer, so bezieht sich das Ereignis auf den Vorgabenamensraum. Das Ereignis tritt nach dem endElement-Ereignis des letzten Elements im Namensraum ein.
endDocument()	Wird als letztes Ereignis des Parsingvorganges ausgelöst.

Der Code aus Beispiel 98 zeigt die Nutzung der verschiedenen Ereignisse zur Erzeugung einer kleinen Dokumentstatistik.
Gegenüber dem vorhergehenden Beispiel wird hier zusätzlich die Schnittstelle Attributes aus dem Paket org.xml.sax importiert.
Nach Abschluß des vollständigen Lesevorganges, d.h. Eintritt des Ereignisses endDocument, werden die aufsummierten Zahlen durch die Methode printStatistics auf der Standardausgabe dargestellt.

Beispiel 98: SAX2-Ereignisse

```
(1)import org.xml.sax.Attributes;
(2)import org.xml.sax.helpers.DefaultHandler;
(3)import javax.xml.parsers.SAXParser;
(4)import javax.xml.parsers.SAXParserFactory;
(5)
(6)public class SAXExample2 extends DefaultHandler {
(7)    private int startDocument = 0,
(8)    endDocument = 0,
(9)    characters = 0,
(10)    startElement = 0,
(11)    endElement = 0,
(12)    attributes = 0;
(13)
(14)    public void printStatistics() {
(15)        System.out.println("# startDocument events:" + startDocument + "\n" +
(16)                "# endDocument events:" + endDocument + "\n" +
(17)                "# startElement events:" + startElement + "\n" +
(18)                "# endElement events:" + endElement + "\n" +
(19)                "# character events:" + characters + "\n" +
(20)                "# attributes:" + attributes + "\n"
(21)        );
(22)    } //printStatistics()
(23)
(24)    public void startDocument() {
```

```
(25)        System.out.println("Analyzing ...");
(26)        startDocument++;
(27)    } //startDocument()
(28)
(29)    public void endDocument() {
(30)        endDocument++;
(31)        printStatistics();
(32)    } //endDocument()
(33)
(34)    public void characters(char[] ch, int start, int length) {
(35)        characters++;
(36)    } //characters()
(37)
(38)    public void startElement(String namespaceURI, String localName, String qName, Attributes
atts) {
(39)        startElement++;
(40)        attributes += atts.getLength();
(41)    } //startElement()
(42)
(43)    public void endElement(String namespaceURI, String localName, String qName) {
(44)        endElement++;
(45)    } //endElement()
(46)
(47)    public static void main(String args[]) throws Exception {
(48)        SAXParserFactory spf = SAXParserFactory.newInstance();
(49)
(50)        SAXParser sp = spf.newSAXParser();
(51)        sp.parse(args[0], new SAXExample2());
(52)    } //main()
(53)}//class SAXExample2
```

Download des Beispiels

Das Programm aus Beispiel 99 implementiert eine Häufigkeitsermittlung für Elementnamen.
Hierzu wird mittels der Java-Collection-API-Klasse HashMap eine Liste mit Tupeln aus Elementnamen und Auftretensanzahlen verwaltet. Eintragungen und Aktualisierungen dieser Liste erfolgen bei jedem `startElement`-Ereignis.

Beispiel 99: Häufigkeitsermittlung einzelner Elementnamen

```
(1)import org.xml.sax.Attributes;
(2)import org.xml.sax.helpers.DefaultHandler;
(3)import javax.xml.parsers.SAXParser;
(4)import javax.xml.parsers.SAXParserFactory;
(5)import java.util.HashMap;
(6)
(7)public class SAXExample41 extends DefaultHandler {
(8)    private HashMap elementHM;
(9)    private final Integer ONE = new Integer(1);
(10)
(11)    public void startElement(String namespaceURI, String localName, String qName, Attributes
atts) {
(12)        Integer freq = (Integer) elementHM.get(qName);
(13)
(14)        elementHM.put(qName, (freq == null ? ONE : new Integer(freq.intValue() + 1)));
(15)    } //startElement()
(16)
(17)    public void endDocument() {
(18)        System.out.println(elementHM);
(19)    } //endDocument()
(20)
(21)    public static void main(String args[]) throws Exception {
(22)        SAXParserFactory spf = SAXParserFactory.newInstance();
(23)        SAXParser sp = spf.newSAXParser();
(24)        sp.parse(args[0], new SAXExample41());
(25)    } //main()
(26)
(27)    public SAXExample41() {
(28)        elementHM = new HashMap();
(29)    } //constructor
(30)}//class SAXExample41
```

Download des Beispiels

Angewendet auf die Projektverwaltung liefert das Programm folgende Ausgabe:
`{Qualifikation=3, em=3, Qualifikationsprofil=1, Leistungsstufe=1, ProjektVerwaltung=1, Person=3, Projekt=2, b=1, u=3, Nachname=3, Vorname=4}`

Angewendet auf das Dokument aus Beispiel 20 liefert der Code jedoch die Ausgabe: `{document=1, svg=1, ci=3, math=1, g=1, text=1, set=2}`!

Offensichtlich ignoriert der verwendete SAX-Parser in der Standardkonfiguration die deklarierten Namensräume.

Zur Änderung dieses Verhaltens, in eine XML v1.0 2nd editon konforme Variante, kann die SAXParserFactory entsprechend instruiert werden.
Für die SUN-Implementierung geschieht dies durch die Methode setNamespaceAware.
Beispiel 100 modifiziert den bisherigen Code entsprechend. Zusätzlich wird die Routine zur Zählung der Elementauftritte entsprechend angepaßt.

Beispiel 100: Namensraum-konformer SAX-Parser

```
(1)import org.xml.sax.Attributes;
(2)import org.xml.sax.helpers.DefaultHandler;
(3)import javax.xml.parsers.SAXParser;
(4)import javax.xml.parsers.SAXParserFactory;
(5)import java.util.HashMap;
(6)
(7)public class SAXExample4 extends DefaultHandler {
(8)    private static HashMap elementHM;
(9)    private static final Integer ONE = new Integer(1);
(10)
(11)    public void startElement(String namespaceURI, String localName, String qName, Attributes
atts) {
(12)        String elementFullName = new String(namespaceURI + ":" + localName);
(13)
(14)        Integer freq = (Integer) elementHM.get(elementFullName);
(15)
(16)        elementHM.put(elementFullName, (freq == null ? ONE : new Integer(freq.intValue() + 1)));
(17)    } //startElement()
(18)
(19)    public void endDocument() {
(20)        System.out.println(elementHM);
(21)    } //endDocument()
(22)
(23)
(24)    public static void main(String args[]) throws Exception {
(25)        elementHM = new HashMap();
(26)
(27)        SAXParserFactory spf = SAXParserFactory.newInstance();
(28)
(29)        spf.setNamespaceAware(true);
(30)
(31)        SAXParser sp = spf.newSAXParser();
(32)
(33)        sp.parse(args[0], new SAXExample4());
(34)    } //main(args[])
(35)}//class SAXExample4
```

Download des Beispiels

Angewendet auf das Beispiel-Dokument liefert es nun das erwartete korrekte Ergebnis:
{**http://www.w3.org/1998/Math/MathML:set=1**,
http://www.w3.org/1998/Math/MathML:math=1,
http://www.w3.org/2000/svg:svg=1,
http://www.w3.org/1998/Math/MathML:ci=3,
http://www.w3.org/2000/svg:text=1,
:document=1,
http://www.w3.org/2000/svg:g=1,
http://www.w3.org/2000/svg:set=1}
Konsequenterweise wird dem Wurzelelement document, für das kein Namensraum definiert ist (es befindet sich daher spezifikationsgemäß im NULL-Namensraum), die leere Namensraum-URI vorangestellt.
Hinweis: Bei der Ausgabenotation handelt es sich um keine syntaktisch korrekte Elementdeklaration, sie dient lediglich der Veranschaulichung!

Über die gezeigte parserspezifische Möglichkeit zur Aktivierung der namensraumkonformen Verarbeitung hinaus setzen alle SAX-Implementierungen die Methode setFeature für die ParserFactory um.
Sie gestattet es durch standardisierte Namen (tatsächlich werden URIs verwendet) gewisse Eigenschaften an- und abzuschalten.
Die Funktionalität des Beispiels 100 ließe sich daher auch mit folgendem Code erreichen:

Beispiel 101: Namensraum-konformer SAX-Parser (nutzt SAX-Features)

```
(1)import org.xml.sax.Attributes;
(2)import org.xml.sax.helpers.DefaultHandler;
(3)import javax.xml.parsers.SAXParser;
(4)import javax.xml.parsers.SAXParserFactory;
(5)import java.util.HashMap;
(6)
(7)public class SAXExample42 extends DefaultHandler {
(8)    private HashMap elementHM;
(9)    private final Integer ONE = new Integer(1);
(10)
(11)    public void startElement(String namespaceURI, String localName, String qName, Attributes
atts) {
(12)        String elementFullName = new String(namespaceURI + ":" + localName);
(13)
(14)        Integer freq = (Integer) elementHM.get(elementFullName);
(15)
(16)        elementHM.put(elementFullName, (freq == null ? ONE : new Integer(freq.intValue() + 1)));
(17)    } //startElement()
(18)
(19)    public void endDocument() {
(20)        System.out.println(elementHM);
(21)    } //endDocument()
(22)
(23)
(24)    public static void main(String args[]) throws Exception {
(25)        SAXParserFactory spf = SAXParserFactory.newInstance();
(26)        spf.setFeature("http://xml.org/sax/features/namespaces", true);
(27)
(28)
(29)        SAXParser sp = spf.newSAXParser();
(30)
(31)        sp.parse(args[0], new SAXExample42());
(32)    } //main(args[])
(33)    public SAXExample42() {
(34)        elementHM = new HashMap();
(35)    } //constructor
(36)}//class SAXExample42
```

Download des Beispiels

Das Beispiel ersetzt den Aufruf von `setNamespaceAware` durch die Aktivierung der beiden Features `http://xml.org/sax/features/namespaces` und `http://xml.org/sax/features/namespace-prefixes` um dieselbe Funktionalität zu erreichen.

Einschließlich dieser Eigenschaften definiert die SAX2-Schnittstelle folgende Features:

Tabelle 26: SAX2-Features

Feature-URI	Funktionalität
`http://xml.org/sax/features/namespace`	Steuert die Berücksichtigung von Namensräumen
`http://xml.org/sax/features/namespace-prefixes`	Steuert ob die Namensraumdeklarations(pseudo-)attribute als den Attributen gleichgestellt behandelt werden
`http://xml.org/sax/features/string-interning`	Steuert den Aufruf der Methode `intern` zur Zeichenkettenverarbeitung
`http://xml.org/sax/features/validation`	Aktiviert bzw. Deaktiviert den validierenden Modus. Wird dieses Feature aktiviert, so müssen auch `external-general-entities` und `external-parameter-entities` aktiviert sein.
`http://xml.org/sax/features/external-general-entities`	Steuert die Inklusion externer Textentitäten
`http://xml.org/sax/features/external-parameter-entities`	Steuert die Inklusion externer Parameterentitäten sowie die Verarbeitung einer externen DTD

Fehlerbehandlung

Während der Arbeit mit der SAX-Schnittstelle können innerhalb zwei getrennter Operationsphasen Fehlersituationen auftreten. Zum einen während der Konstruktionsphase des SAX-basierten Parsers, d.h. vor Aufruf der Methode `parse`, zum anderen während des eigentlichen Parsingvorganges beim Einlesen des Dokuments.
Die Schnittstelle bietet die drei von `SAXException` abgeleiteten Ausnahmeereignisklassen an: `SAXNotRecognizedException`, `SAXParseException` und `SAXNotSupportedException`.

Fehler während der Parserkonstruktions- und Initialisierungsperiode rühren, abgesehen von externen Effekten, zumeist vom Versuch her, einen Parser in unzulässiger Weise zu parametrisieren.
Das Codebeispiel 102 zeigt dies anhand des Versuchs, eine (nicht existierende) durch die URL `http://www.jeckle.de` identifizierte Eigenschaft (engl. *Property*) des Parsers zu setzen. Während der Ausführung tritt konsequenterweise eine `SAXNotRecognizedException` auf.

Beispiel 102: Auftreten einer SAX-Exception

```
(1)import org.xml.sax.SAXException;
(2)import org.xml.sax.helpers.DefaultHandler;
(3)import javax.xml.parsers.SAXParser;
(4)import javax.xml.parsers.SAXParserFactory;
(5)
(6)
(7)public class SAXExample5 extends DefaultHandler {
(8)    public static void main(String args[]) throws Exception {
(9)        SAXParserFactory spf = SAXParserFactory.newInstance();
(10)
(11)        SAXParser sp = spf.newSAXParser();
(12)
(13)        try {
(14)            System.out.println(sp.getProperty("http://www.jeckle.de"));
(15)        } //try
(16)        catch (SAXException e) {
(17)            System.out.println("SAXException:" + e);
(18)        } //catch()
(19)        sp.parse(args[0], new SAXExample5());
(20)    } //main()
(21)}//class SAXExample5
```

Download des Beispiels

Tritt während des Parsingvorganges ein Verstoß hinsichtlich der well-formedness Regeln auf, so wird ein `SAXParseException`-Ausnahmeereignis erzeugt. Es stellt Methoden zur näheren Lokalisierung der Fehlerstelle im Eingabedokument zur Verfügung.
Die Implementierung des Beispiels 103 setzt dies um: Mittels der Methoden `getColumnNumber` und `getLineNumber` lassen sich Spalten- und Zeilennummer des Fehlers ermitteln. `getSystemId` und `getPublicId` können zur Ausgabe des System-Identifiers, der bei lokaler Referenzierung identisch zum Dateisystempfad ist, bzw. zur Ausgabe des Public-Identifiers -- falls gesetzt -- herangezogen werden.

Beispiel 103: Behandlung einer SAXParseException

```
(1)import org.xml.sax.SAXParseException;
(2)import org.xml.sax.helpers.DefaultHandler;
(3)import javax.xml.parsers.SAXParser;
(4)import javax.xml.parsers.SAXParserFactory;
(5)
(6)public class SAXExample6 extends DefaultHandler {
(7)    public static void main(String args[]) throws Exception {
(8)        SAXParserFactory spf = SAXParserFactory.newInstance();
(9)
(10)        SAXParser sp = spf.newSAXParser();
(11)
(12)        try {
(13)            sp.parse(args[0], new SAXExample6());
(14)        } //try
(15)        catch (SAXParseException spe) {
(16)            System.out.println("A SAXParseException occured ...\n" +
(17)                    "at column " + spe.getColumnNumber() + "\n" +
(18)                    "at line " + spe.getLineNumber() + "\n" +
(19)                    "public identifier of document is: " + spe.getPublicId() + "\n" +
(20)                    "system identifier of document is: " + spe.getSystemId());
(21)        } //catch()
(22)    } //main()
(23)}//class SAXExample6
```

Download des Beispiels

Das Auftreten einer `SAXNotSupportedException` wird durch den Versuch ausgelöst, eine zulässige und erkannte Parameterisierung des SAX-Parsers vorzunehmen, die durch diesen nicht unterstützt wird.
Diese Fehlersituation läßt sich daher an keinem statischen Beispiel zeigen, sondern hängt von der konkreten Parser-Implementierung ab.

Angewendet auf das nicht-wohlgeformte Dokument aus Beispiel 10 liefert die Ausführung die Ausgabe:

```
A SAXParseException occured ...
at column 17
at line 3
public identifier of document is: null
system identifier of document is: http://www.jeckle.de/vorlesung/xml/examples/notWellFormed.xml
```

Das **Abschlußbeispiel** 104 zeigt eine vollständige Implementierung der verschiedenen SAX-Ereignisse der `ConentHandeler`-Schnittstelle.
Als Eingabe diene die Beispieldatei:

```
(1)<?xml version="1.0"?>
```

```
(2)<?myPI this is a test?>
(3)   <html xmlns="http://www.w3.org/1999/xhtml" xmlns:svg="http://www.w3.org/2000/svg">
(4)   <head>
(5)      <title>Testpage</title>
(6)   </head>
(7)   <body>
(8)      <p>This is a <a href="http://www.jeckle.de">link</a>
(9)      <svg:svg width="4cm" height="8cm">
(10)         <svg:ellipse cx="2cm" cy="4cm" rx="2cm" ry="1cm"/>
(11)      </svg:svg>
(12)      </p>
(13)   </body>
(14)</html>
```

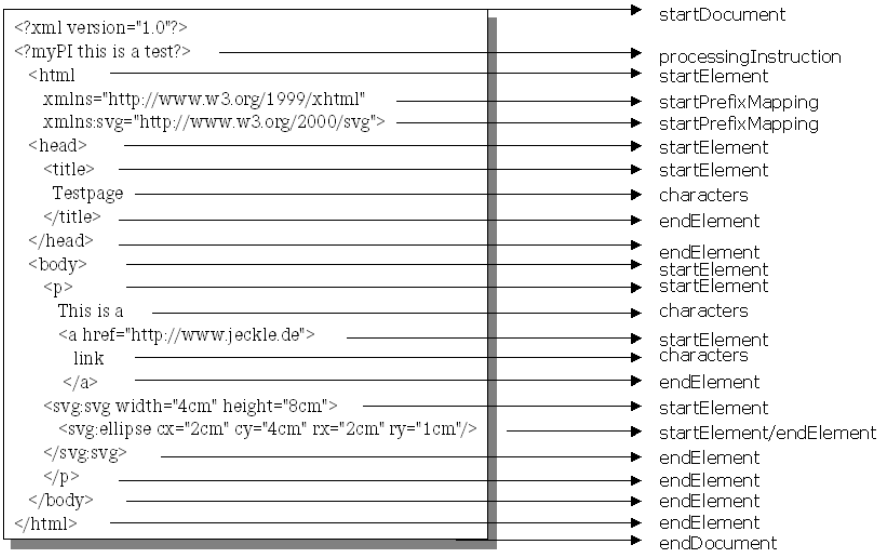
Beispiel 104: Implementierung verschiedener SAX2-Callback-Methoden

```
(1)import org.xml.sax.helpers.DefaultHandler;
(2)import org.xml.sax.Attributes;
(3)import javax.xml.parsers.SAXParserFactory;
(4)import javax.xml.parsers.SAXParser;
(5)import org.xml.sax.SAXParseException;
(6)
(7)public class SAXExample8 extends DefaultHandler {
(8)     public void startDocument()     {
(9)             System.out.println("EVENT: startDocument");
(10)    } //startDocument()
(11)
(12)    public void endDocument()       {
(13)            System.out.println("EVENT: endDocument");
(14)    } //endDocument()
(15)
(16)    public void startPrefixMapping(String prefix, String uri)    {
(17)    System.out.println("EVENT: startPrefixMapping");
(18)    System.out.println("prefix="+prefix);
(19)    System.out.println("namespaceURI="+uri);
(20)   } //startPrefixMapping()
(21)
(22)   public void endPrefixMapping(String prefix)    {
(23)    System.out.println("EVENT: endPrefixMapping");
(24)    System.out.println("prefix="+prefix);
(25)   } //endPrefixMapping()
(26)
(27)   public void processingInstruction(String target, String data)    {
(28)    System.out.println("EVENT: processingInstruction");
(29)    System.out.println("target="+target);
(30)    System.out.println("data="+data);
(31)   } //processingInstruction()
(32)
(33)    public void ignorableWhitespace(char[] ch, int start, int length)       {
(34)            System.out.println("EVENT: ignorableWhitespace");
(35)            for (int i=start; i<start+length; i++)
(36)                    System.out.print(ch[i]);
(37)            System.out.print("\n");
(38)    } //ignorableWhietespace()
(39)
(40)    public void startElement(String namespaceURI, String localName, String qName, Attributes
atts)   {
(41)            System.out.println("EVENT: startElement");
(42)            System.out.println("namespaceURI="+namespaceURI);
(43)            System.out.println("localName="+localName);
(44)            System.out.println("qName="+qName);
(45)
(46)            for (int i=0; i<atts.getLength(); i++)          {
(47)                    System.out.println("attribute:");
(48)                    System.out.println("localName="+atts.getLocalName(i));
(49)                    System.out.println("qName="+atts.getQName(i));
(50)                    System.out.println("type="+atts.getType(i));
(51)                    System.out.println("namespaceURI="+atts.getURI(i));
(52)                    System.out.println("value="+atts.getValue(i));
(53)            } //for
(54)    } //startElement()
(55)
(56)    public void endElement(String namespaceURI, String localName, String qName)     {
(57)            System.out.println("EVENT: endElement");
(58)            System.out.println("namespaceURI="+namespaceURI);
(59)            System.out.println("localName="+localName);
(60)            System.out.println("qName="+qName);
(61)    } //endElement()
(62)
(63)    public void characters(char[] ch, int start, int length)        {
(64)            System.out.println("EVENT: characters");
(65)            for (int i=start; i<start+length; i++)
(66)                    System.out.print(ch[i]);
```

```
(67)            System.out.print("\n");
(68)    } //characters()
(69)
(70)   public static void main (String args[]) throws Exception        {
(71)            SAXParserFactory spf = SAXParserFactory.newInstance();
(72)      spf.setNamespaceAware(true);
(73)            SAXParser sp = spf.newSAXParser();
(74)
(75)            try             {
(76)                    sp.parse( args[0], new SAXExample8() );
(77)            } catch (SAXParseException spe) {
(78)         System.out.println("A SAXParseException occured ...\n"+
(79)                            "at column "+spe.getColumnNumber()+"\n"+
(80)                            "at line "+spe.getLineNumber()+"\n"+
(81)                            "public identifier of document is: "+spe.getPublicId()+"\n"+
(82)                            "system identifier of document is: "+spe.getSystemId());
(83)      } //catch()
(84)    } //main(args[])
(85)}//class SAXExample8
```

Download des Beispiels

Die Ausführung liefert eine textuelle Ausgabe, deren Ereignisreihenfolge der der Abbildung 44 entspricht



Einfache Transformationen

Bedingt durch die sequentielle Aktivierung der verschiedenen Ereignisbehandlungsroutinen lassen sich mit SAX sehr komfortabel einfache Dokumenttransformationen, wie z.B. die Umbenennung von Elementen, realisieren.
Der Code aus Beispiel 105 zeigt die Umbenennung eines Elements von `foo` nach `bar`. Alle anderen Elemente, Attribute und Zeichenketten-artigen Elementinhalte werden unverändert kopiert.
Hinweis: Das Beispiel berücksichtigt dabei jedoch weder Namensräume noch Processing Instructions.

Beispiel 105: Umbenennung eines Elements

```
(1)import org.xml.sax.Attributes;
(2)import org.xml.sax.helpers.DefaultHandler;
(3)import javax.xml.parsers.SAXParser;
(4)import javax.xml.parsers.SAXParserFactory;
(5)
(6)public class SAXExample7 extends DefaultHandler {
(7)    public void startElement(String namespaceURI, String localName, String qName, Attributes
atts) {
(8)        if (qName == "foo") {
(9)            System.out.print("<bar");
(10)        } //if
(11)        else
(12)            System.out.print("<" + qName);
(13)
(14)        for (int i = 0; i < atts.getLength(); i++) {
(15)            System.out.print(" " + atts.getQName(i) + "=\"" + atts.getValue(i) + "\"");
(16)        } //for
(17)        System.out.println(">");
(18)    } //startElement()
(19)
(20)    public void endElement(String namespaceURI, String localName, String qName) {
(21)        if (qName == "foo") {
(22)            System.out.println("</bar>");
(23)        } //if
(24)        else
```

```
(25)            System.out.println("</" + qName + ">");
(26)    } //endElement()
(27)
(28)    public void characters(char[] ch, int start, int length) {
(29)        for (int i = start; i < start + length; i++)
(30)            System.out.print(ch[i]);
(31)    } //characters()
(32)
(33)    public static void main(String args[]) throws Exception {
(34)        SAXParserFactory spf = SAXParserFactory.newInstance();
(35)
(36)        SAXParser sp = spf.newSAXParser();
(37)
(38)        sp.parse(args[0], new SAXExample7());
(39)    } //main()
(40)}//class SAXExample7
```

Download des Beispiels

Zur Realisierung komplexer Transformationen, insbesondere solcher, die die Zwischenspeicherung von Dokumentinformationen erfordern sind jedoch Ansätze mit expliziter Abbildung in Hauptspeicherstrukturen wie DOM oder XSLT besser geeignet.

Einbindung von SAX in Applikationsprogramme

Beispiel 106 zeigt die Einbindung eines SAX-Parser in ein Applikationsprogramm. Die Anwendung überführt beliebige XML-Eingabedokumente in eine Java-SWING-konforme Baumdarstellung.

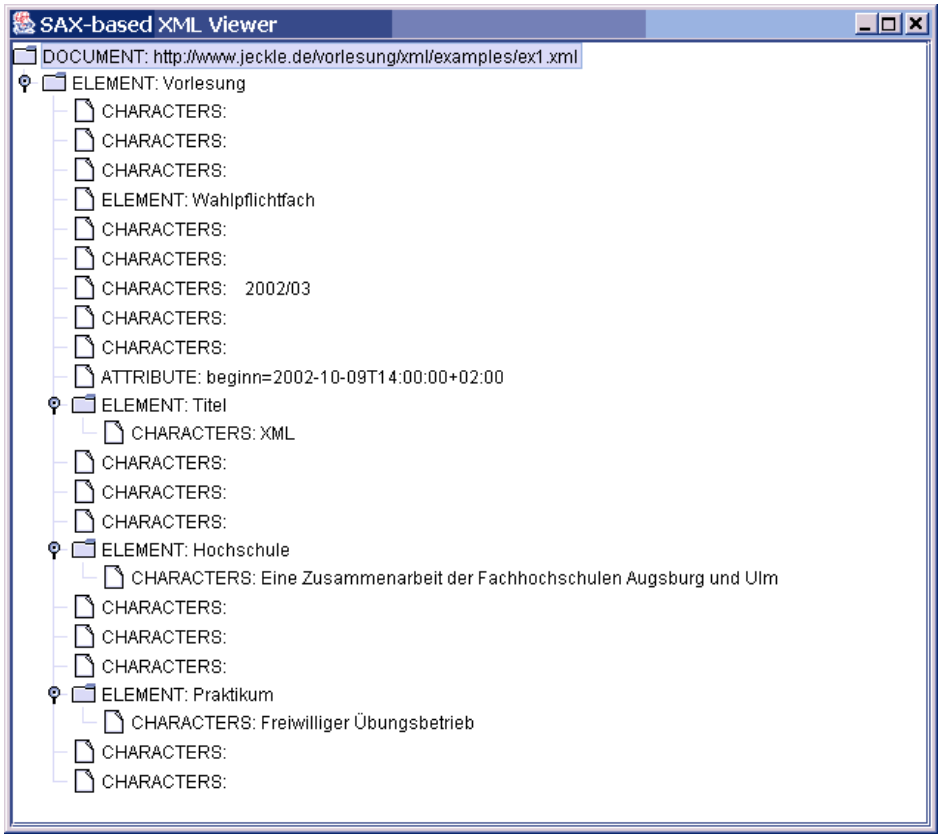
Beispiel 106: Konstruktion einer SWING-basierten Baumansicht mit SAX

```
(1)import org.xml.sax.Attributes;
(2)import org.xml.sax.SAXException;
(3)import org.xml.sax.helpers.DefaultHandler;
(4)import javax.swing.JFrame;
(5)import javax.swing.JScrollPane;
(6)import javax.swing.JTree;
(7)import javax.swing.tree.DefaultMutableTreeNode;
(8)import javax.xml.parsers.FactoryConfigurationError;
(9)import javax.xml.parsers.ParserConfigurationException;
(10)import javax.xml.parsers.SAXParser;
(11)import javax.xml.parsers.SAXParserFactory;
(12)import java.awt.BorderLayout;
(13)import java.awt.HeadlessException;
(14)import java.awt.event.WindowAdapter;
(15)import java.awt.event.WindowEvent;
(16)import java.io.IOException;
(17)
(18)public class TreeViewer extends JFrame {
(19)
(20)    public static void main(String argv[]) {
(21)        JFrame frame = new TreeViewer(argv[0]);
(22)        frame.addWindowListener(new WindowAdapter() {
(23)            public void windowClosing(WindowEvent e) {
(24)                System.exit(0);
(25)            }
(26)        });
(27)        frame.pack();
(28)        frame.setVisible(true);
(29)    } //main()
(30)
(31)    public TreeViewer(String XMLFile) {
(32)        super("SAX-based XML Viewer");
(33)        try {
(34)            DefaultMutableTreeNode root = new DefaultMutableTreeNode("DOCUMENT: " + XMLFile);
(35)            JTree tree = new JTree(root);
(36)            JScrollPane treeView = new JScrollPane(tree);
(37)            getContentPane().add(new JScrollPane(treeView), BorderLayout.CENTER);
(38)
(39)            //add elements to root
(40)            SAXParserFactory spf = SAXParserFactory.newInstance();
(41)            spf.setNamespaceAware(true);
(42)            SAXParser sp = spf.newSAXParser();
(43)            sp.parse(XMLFile, new Consumer(root));
(44)        } catch (HeadlessException e) {
(45)            System.out.println("EXCEPTION CAUGHT: " + e.getClass().getName());
(46)            e.printStackTrace();
(47)        } catch (FactoryConfigurationError factoryConfigurationError) {
(48)            System.out.println("EXCEPTION CAUGHT: " + factoryConfigurationError.getClass().
getName());
(49)            factoryConfigurationError.printStackTrace();
(50)        } catch (ParserConfigurationException e) {
```

```
(51)            System.out.println("EXCEPTION CAUGHT: " + e.getClass().getName());
(52)            e.printStackTrace();
(53)        } catch (SAXException e) {
(54)            System.out.println("EXCEPTION CAUGHT: " + e.getClass().getName());
(55)            e.printStackTrace();
(56)        } catch (IOException e) {
(57)            System.out.println("EXCEPTION CAUGHT: " + e.getClass().getName());
(58)            e.printStackTrace();
(59)        }
(60)    }
(61)} //class TreeViewer
(62)
(63)class Consumer extends DefaultHandler {
(64)    private DefaultMutableTreeNode current;
(65)
(66)    public Consumer(DefaultMutableTreeNode root) {
(67)        current = root;
(68)    }
(69)
(70)    public void startElement(String namespaceURI, String localName, String qName, Attributes
atts) {
(71)        for (int i = 0; i < atts.getLength(); i++) {
(72)            current.add(new DefaultMutableTreeNode("ATTRIBUTE: " + atts.getQName(i) + "=" +
atts.getValue(i)));
(73)        }
(74)
(75)        DefaultMutableTreeNode newNode = new DefaultMutableTreeNode("ELEMENT: " + localName);
(76)        current.add(newNode);
(77)        current = newNode;
(78)    }
(79)
(80)    public void endElement(String namespaceURI, String localName, String qName) {
(81)        current = (DefaultMutableTreeNode) current.getParent();
(82)    }
(83)
(84)    public void processingInstruction(String target, String data) {
(85)        current.add(new DefaultMutableTreeNode("PROCESSING INSTRUCTION: " + target));
(86)    }
(87)
(88)    public void characters(char[] ch, int start, int length) {
(89)        current.add(new DefaultMutableTreeNode("CHARACTERS: " + new String(ch, start, length)));
(90)    }
(91)} //class Consumer
```

Download des Beispiels

Der Code zeigt die sukzessive Konstruktion der Baumansicht entlang der beim Lesevorgang eintretenden SAX-Ereignisse.
Hervorzuheben ist hierbei die Erzeugung je eines Baumknotens innerhalb der Ereignisbehandlungsroutinen `startElement`, `processingInstruction` und `characters`. All diese Methoden fügen einen neuen Kindknoten zum aktuell bearbeiteten Baumknoten zu. Zusätzlich wird innerhalb der Behandlung des `startElement`-Ereignisses der neu erzeugte Kindknoten für die weitere Verarbeitung als Aktueller definiert und damit eine zusätzliche Baumstufe eröffnet.
Das rekursive Aufsteigen im Baum findet beim Verlassen eines Elements (Ereignis: `endElement`) statt.
Abbildung 45 zeigt die durch Verarbeitung des Dokuments aus Beispiel 1 erzeugte Bildschirmansicht.



Anmerkung zur Graphik: Die vermeintlich „leeren" CHARACTER-Elemente entstehen durch die nichtdruckbaren Zeichen wie Zeilenumbrüche und Wagenrückläufe.

Abschlußbemerkungen und Einsatzempfehlungen

SAX offenbart sich als leicht einzusetzende und trotzdem für geeignete Anwendungsfälle sehr mächtige Schnittstelle. Insbesondere ist der serielle Verarbeitungsansatz, der nur geringe Hauptspeicheranforderungen stellt, sehr gut für große XML-Dokumente geeignet. Gleichzeitig skalieren SAX-basierte Anwendungen vergleichsweise gut, da das Eingabedokument nur einmal durchlaufen wird.
Als gravierende Nachteile offenbaren sich jedoch die fehlenden Navigationsmöglichkeiten, die der Applikation die Reihenfolge der Elemente im Dokument als Verarbeitungsreihenfolge aufzwingen.
Festzuhalten bleibt, daß es sich bei SAX lediglich um eine Schnittstelle handelt, auf der Parser realisiert werden können. SAX selbst ist jedoch kein solcher.

Web-Referenzen 17: Weiterführende Links

•SAX-Ur-Seite von David Megginson
SAX-Implementierungen:
•C++
•Perl
•Eiffel
•SmallTalk
•Top Ten SAX2 Tips
•W. Scott Means, Michael A. Brody: *The Book of SAX: The Simple API for XML*

3.2 Das *Document Object Model*

Die W3C-Spezifikation des *Document Object Model*s (abgekürzt als: DOM) definiert eine Programmiersprachen-unabhängig formulierte Menge abstrakter Schnittstellen zum lesenden und schreibenden Zugriff auf gültige HTML und wohlgeformte XML-Dokumente sowie eine Reihe weiterer Formate.
Derzeit verabschiedet (Status einer W3C-Recommendation) ist das sog. DOM Level 1 und die darauf aufsetzende Spezifikation eines DOM Level 2. Aktuell wird bereits an der als DOM Level 3 bezeichneten Weiterentwicklung gearbeitet. Diese Bemühungen haben jedoch erst den Stand eines *working draft*s erreicht.
DOM Level 2 erweitert die in Level 1 eingeführten Schnittstellen hinsichtlich der Erfordernisse des aktuellen XML-Standes um Namensräume und bietet einige neue Operationen, die seitens der Anwendergemeinde gefordert wurden. Die bisherigen Operationen existieren aus Kompatibilitätsgründen weiter, die Namespace-berücksichtigenden Pendants sind identisch benannt, jedoch um ein angehängtes *NS* erweitert.

DOM versteht sich als *Application Programming Interface* (API) für beliebige XML-Dokumente. Hierzu versammelt es auf Basis einer generischen Speicherrepräsentation für XML eine Menge von Operationen zur Extraktion verschiedenster Informationen aus dem Dokument, sowie zur Modifikation der speicherresidenten Strukturen und späteren (Wieder-)Ausgabe in ein XML-Dokument.

Die Konformität zur DOM-Spezifikation fächert sich in die verschiedenen DOM-Module auf. Wird eines der Module implementiert, so spricht man von *DOM 2 module* Unterstützung für das jeweilige Modul. Wird das Kernmodul *core*

unterstützt, so ist *DOM 2* Unterstützung erreicht.
Die einzelnen DOM-Module sind in Tabelle 27 zusammengestellt.

Tabelle 27: Übersicht der DOM-Module

Modul	Aufgabe / Funktionsumfang
Core	Definiert programmiersprachenunabhängige Schnittstellen für den Zugriff auf, und die Manipulation von, Objekten eines Dokuments.
XML	Definiert den über HTML hinausgehenden Teil von Core.
Views	Schnittstellen zur Realisierung verschiedener Sichten auf dasselbe Dokument. Dieses Modul ist optional; im Falle einer Implementierung muß auch *Core* umgesetzt werden.
StyleSheets	Schnittstellen zum Zugriff auf Informationen eines Stylesheets. Dieses Modul ist optional; im Falle einer Implementierung muß auch *Core* umgesetzt werden.
CSS	Schnittstellen zum Zugriff auf Informationen einer CSS-Beschreibung. Dieses Modul ist optional; im Falle einer Implementierung muß auch *Core* und *Views* umgesetzt werden.
CSS2	Schnittstellen zum Zugriff Informationen einer CSS2-Beschreibung. Dieses Modul ist optional; im Falle einer Implementierung muß auch *CSS* umgesetzt werden.
Events	Schnittstellen zur plattform- und sprachunabhängigen Behandlung von Ereignissen. Implementierungen dieses Moduls müssen auch *Core* umsetzen.
UIEvents	Umfaßt eine Untermenge von *Events*, die den HTML-spezifischen Ereignissen aus DOM 0 entspricht. Implementierungen müssen zusätzlich *Events* und *Views* unterstützen.
MouseEvents	Umfaßt eine Untermenge von *Events*, die den HTML-spezifischen Maus-Ereignissen aus DOM 0 entspricht. Implementierungen müssen zusätzlich die Schnittstellen aus *UIEvents* unterstützen.
MutationEvents	Bietet Schnittstellen zur Benachrichtigung bei Ereignissen, durch welche Struktur oder Inhalt eines Dokuments verändert werden (z.B. Hinzufügen oder Ändern eines Elements oder Attributs). Implementierungen müssen zusätzlich die Schnittstellen aus *Events* realisieren.
Range	Schnittstellen zum Zugriff auf Bereiche eines XML-Dokuments, wie sie beispielsweise durch Selektionen mit der Maus am Bildschirm in der visuellen Darstellung gebildet werden können. Implementierungen müssen zusätzlich die Schnittstellen aus *Core* realisieren.
Traversal	Bietet mit den Schnittstellen `TreeWalker`, `NodeIterator` und `NodeFilter` Möglichkeiten zur einfachen Traversierung von Dokumentbäumen an. Implementierungen müssen zusätzlich die Schnittstellen aus *Core* realisieren.

Die Unterstützung der einzelnen Module kann durch die Schnittstellenfunktion `hasFeature` der Schnittstelle `DOMImplementation` des Moduls `Core` ermittelt werden.
Die Java-Implementierung aus Beispiel 107 zeigt die Nutzung dieser Operation. DOM definiert zwar die Operationssignatur, gebildet aus dem Modulnamen und dessen Version, legt jedoch keine Klasse zur Implementierung fest. Das Beispiel zeigt die Umsetzung für die DOM-Unterstützung im Java Development Kit v1.4

Beispiel 107: Ermittlung der unterstützten DOM-Module

```
(1)import org.w3c.dom.DOMImplementation;
(2)import javax.xml.parsers.DocumentBuilder;
(3)import javax.xml.parsers.DocumentBuilderFactory;
(4)
(5)public class domtest {
(6)    public static void main(String argv[]) throws Exception {
(7)        // implementation specific part begins ...
(8)        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
(9)        DocumentBuilder builder = factory.newDocumentBuilder();
(10)        DOMImplementation implementation = builder.getDOMImplementation();
(11)              // ... ends
(12)
(13)        String feature[] = {"Core", "XML", "HTML", "Views", "StyleSheets",
(14)                            "CSS", "CSS2", "Events", "UIEvents", "MouseEvents",
(15)                            "MutationEvents", "HTMLEvents", "Range", "Traversal"};
(16)
(17)        for (int i = 0; i < feature.length; i++) {
(18)            System.out.print(feature[i]);
(19)            if (!implementation.hasFeature(feature[i], "2.0")) {
(20)                System.out.println(" not supported");
(21)            } //if
(22)            else {
(23)                System.out.println(" supported");
(24)            } //else
(25)        } //for
(26)    } //main()
(27)} //class domtest
```

Download des Beispiels

Die aktuell verfügbare Umsetzung des JDK v1.4.1 liefert daher die Ausgabe:

```
(1)Core supported
(2)XML supported
(3)HTML not supported
(4)Views not supported
(5)StyleSheets not supported
(6)CSS not supported
(7)CSS2 not supported
(8)Events supported
(9)UIEvents not supported
(10)MouseEvents not supported
(11)MutationEvents supported
(12)HTMLEvents not supported
(13)Range supported
(14)Traversal supported
```

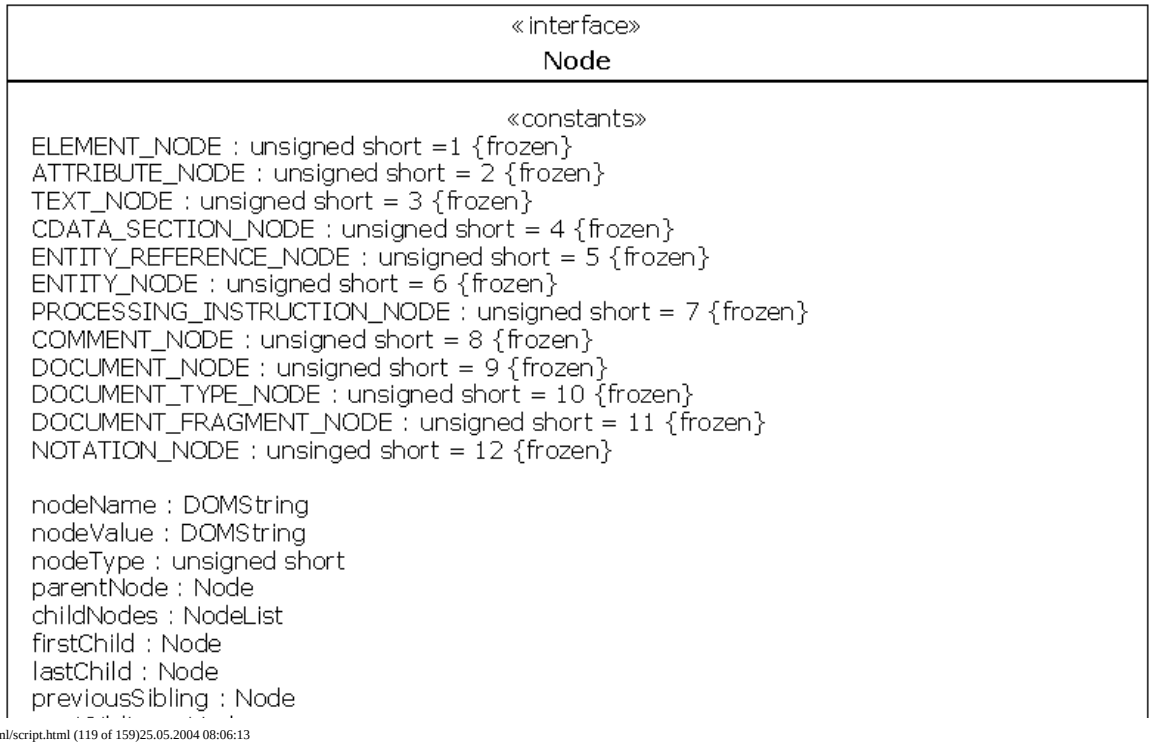
Das DOM-Objektmodell

Das in CORBA-IDL formulierte DOM-Objektmodell orientiert sich strukturell stark am XML Information Set. DOM erweitert die dort definierte Baum-artige Struktur um Signatur- und Semantikdefinitionen zur Operation auf den erzeugten Speicherobjekten. Der Begriff *Objektmodell* wurde daher bewußt in Abgrenzung zum *Datenmodell* gewählt, da durch DOM neben den Datenstrukturen auch darauf aufbauende gekapselte Funktionalität definiert wird.
Jedoch definiert DOM keine Binärrepräsentation der verarbeiteten Dokumentdaten. DOM-implementierende Applikationen können daher durchaus in ihrer Implementierung variieren, insbesondere ist durch DOM nicht die Herstellung einer binären Interoperabilität beabsichtigt.
Ebenso gibt die DOM-Level-1-Spezifikation keinen Weg zur Erzeugung des Objektmodells aus einem XML-Dokument oder zur Ausgabe des Objektmodells in ein XML-Dokument an. Hierfür haben die einzelnen Hersteller in ihren Implementierungen teilweise abweichende Lösungen entwickelt. Zumeist wird das DOM-Objektmodell sukzessive aus den gelieferten Ereignissen eines SAX-Parsers erzeugt.
Dieses „verborgene" Implementierungsdetail läßt sich vergleichsweise leicht durch Auffangen der `SAXParseException`-Ausnahme in einer DOM-basierten Parserumgebung prüfen. Tritt während des Konstruktionsvorganges der DOM-Repräsentation ein SAX-Parserfehler ein, so wird spezifikationsgemäß ein Ausnahmeereignisobjekt dieses Typs generiert.
Durch die Orientierung am InfoSet-Ansatz gleichen sich von verschiedenen DOM-Implementierungen erzeugte Speicherstrukturen immer strukturell. Diese Eigenschaft wird als *strukturelle Isomorphie* bezeichnet.
DOM Level 1 zerfällt in zwei Teile: dem Paket *DOM HTML* zur Operation auf HTML strukturierten Dokumenten und *DOM Core* zur Verarbeitung nativer XML-Dokumente. Im folgenden wird ausschließlich das DOM-Kernpaket der aktuellen Spezifikation *Level 2* betrachtet. Die Implementierungsbeispiele orientieren sich an SUNs Umsetzung für JDK v1.4.

Die fundamentalen DOM-Schnittstellen

Node

Die Schnittstelle `Node` konzentriert alle gemeinsamen Anteile der verschiedenen Knoten eines XML-Baumes. Hierunter fallen: Das Dokument selbst, alle darin enthaltenen Attribute, Elemente, Kommentare und Textelemente sowie weitere durch die InfoSet-Spezifikation definierte Primitive.
Das Diagramm der Abbildung 46 zeigt die Schnittstelle mit den durch sie definierten Konstanten zur Codierung der Knotentypisierung, ihnen entspricht jeweils eine eigene DOM-Schnittstelle, neben einigen grundlegenden Attributen zur vereinfachten Navigation. Die aufgeführten Operationen erlauben einige Veränderungen der Knotenstruktur wie das Einfügen oder Anhängen neuer Knoten in ein bestehendes DOM-Objektmodell. Darüberhinaus das Ersetzen, Löschen und Kopieren existierender Knoten.
Die Schnittstelle `Node` wird von allen im folgenden diskutierten nach Knotentyp spezialisierten Schnittstellen erweitert.

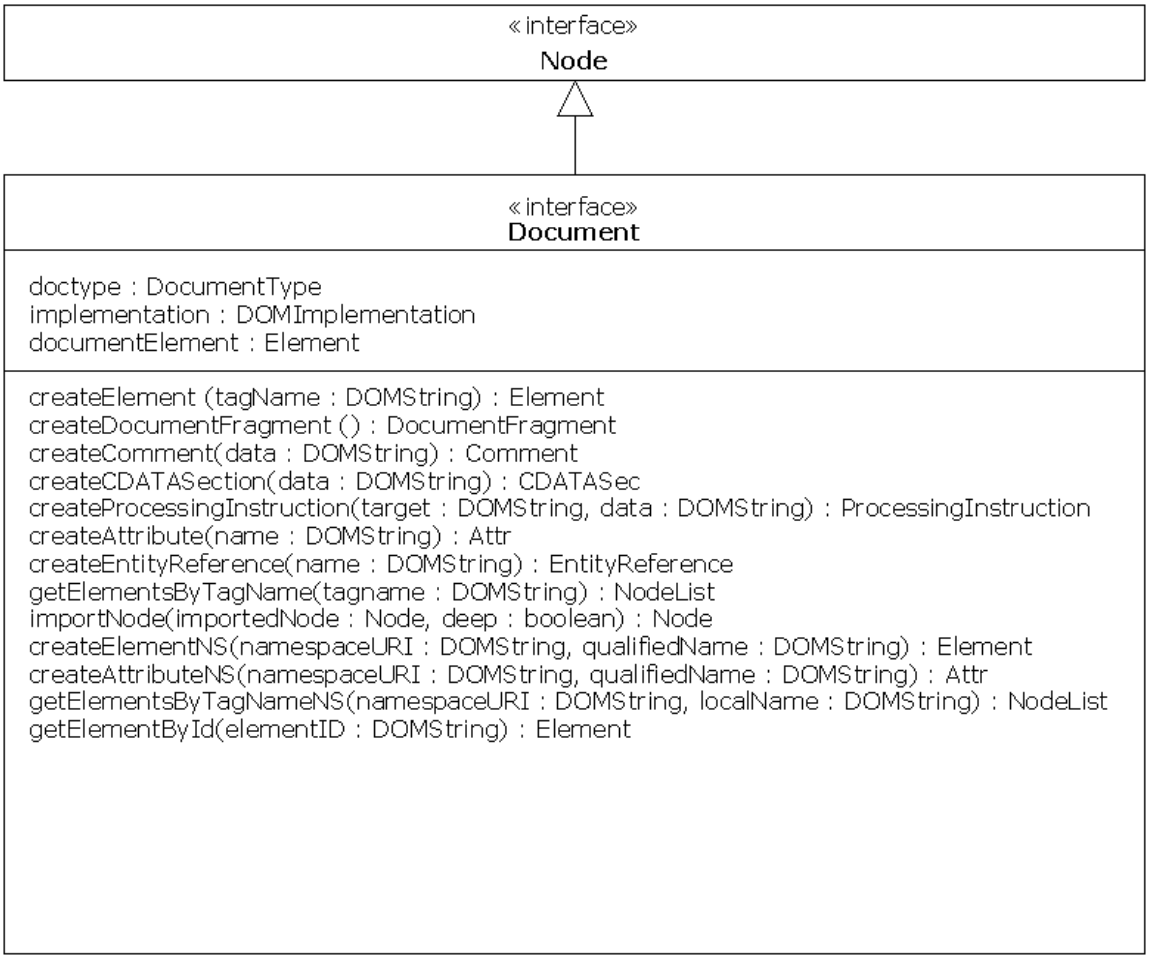


previousSibling : Node
nextSibling : Node
attributes : NamedNodeMap
ownerDocument : Document
namespaceURI : DOMString
prefix : DOMString

insertBefore(newChild : Node, refChild : Node) : Node
replaceChild(newChild : Node, oldChild : Node) : Node
removeChild(oldChild : Node) : Node
appendChild(newChild : Node) : Node
hasChildNodes() : boolean
cloneNode(deep : boolean) : Node
normalize()
isSupported(feature : DOMString, version : DOMString) : boolean
hasAttributes() : boolean

Document

Die Schnittstelle `Document` versammelt die in der Graphik der Abbildung 47 dargestellten Eigenschaften und Operationen.
Sie bildet als Basis den Einstiegspunkt jeder DOM-Baumstruktur.
Abgesehen vom Datentyp `DOMString`, der als Zeichenkette mit jeweils 16-Bit zur Codierung der Einzelzeichen definiert ist, sind alle verwendeten Datentypen selbst Schnittstellen die durch DOM definiert werden.



Der Code des Beispiels 108 zeigt die Erzeugung einer DOM-basierten Speicherstruktur mit SUNs JDK.
Die Pakete zur Realisierung des Einlesevorganges sind in der Hierarchie `javax.xml.parsers` organisiert. Analog zur Erzeugungs- und Initialisierungsphase eines SAX-Parsers wird auch für DOM zunächst eine *factory*-Instanz gebildet, die den Parser (ein Objekt der Klasse `DocumentBuilder`) liefert.
Die Methode `parse` erzeugt ein `Document`-Objekt gemäß der W3C Spezifikation (Level 2). Die entsprechenden Schnittstellendefinitionen sind im Paket `org.w3c.dom` zusammengefaßt.

Beispiel 108: Ein einfacher DOM-basierter Parser

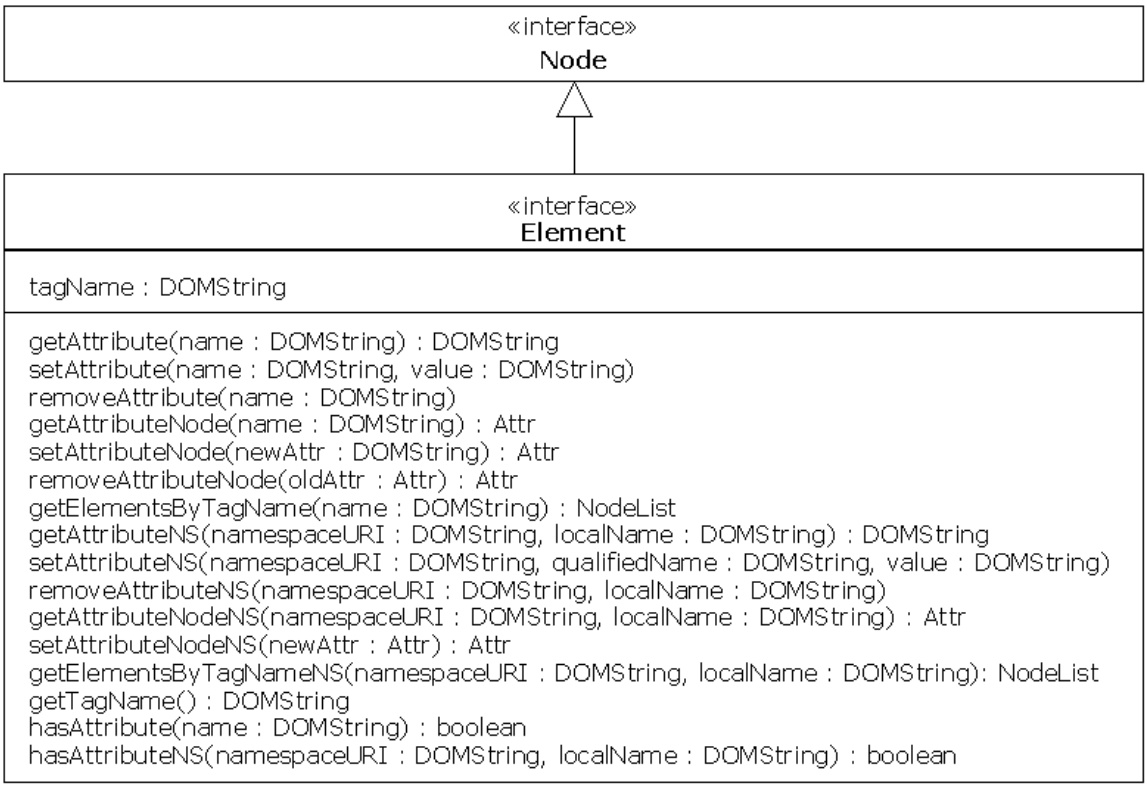
```
(1)import org.w3c.dom.Document;
(2)import javax.xml.parsers.DocumentBuilder;
(3)import javax.xml.parsers.DocumentBuilderFactory;
(4)
(5)public class DOMExample1 {
(6)    public static void main(String[] args) throws Exception {
(7)        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
(8)        DocumentBuilder builder = factory.newDocumentBuilder();
(9)        Document document = builder.parse(args[0]);
(10)    } //main()
(11)} //class DOMExample1
```

Download des Beispiels

Der Zugriff auf das Wurzelelement des verarbeiteten Dokuments wird über das Attribut `documentElement` bereitgestellt. Es liefert ein Objekt des Typs `Element`.

Element

Die Schnittstelle `Element` erweitert `Node` um ein Element und eine Reihe von Operationen zum Zugriff auf XML-Elemente.
Die Graphik stellt diese als UML-Klassendiagramm dar:



Beispiel 109 zeigt den Einsatz der Operation `hasAttributes`, die auf Existenz von Attributen eines gegebenen Elements testet.
Zusätzlich nutzt das Beispiel zwei Methoden, denen keine DOM2-Operationen entsprechen: `getTagName` und `getDocumentElement`. Dies rührt aus der Umsetzung der DOM2-Schnittstellendefinition in Java her. Diese Programmiersprache erlaubt in Schnittstellen keine änderbaren Attribute, sondern lediglich Konstanten. Daher stellt die DOM-Implementierung des JDK für diese Attribute eigene Zugriffsmethoden zur Verfügung.

Beispiel 109: Zugriff auf Elementinformation mit DOM2

```
(1)import org.w3c.dom.Document;
(2)import org.w3c.dom.Element;
(3)import javax.xml.parsers.DocumentBuilder;
(4)import javax.xml.parsers.DocumentBuilderFactory;
(5)
(6)public class DOMExample2 {
(7)    public static void main(String[] args) throws Exception {
(8)        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
(9)        DocumentBuilder builder = factory.newDocumentBuilder();
(10)        Document document = builder.parse(args[0]);
(11)
(12)        Element theRootElement = document.getDocumentElement();
(13)
(14)        System.out.println("root element's name: " + theRootElement.getTagName());
(15)
(16)        System.out.print("the element has");
(17)        if (!theRootElement.hasAttributes())
(18)            System.out.print(" no");
```

```
(19)        System.out.println(" attributes");
(20)    } //main()
(21)} //class DOMExample2
```

Download des Beispiels

Angewandt auf die XML-Datei der Projektverwaltung liefert das Programm die Ausgabe:

```
root element's name: ProjektVerwaltung the element has attributes
```

Darüberhinaus gestattet DOM -- im Gegensatz zu SAX -- Modifikationen am eingelesenen Dokument. So wird im nachfolgenden Beispiel dem Wurzelelement des Dokuments (das Objekt `theRootElement` des Typs `Element`) ein Attribut (benannt mit `myFirstNewAttribute`) hinzugefügt und mit dem Wert `01` belegt.
Anschließend wird durch den Aufruf `createElement` auf dem Dokumentobjekt ein neues Element erzeugt, das jedoch noch nicht im Dokumentbaum plaziert wird. Dies geschieht durch die Methode `appendChild`.
Der Aufruf der Methode `String.println` serialisiert den DOM-Baum in eine Zeichenkettenrepräsentation und gibt diese über die Standardausgabe aus.

Beispiel 110: Modifikationen am Dokument mittels DOM2

```
(1)import org.w3c.dom.Document;
(2)import org.w3c.dom.Element;
(3)import javax.xml.parsers.DocumentBuilder;
(4)import javax.xml.parsers.DocumentBuilderFactory;
(5)
(6)public class DOMExample3 {
(7)    public static void main(String[] args) throws Exception {
(8)        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
(9)        DocumentBuilder builder = factory.newDocumentBuilder();
(10)        Document document = builder.parse(args[0]);
(11)
(12)        Element theRootElement = document.getDocumentElement();
(13)
(14)        theRootElement.setAttribute("myFirstNewAttribute", "01");
(15)
(16)        Element aNewElement = document.createElement("myNewElement");
(17)
(18)        theRootElement.appendChild(aNewElement);
(19)
(20)        System.out.print(theRootElement);
(21)    } //main()
(22)} //class DOMExample3
```

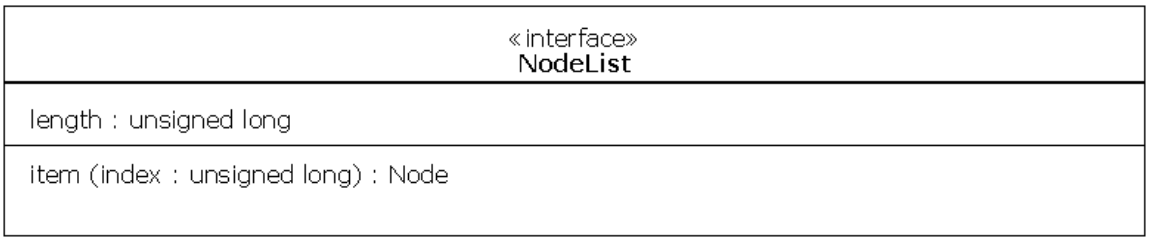
Download des Beispiels

Wird das Programm auf einem XML-Dokument ausgeführt, das ausschließlich aus dem Element `empty` besteht, liefert es die Ausgabe:

```
<empty myFirstNewAttribute="01"><myNewElement /></empty>
```

NodeList

Die Schnittstelle `NodeList` definiert einen Container zur Aufnahme beliebiger Objekte des Typs `Node`. Sie definiert das Attribut `length`, welches zu jedem Zeitpunkt die Anzahl der verwalteten Elemente enthält. Der Zugriff auf die verwalteten `Nodes` erfolgt indexsequentiell durch die Methode `item`.



Das Codebeispiel 111 zeigt die Verwendung der Schnittstelle zur Ermittlung der Auftrittsanzahl eines als Kommandozeilenparameter übergebenen Elementnamens.
Die Methode `getElementsByTagName` der Schnittstelle `Document` fügt während der Ausführung alle Auftreten von Elementen mit dem gesuchten Namen in die Resultatmenge ein, unbeachtlich deren Position im Dokument.

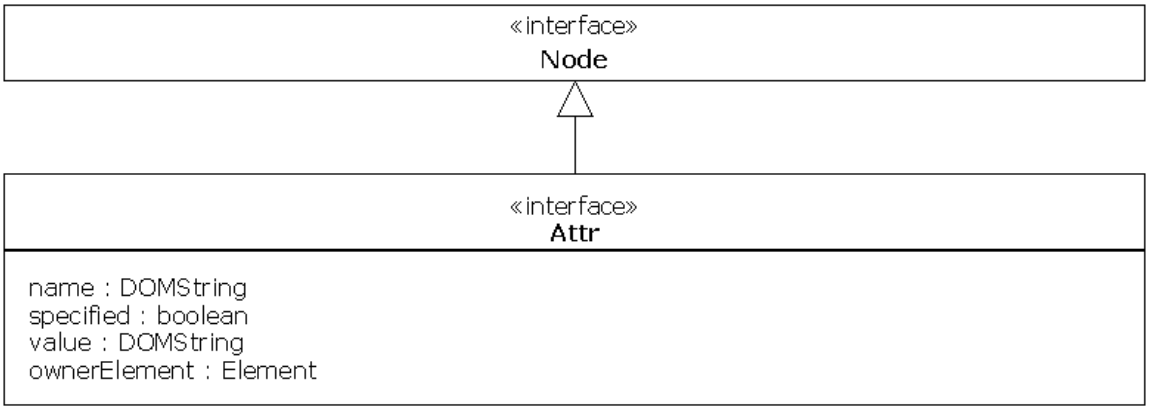
Beispiel 111: Nutzung der Schnittstelle NodeList

```
(1)import org.w3c.dom.Document;
(2)import org.w3c.dom.Element;
(3)import org.w3c.dom.NodeList;
(4)import javax.xml.parsers.DocumentBuilder;
(5)import javax.xml.parsers.DocumentBuilderFactory;
(6)
(7)public class DOMExample4 {
(8)    public static void main(String[] args) throws Exception {
(9)        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
(10)        DocumentBuilder builder = factory.newDocumentBuilder();
(11)        Document document = builder.parse(args[0]);
(12)
(13)        Element theRootElement = document.getDocumentElement();
(14)
(15)        NodeList nodes = theRootElement.getElementsByTagName(args[1]);
(16)
(17)        System.out.println("document contains " + nodes.getLength() + " elements of type " +
args[1]);
(18)    } //main()
(19)} //class DOMExample4
```

Download des Beispiels

Attr

Die Schnittstelle `Attr` erweitert `Node` um drei Attribute zur Abbildung der Charakteristika eines XML-Attributs.
Die Graphik stellt diese als UML-Klassendiagramm dar:



Die Attribut-spezifischen Eigenschaften werden durch den Attributnamen (`name`) und seinen Wert (`value`) abgebildet. Ferner ist verfügbar, ob es sich um ein im eingelesenen Quelldokument auftretendes Attribut handelt, oder durch den Parser der in der DTD oder dem Schema festgelegte Vorgabewert geliefert wird. Einen Verweis auf das umgebende Element liefert das Attribut `ownerElement`.

Das Beispiel 112 zeigt die Ermittlung verschiedener Attribut-bezogener Informationen. Als Eingabe dient eine um die `DOCTYPE`-Deklaration erweiterte Variante der Projektverwaltung.
Zunächst werden alle Elemente des Typs `Projekt` in einer `NodeList` zusammengestellt. Danach werden alle Elemente der Knotenmenge durchlaufen, und im Falle der Existenz (geprüft mit `hasAttributes`) verschiedene Charakteristika des Attributs ausgegeben.
Während die Auswertungen zum `ID`-Attribut direkt aus dem XML-Eingabedokument ersichtlich sind, nutzt der XML-Prozessor zur Ermittlung der Informationen über budget die referenzierte Dokument Typ Deklaration. In ihr ist das genannte Attribut mit dem Vorgabwert 10000 definiert, der an die Applikation zurückgegeben wird, falls keine andere Belegung im Dokument gefunden wird. Dies ist für beide `Projekt`-Elemente der Fall.
Verfügt das Eingabedokument über keine `DOCTYPE`-Deklaration, so kann diese Information nicht ausgewertet werden.

Beispiel 112: Zugriff auf Attributinformation

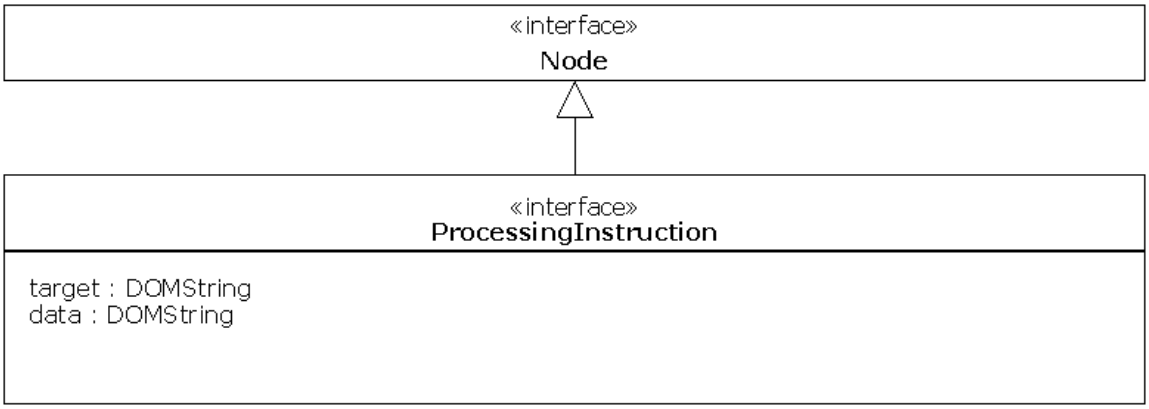
```
(1)import org.w3c.dom.Attr;
(2)import org.w3c.dom.Document;
(3)import org.w3c.dom.Element;
(4)import org.w3c.dom.NodeList;
(5)import javax.xml.parsers.DocumentBuilder;
(6)import javax.xml.parsers.DocumentBuilderFactory;
(7)
(8)public class DOMExample5 {
(9)    public static void main(String[] args) throws Exception {
(10)        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
(11)        DocumentBuilder builder = factory.newDocumentBuilder();
(12)        Document document = builder.parse(args[0]);
(13)
(14)        Element theRootElement = document.getDocumentElement();
(15)
(16)        NodeList projects = theRootElement.getElementsByTagName("Projekt");
(17)
(18)        for (int i = 0; i < projects.getLength(); i++) {
```

```
(19)            Element theElement;
(20)            if ((theElement = (Element) projects.item(i)).hasAttributes()) {
(21)                Attr theAttribute = theElement.getAttributeNode("ID");
(22)                System.out.println("Attribute: " + theAttribute.getName());
(23)                System.out.println("value=" + theAttribute.getValue());
(24)                System.out.println("specified=" + theAttribute.getSpecified());
(25)
(26)                theAttribute = theElement.getAttributeNode("budget");
(27)                System.out.println("Attribute: " + theAttribute.getName());
(28)                System.out.println("value=" + theAttribute.getValue());
(29)                System.out.println("specified=" + theAttribute.getSpecified());
(30)            }//if
(31)        }//for
(32)    } //main()
(33)} //class DOMExample5
```

Download des Beispiels

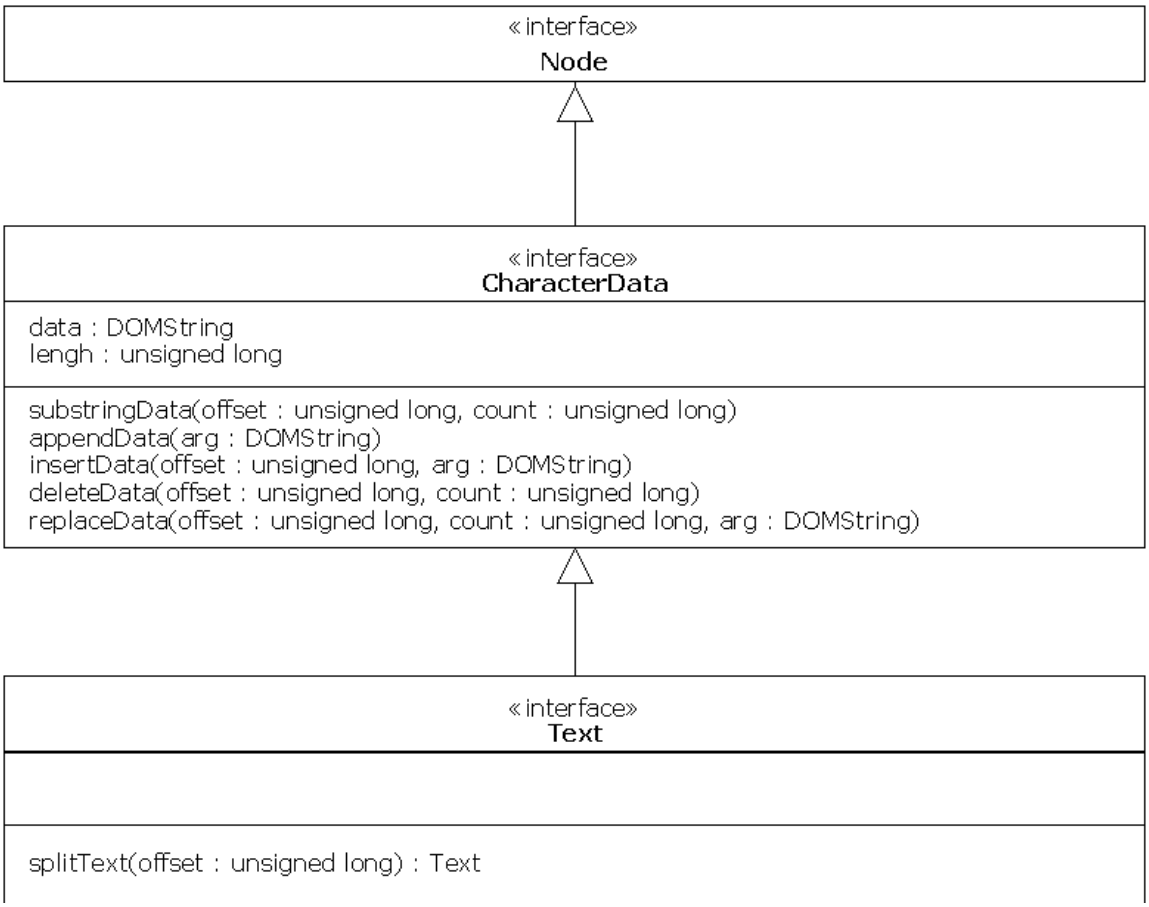
ProcessingInstruction

Die Schnittstelle `ProcessingInstruction` erweitert Node um zwei Attribute zur Abbildung der Charakteristika einer XML-Verfahrensanweisung.
Die Graphik stellt diese als UML-Klassendiagramm dar:



Text

Die Schnittstelle `Text` erweitert `CharacterData`, die die basalen Operationen zur Manipulation von Zeichenketten definiert, geeignet um textuelle Inhalte eines XML-Dokuments darstellen zu können.
Die Graphik stellt diese als UML-Klassendiagramm dar:



splitText(offset : unsigned long) : Text

Das abschließende Beispiel zeigt die Nutzung aller vorgestellten Schnittstellen zur Konstruktion eines neuen XML-Dokuments.
Hierbei wird das in Abbildung 44 verwendete XML-Dokument vollständig durch DOM-Aufrufe im Hauptspeicher erzeugt und anschließend über die Standardausgabe ausgegeben.
Ebenso wie durch den Standard keine lesenden Operationen definiert werden, fehlt auch die symmetrische Möglichkeit zum Schreiben der erstellen Objektstrukturen. Die hierfür notwendigen Operationen werden in konkreten Implementierungen durch proprietären Code umgesetzt. Das vorliegende Beispiel nutzt daher die von SUN für die JDK-Implementierung vorgeschlagene (umständliche) Verfahrensweise der Erzeugung eines `StreamResult`-Objektes als Ergebnis der Dokumenttransformation mittels XSLT.

Beispiel 113: Erzeugung eines XML-Dokuments im Hauptspeicher

```
(1)import org.w3c.dom.Document;
(2)import org.w3c.dom.Element;
(3)import javax.xml.parsers.DocumentBuilder;
(4)import javax.xml.parsers.DocumentBuilderFactory;
(5)import javax.xml.transform.Transformer;
(6)import javax.xml.transform.TransformerFactory;
(7)import javax.xml.transform.dom.DOMSource;
(8)import javax.xml.transform.stream.StreamResult;
(9)
(10)public class DOMExample6 {
(11)    public static void main(String[] args) throws Exception {
(12)        // implementation specific part begins ...
(13)        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
(14)        DocumentBuilder builder = factory.newDocumentBuilder();
(15)        Document myDocument = builder.newDocument();
(16)        // ... ends
(17)
(18)        myDocument.appendChild(myDocument.createProcessingInstruction("myPI", "this is a
test"));
(19)
(20)        Element htmlElement = myDocument.createElementNS("http://www.w3.org/1999/xhtml",
"html");
(21)        htmlElement.setAttribute("xmlns", "http://www.w3.org/1999/xhtml");
(22)        htmlElement.setAttribute("xmlns:svg", "http://www.w3.org/2000/svg");
(23)
(24)        myDocument.appendChild(htmlElement);
(25)
(26)        Element headElement = myDocument.createElementNS("http://www.w3.org/1999/xhtml",
"head");
(27)        htmlElement.appendChild(headElement);
(28)
(29)        Element titleElement = myDocument.createElementNS("http://www.w3.org/1999/xhtml",
"title");
(30)
(31)        titleElement.appendChild(myDocument.createTextNode("Testpage"));
(32)
(33)        headElement.appendChild(titleElement);
(34)
(35)        // this is the usual sequence ...
(36)        Element pElement = (Element) htmlElement.appendChild((myDocument.createElementNS
("http://www.w3.org/1999/xhtml", "p")));
(37)        Element aElement = myDocument.createElementNS("http://www.w3.org/1999/xhtml", "a");
(38)        aElement.setAttribute("href", "http://www.jeckle.de");
(39)        aElement.appendChild(myDocument.createTextNode("link"));
(40)
(41)        pElement.appendChild(myDocument.createTextNode("This is a"));
(42)        pElement.appendChild(aElement);
(43)
(44)        Element svgElement = myDocument.createElementNS("http://www.w3.org/2000/svg", "svg:
svg");
(45)        pElement.appendChild(svgElement);
(46)        svgElement.setAttribute("width", "4cm");
(47)        svgElement.setAttribute("height", "8cm");
(48)
(49)        Element ellipseElement = myDocument.createElementNS("http://www.w3.org/2000/svg", "svg:
ellipse");
(50)        ellipseElement.setAttribute("cx", "2cm");
(51)        ellipseElement.setAttribute("cy", "4cm");
(52)        ellipseElement.setAttribute("rx", "2cm");
(53)        ellipseElement.setAttribute("ry", "1cm");
(54)
(55)        svgElement.appendChild(ellipseElement);
(56)
(57)        // implementation specific part begins ...
(58)        TransformerFactory transFactory = TransformerFactory.newInstance();
(59)        Transformer myTransformer = transFactory.newTransformer();
(60)        DOMSource src = new DOMSource(myDocument);
(61)        myTransformer.transform(src, new StreamResult());
(62)        // ... ends
```

```
(63)    } //main()
(64)} //class DOMExample6
```

Download des Beispiels

Das Beispiel zeigt die unzureichende Berücksichtigung der Namensräume in der aktuellen DOM2-Spezifikation. So werden die Namensraumdeklarationen wie Attribute durch `setAttribute`-Operationen den entsprechenden Elementen hinzufügt. Als Voraussetzung dieser Mimik muß die Implementierung daher gezielt ungültige XML-Bezeichner (Verstoß gegen das Verbot Element- und Attributnamen mit `xml` beginnen zu lassen) gestatten.
Ebenso ist die Übergabe qualifizierter Namen bei der Elementerzeugung (`createElementNS`) im selben Maße fehlerträchtig, da sie ungeprüft die literale Angabe des Namensraumpräfixes erfordert. Undefinierte Präfixe können hierbei zu ungültigen Dokumenten führen.

Anmerkung: Das die Erweiterung der Methode `createElement` um Namespacefunktionalität zu einer Methode neuen Namens (`createElementNS` führt mag den mit Überladungspolymorphie „aufgewachsenen" Anwender objektorientierter Sprachen wie Java oder C++ zunächst verwundern. Hier offenbart sich jedoch nochmals die Unabhängigkeit der DOM-Schnittstelle, die gezielt keine Annahmen über eine technische Umsetzung trifft und daher in diesem Falle die möglicherweise nutzbare Polymorphie zu Gunsten der sprachunabhängiget vernachlässigt.

Ausblick auf DOM Level 3

Die nächste erweiternde Überarbeitung der DOM-Schnittstelle wird u.a. weitere speziell auf XML zugeschnittene Funktionalität bieten.
Darunter Operationen zur Ermittlung der Verwaltungsdaten eines XML-Dokuments, wie Version (`public String getVersion()`), Inhalt der Standalone-Deklaration (`public boolean getStandalone()`) sowie zur Abfrage des verwendeten Zeichenschemas (`public String getEncoding()`).
Diese Spezifikationen befinden sich jedoch noch im Status eines *Working Drafts* und liegen nur unvollständig in verfügbaren Implementierungen vor.

Abschlußbemerkungen und Einsatzempfehlungen

Mit den im W3C Document Object Model versammelten Schnittstellen lassen sich XML-strukturierte Dokumente durch eine generische Schnittstelle vergleichsweise einfach vollständig im Hauptspeicher halten und manipulieren.
Jedoch offenbart sich die Generizität der Schnittstelle allzuoft als Hemmschuh im praktischen Einsatz. Dies liegt einerseits in der entstehenden Lücke zwischen dem DOM-Typsystem und den Applikationsobjekten begründet, die eine explizite Abbildung oder Konvertierung zur Laufzeit erfordern. Darüberhinaus gibt das Objektmodell des DOM Traversierungswege vor, ohne Möglichkeiten zur konformen Erweiterung anzubieten.
Die entstehenden Speicherobjekte sind daher nur bedingt als Datenstrukturen der Applikation geeignet.
Zusätzlich konsumiert der Aufbau der DOM-Instanz mitunter beachtliche Speichermengen, was sich bei wachsender Dokumentgröße durchaus als Laufzeitproblem bemerkbar machen kann. Konkret ist das Skalierungsverhalten des DOM hinsichtlich Speicher und Laufzeit deutlich höher als bei vergleichbaren SAX-Implementierungen.
Generell gilt es im praktischen Einsatz die Abwägung zwischen angestrebtem Modifkationsumfang und Lesegeschwindigkeit anzustellen. Während die Vorteile des DOM klar auf Seiten der Änderbarkeit -- bis hin zur kompletten Erstellung vollständiger XML-Dokumente im Hauptspeicher -- liegt, gewinnt SAX durch sein planbares Laufzeit- und Speicherplatzverhalten.
Der nachfolgende Abschnitt stellt einige Laufzeit- und Speicherplatzuntersuchungen zusammen.

Web-Referenzen 18: Weiterführende Links
DOM-Implementierungen:
- SmallTalk
- JAXP @ SUN.com (Java)
- DOM4J (Java)
- JDOM Java Specification Request (JSR 102)
- JDOM (Java)
- XML4J (Java)
- PyXML (Python)

3.3 Extrahierende Parser

Neben den beiden bisher vorgestellten API- und Parsertypen konnte sich in jüngerer Zeit ein neuer Ansatz zur lesenden Verarbeitung von XML-Dokumenten durch Programmiersprachen etablieren: die sogenannten *extrahierenden Parser* (engl. pull parser).
Ihr Verarbeitungsmodell wurde gezielt im Kontrast zu den bisher vorgestellten Mechanismen entwickelt. Während SAX und DOM XML-Eingaben verarbeiten ohne dabei dem Anwender den Eingriff in den Fluß der Verarbeitung zu gestatten, d.h. der Kontrollfluß wird nicht mehr explizit durch den Code im Programm definiert, sondern wird zur Ausführungszeit durch die Struktur des verarbeiteten XML-Dokuments bestimmt. Im Falle des SAX-Modells spiegelt sich dieser Fluß in der Reihenfolge der durch den Parser aufgerufenen Call-back-Routinen wieder, während im Verlauf der Baumkonstruktion eines DOM-basierten Parsers keinerlei Eingriffsmöglichkeiten für den Programmierer vorgesehen sind.

Extrahierende Parser kehren dies Paradigma um. Sie definieren eine Schnittstelle um innerhalb des Programmflusses aktiv Inhalte eines XML-Dokumentes zu extrahieren. Allerdings erfolgt der Zugriff hierbei nicht wahlfrei wie beispielsweise bei der Ergebniskonstruktion eines XPath- oder XQuery-Ausdruckes, sondern konsekutiv entlang der Reihenfolge der einzelnen Primitive im XML-Dokument.
Dieses Verhalten ist daher weniger mit einer Anfrage, als eher mit dem Vorrücken eines Dateizeigers bei linearer

Konsumption vergleichbar.

Definition 16: Extrahierender Parser

Ein extrahierender Parser gestattet dem Programmierer die konsekutive lesende Verarbeitung der einzelnen Primitive eines XML-Dokuments gemäß der Dokumentordnung.

Gegenwärtig liegen Pull-Parser sowohl für die Java-Sprachwelt vor, als auch eine Implementierung als Bestandteil des Microsoft .NET-Frameworks. Jedoch konnte sich für diese Art der Verarbeitung noch kein allgemein anerkanntes und unterstütztes API herausbilden. Lediglich die *Common API for XML Pull Parsing* (XPP) konnte im Java-Umfeld einige Bedeutung erlangen und ist bereits ein ersten Umsetzungen verfügbar. Dieser Schnittstellenvorschlag bildet auch die Grundlage eines Standardisierungsansatzes (JSR 173) im Rahmen des *Java Community Processes*.
Das vorliegende Kapitel stützt sich auf eine Implementierung dieser API die unter dem Namen *XPP3/MXP1* kostenfrei verfügbar ist.

Das XPP-API definiert zur Verwaltung mindestens die folgenden Schnittstellen:

- **`XmlPullParserFactory`**: Fabrikklasse zur Erzeugung eines Parsers
- **`XmlPullParser`**: Abstrakte Klasse die den tatsächlichen Parser repräsentiert, alle konkret instantiierbaren Parser erben von dieser Klasse
- **`XmlPullException`**: Generische Ausnahmeklasse zur Signalisierung verschiedenster Fehler

Bereits diese Übersicht zeigt als einen ersten wesentlichen Unterschied zu den klassischen Parserschnittstellen, daß innerhalb der XPP-API Wert auf die standardisierung der Infrastruktur zur Erzeugung des Parsers sowie die Darstellung auftretender Fehler gelegt wurde.

Zur Extraktion der Inhalte eines XML-Dokuments sieht die XPP-Schnittstelle folgende durch Instanzen der Klasse `XmlPullParser` umgesetzte Methoden vor:

- **`getEventType`**: liefert Aufschluß darüber welche XML-Primitive (`START_TAG`, `END_TAG`, `CONTENT` ...) extrahiert wurde
- **`getLocalName`**: liefert für Ereignisse des Typs `START_TAG` und `END_TAG` den lokalen Namen des extrahierten Elements
- **`getNamespaceUri`**: liefert die URI des Namensraumes dem das extrahierte Element zugeordnet ist
- **`getPrefix`**: liefert -- sofern vorhanden -- das Namensraum-Präfix des extrahierten Elements
- **`readContent`**: liefert den textuellen Inhalt des extrahierten Elements
- **`getAttributeCount`**: liefert für Ereignisse des Typs `START_TAG` die Anzahl der für das extrahierte Element definierten Attribute
- **`getAttributeLocalName(int i)`**: liefert den lokalen Namen des *i*-ten Attributs für Ereignisse des Typs `START_TAG`
- **`getAttributeNamespaceUri(int i)`**: liefert die URI des Namensraumes dem das *i*-te Attribut zugeordnet ist
- **`getAttributePrefix(int i)`**: liefert -- sofern vorhanden -- das Namensraum-Präfix des *i*-ten Attributs
- **`getAttributeType(int i)`**: liefert -- sofern durch Zugriff auf eine DTD oder ein XML-Schema möglich -- den Datentyp des *i*-ten Attributs
- **`getAttributeValue(int i)`**: liefert den Wert des *i*-ten Attributs
- **`getAttributeValue(String namespace, String localName)`**: liefert den Wert des durch Namen und Namensraum bezeichneten Attributs

Beispiel 114: Eine einfache XPP-basierte Applikation

```
(1)import org.gjt.xpp.XmlPullParser;
(2)import org.gjt.xpp.XmlPullParserException;
(3)import org.gjt.xpp.XmlPullParserFactory;
(4)import java.io.FileNotFoundException;
(5)import java.io.FileReader;
(6)import java.io.IOException;
(7)
(8)public class XPPsimple {
(9)    public static void main(String argv[]) {
(10)        try {
(11)            XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
(12)            factory.setNamespaceAware(true);
(13)            XmlPullParser xpp = factory.newPullParser();
(14)            xpp.setInput(new FileReader(argv[0]));
(15)
(16)            int eventType;
(17)            while ((eventType = xpp.next()) != xpp.END_DOCUMENT) {
(18)                if (eventType == xpp.START_TAG)
(19)                    System.out.println("element started");
(20)                if (eventType == xpp.END_TAG)
(21)                    System.out.println("element ended");
(22)            } //while
(23)        } catch (XmlPullParserException e) {
(24)            e.printStackTrace();
(25)        } catch (FileNotFoundException e) {
(26)            e.printStackTrace();
(27)        } catch (IOException e) {
(28)            e.printStackTrace();
(29)        } //try
(30)    } //main()
(31)} //class XPPsimple
```

Download des Beispiels

Der Code des Beispiels 114 zeigt die Nutzung der XPP-API.

Zunächst wird in Zeile 11 eine Parser-Fabrik zur späteren Erzeugung des tatsächlichen Parsers erzeugt. Diese wird durch Aufruf der Methode setNamespaceAware dahingehend parametrisiert, daß die von ihr zur Verfügung gestellten Parser in einem Modus konform zur Namensraumspezifikation arbeiten.
In Zeile 13 wird von der Fabrik eine neue Parserinstanz angefordert deren Eingabekanal auf die als Kommandozeilenparameter übergebene Datei gelenkt wird (Zeile 14).
Die Extraktion der einzelnen Bestandteile des Eingabedokuments vollzieht sich in der Schleife ab Zeile 17. Dort wird, solange nicht das Ereignis END_DOCUMENT gelesen wurde, mittels Aufruf der Methode next ein weiteres Ereignis aus dem XML-Eingabestrom angefordert.
Nach Feststellung des Ereignistyps (Zeilen 18 und 20) kann eine Verarbeitung der mit dem Ereignis verknüpften XML-Primitive erfolgen.

Einschließlich der im Beispiel verwendeten Ereignistypen unterstützen XPP-konforme Parser folgende Ereignistypen:

- **START_DOCUMENT**: Beginn eines Dokuments bevor XML-Anteile durch den Parser verarbeitet wurden.
  Dieses Ereignis wird nicht durch alle Implementierungen zur Verfügung gestellt, da es implizit für den gesamten programmiersprachlichen Quellcode vor dem ersten Aufruf des XPP-Parsers gilt.
- **CDSECT**: Markiert das Auftreten einer CDATA-Sektion innerhalb des XML-Dokumentes.
- **COMMENT**: Markiert das Auftreten eines XML-Kommentars innerhalb des Eingabedokumentes.
- **START_TAG**: Markiert durch Extraktion des Start-Tags den Beginn eines Elements.
- **END_TAG**: Markiert durch Extraktion des End-Tags den Abschluß eines Elements.
  Für leere Elemente wird dieses Ereignis, unabhängig von der möglicherweise genutzten Minimierungssyntax, direkt nach dem zugehörigen START_TAG-Ereignis extrahiert.
- **ENTITY_REF**: Markiert das Auftreten einer Entitätsreferenz.
- **IGNORABLE_WHITESPACE**: Markiert das Auftreten von Leerraumsymbolen, die durch den Parser ohne Informationsverlust überlesen werden können.
- **PROCESSING_INSTRUCTION**: Markiert das Auftreten einer Processing Instruction.
- **TEXT**: Markiert das Auftreten freien Textes als Elementinhalt.
- **END_DOCUMENT**: Markiert das Ende des Dokuments.
  Nach diesem Ereignis können keine Weiteren extrahiert werden.

Der Code des Beispiels 115 setzt eine Anwendung zur Zählung des Auftretens der einzelnen XML-Primitive auf der Basis eines extrahierenden Parsers um. Damit stellt das Programm eine Re-Implementierung der in Beispiel 98 auf Basis des SAX-APIs gezeigten Funktionalität dar.

Beispiel 115: Zählung der einzelnen XML-Primitive

```
(1)import org.gjt.xpp.XmlPullParser;
(2)import org.gjt.xpp.XmlPullParserException;
(3)import org.gjt.xpp.XmlPullParserFactory;
(4)import java.io.FileReader;
(5)import java.io.IOException;
(6)
(7)public class XPPsample3 {
(8)    public static void main(String argv[]) {
(9)        try {
(10)            XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
(11)            factory.setNamespaceAware(true);
(12)            XmlPullParser xpp = factory.newPullParser();
(13)            xpp.setInput(new FileReader(argv[0]));
(14)            int eventType;
(15)            int startDocument = 1;
(16)            int endDocument = 0;
(17)            int startTag = 0;
(18)            int endTag = 0;
(19)            int elementContent = 0;
(20)            do {
(21)                eventType = xpp.next();
(22)                if (eventType == xpp.END_DOCUMENT) {
(23)                    endDocument++;
(24)                } else if (eventType == xpp.START_TAG) {
(25)                    startTag++;
(26)                } else if (eventType == xpp.END_TAG) {
(27)                    endTag++;
(28)                } else if (eventType == xpp.CONTENT) {
(29)                    elementContent++;
(30)                } //if
(31)            } while (eventType != xpp.END_DOCUMENT);
(32)            System.out.println("# startDocument events:" + startDocument + "\n" +
(33)                    "# endDocument events:" + endDocument + "\n" +
(34)                    "# startTag events:" + startTag + "\n" +
(35)                    "# endTag events:" + endTag + "\n" +
(36)                    "# element content events:" + elementContent + "\n"
(37)            );
(38)
(39)        } catch (XmlPullParserException e) {
(40)            System.out.println("EXCEPTION CAUGHT: " + e.getClass().getName());
(41)            e.printStackTrace();
(42)        } catch (IOException e) {
(43)            System.out.println("EXCEPTION CAUGHT: " + e.getClass().getName());
(44)            e.printStackTrace();
(45)        } //try
(46)    } //main()
(47)} //class XPPsample3
```

Download des Beispiels

Insbesondere im Vergleich zum ereignisbasierten Lesen unter Nutzung der SAX-Schnittstelle fällt die veränderte Umsetzung auf. So muß der Gesamtablauf nun nicht mehr in verschiedene Methoden aufgespalten werden, die durch den Parser aufgerufen werden, sondern kann innerhalb einer Methode realisiert werden.
Aus diesem Grunde kann auch die konzeptionell suboptimale Verwendung globaler Variablen unterbleiben.
Im Code fällt die Initialisierung der Zählvariable `startDocument` mit 1 auf (Zeile 15). Diese wird notwendig, da die verwendete Parserimplementierung keine Extraktion von `START_DOCUMENT` leistet weil dieses Ereignis immer implizit vor dem Start des Extraktionsvorganges anliegt.

Beispiel 116: Häufigkeitsermittlung einzelner Elementnamen

```
(1)import org.gjt.xpp.XmlPullParserFactory;
(2)import org.gjt.xpp.XmlPullParser;
(3)import org.gjt.xpp.XmlPullParserException;
(4)import java.util.HashMap;
(5)import java.io.FileReader;
(6)import java.io.FileNotFoundException;
(7)import java.io.IOException;
(8)
(9)public class XPPsample2 {
(10)    private static final Integer ONE = new Integer(1);
(11)
(12)    public static void main(String argv[]) {
(13)        try {
(14)            HashMap elementHM = new HashMap();
(15)            XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
(16)            factory.setNamespaceAware(true);
(17)            XmlPullParser xpp = factory.newPullParser();
(18)            xpp.setInput(new FileReader(argv[0]));
(19)
(20)            int eventType;
(21)            Integer freq;
(22)            String qName;
(23)            while ((eventType = xpp.next()) != xpp.END_DOCUMENT) {
(24)                if (eventType == xpp.START_TAG) {
(25)                    qName = xpp.getNamespaceUri()+":"+xpp.getLocalName();
(26)                    freq = (Integer) elementHM.get(qName);
(27)                    elementHM.put(qName, (freq == null ? ONE : new Integer(freq.intValue() +
1)));
(28)                } //if
(29)            } //while
(30)
(31)            System.out.println("RESULT=" + elementHM);
(32)
(33)        } catch (XmlPullParserException e) {
(34)            e.printStackTrace();
(35)        } catch (FileNotFoundException e) {
(36)            e.printStackTrace();
(37)        } catch (IOException e) {
(38)            e.printStackTrace();
(39)        } //try
(40)    } //main()
(41)} //class XPPsample2
```

Download des Beispiels

Der Quellcode des Beispiels 116 entspricht funktional dem des Beispiels 99, er zählt die Auftretenshäufigkeit der Elemente in einem XML-Dokument.
Jedoch unterstreicht er -- insbesondere im Vergleich zum SAX-basierten Beispiel -- die Natur der XPP-Schnittstelle.
Im vorliegenden Falle werden durch den Parser keine Methodenaufrufe durchgeführt, sondern die Extraktion und Verarbeitung der einzelnen Elemente des betrachteten XML-Dokuments obliegt ausschließlich der Verantwortung des Programmierers.

Fehlerbehandlung

Ausgehend von den vorstehend beschriebenen Ereignistypen läßt sich leicht eine Prüfung auf Wohlgeformtheit realisieren, welche typischerweise durch alle verfügbaren XPP-Implementierungen durchgeführt wird.
Allerdings können Verletzungen der grundlegenden XML-Strukturierungsregeln auch diesem Parsingmodell, wie auch für SAX, nicht während eines nach außen abgeschlossenen Aufrufs erkannt werden, sondern offenbaren sich erst im Verlaufe des Extraktionsvorganges.

Beispiel 117: Behandlung einer XmlPullParserException

```
(1)import org.gjt.xpp.XmlPullParser;
(2)import org.gjt.xpp.XmlPullParserException;
(3)import org.gjt.xpp.XmlPullParserFactory;
(4)import java.io.FileReader;
(5)import java.io.IOException;
(6)
(7)public class XPPsample4 {
(8)    public static void main(String argv[]) {
(9)        XmlPullParser xpp = null;
(10)        try {
(11)            XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
(12)            factory.setNamespaceAware(true);
(13)            xpp = factory.newPullParser();
(14)            xpp.setInput(new FileReader(argv[0]));
(15)            int eventType;
(16)            while ((eventType = xpp.next()) != xpp.END_DOCUMENT) {
(17)            } //while
(18)        } catch (XmlPullParserException e) {
(19)            System.out.println("EXCEPTION CAUGHT: " + e.getClass().getName());
(20)            System.out.println("Error occured at line: " + xpp.getLineNumber() + " at column "
+ xpp.getColumnNumber());
(21)            e.printStackTrace();
(22)        } catch (IOException e) {
(23)            System.out.println("EXCEPTION CAUGHT: " + e.getClass().getName());
(24)            e.printStackTrace();
(25)        } //try
(26)    } //main()
(27)} //class XPPsample4
```

Download des Beispiels
Download der Ergebnisdatei

Führt man den Code des Beispiels 117 mit dem Eingabedokument des Beispiels 10 aus, erzeugt der Parser folgende Ausnahme:

```
(1)EXCEPTION CAUGHT: org.gjt.xpp.impl.tokenizer.TokenizerException
(2)Error occured at line: 3 at column 17
(3)org.gjt.xpp.impl.tokenizer.TokenizerException: attribute value must start with double quote or
apostrophe not 'a' at line 3 and column 17 seen "...<root>\r\n\t<elementA att"...
(4)        at org.gjt.xpp.impl.tokenizer.Tokenizer.next(Tokenizer.java:1156)
(5)        at org.gjt.xpp.impl.pullparser.PullParser.next(PullParser.java:392)
(6)        at XPPsample4.main(XPPsample4.java:23)
```

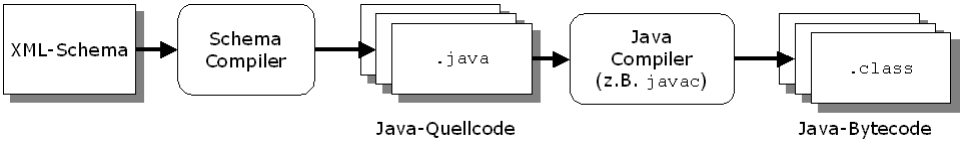
Web-Referenzen 19: Weiterführende Links
•XML Pull Parsing -- Übersicht der API sowie Verweis auf verschiedene Implementierungen
•N. Bornstein: *Pull Parsing in C# and Java*
•Vergleich zwischen SAX2- und XPP-basierten Parsern

3.4 Nahtlose Integration von XML und Hochsprache -- Java XML Data Binding

Einen neuartigen Ansatz zur Integration zwischen XML-basierter Datenbeschreibung und applikationsinternen Objektstrukturen stellen die sog. *data bindings* für XML dar. Sie stellen anders als die Schnittstellen DOM und SAX keine Sammlung abstrakter Zugriffsroutinen dar, sondern leiten aus einer XML-Grammatik vokabularspezifische Speicher- und Verarbeitungsstrukturen ab. Zur Erreichung dieses Ziels werden aus einer als DTD oder XML-Schema vorliegenden Grammatik programmiersprachliche Konstrukte, etwa Klassendefinitionen und darauf operierende Methoden, erzeugt, die in eigene Applikationen eingebunden werden können.
Damit ist dieses datenorientierte Vorgehen deutlich abgrenzbar gegenüber ausführungsgetriebenen generischen Schnittstellen.

Mit der *XML Data Binding Specification* für Java liegt ein erster Ansatz zur Standardisierung vor, der sich sowohl der Ableitung der datenspeichernden Klassenstrukturen als auch den darauf operierenden Methoden widmet und für diese Referenzimplementierungen vorgibt. Die so erzeugten Datenstrukturen gestatten die direkte Verarbeitung der in einem schemakonformen Dokument abgelegten Daten in einer Java-Applikation.
Aktuell sind neben der Beschreibung des Spezifikationsvorhabens bereits erste Implementierungen verfügbar.
Die nachfolgenden Aussagen beziehen sich auf die frei zugängliche (Open Source) Implementierung *Castor* welche bereits große Teile der angestrebten Mächtigkeit abdeckt.

Abbildung 53 zeigt die einzelnen Schritte zur Erstellung der Java-Quelldateien aus einem XML-Schema.
Hierbei werden durch eine *Schema-Compiler* genannte Softwarekomponente aus dem XML-Schema zunächst Java-Klassen im Quellcode erzeugt. Ausgehend von diesem generierten Code kann die eigene Implementierung aufsetzen, die gemäß dem klassischen Java-Build-Compile-Zyklus verläuft.

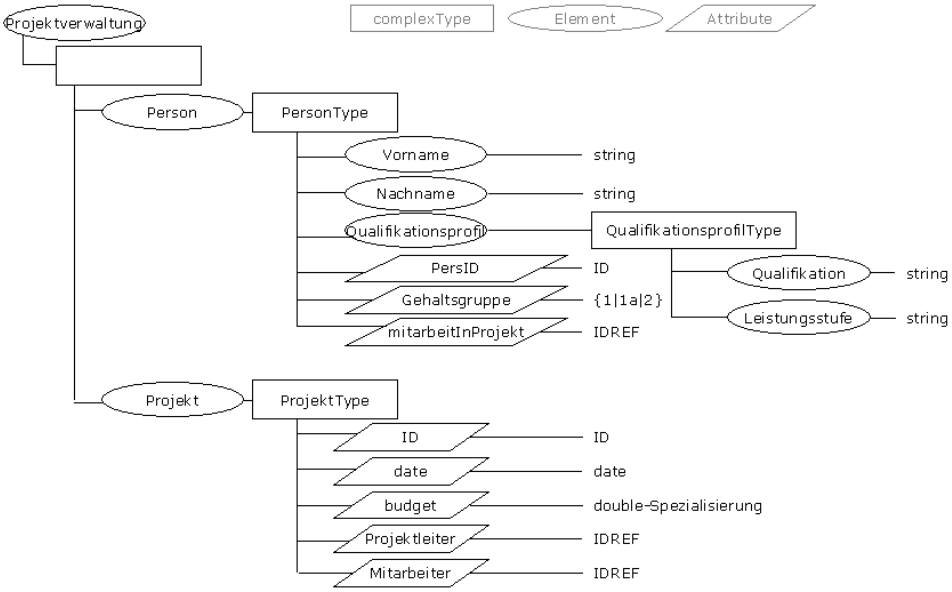


Zusätzlich zu den statischen Klassenstrukturen erzeugt der Schema-Compiler einige Methoden, die zum lesenden und schreibenden Zugriff auf die XML-Eingangsdaten benötigt werden. Dazu zählen Methoden zur Abfrage existierender Elemente und Attribute ebenso wie Codeteile, die das Erzeugen neuer Anteile oder das Entfernen bestehender gestatten.
Zusätzlich werden mit den Methoden `marshal` bzw. `unmarshal` Routinen zur Verfügung gestellt, die ein gegebenes schemakonformes XML-Dokument in den Hauptspeicher abbilden, bzw. das Ausschreiben der speicherresidenten Objekte als XML-Datei umsetzen.

Im einzelnen geschieht die Abbildung der XML-Schemastrukturen auf Java-Klassen wie folgt:

- **Elemente**: Alle Elemente, für die eine `complexType`-Definition existiert, werden in eine Klasse überführt.
  Elemente eines simplen Typs werden nicht in Klassen überführt.
- **Komplexe Typen**, die auf der obersten Schemaebene definiert wurden, werden in eine eigenständige abstrakte Java-Klasse abgebildet.
  Klassen, die aus einer Elementedefinitionen resultieren, die auf der der obersten Schemaebene angesiedelt ist und die durch eine solche `complexType`-Definition typisiert werden, erben von der aus dem komplexen Typ erzeugten Klasse.
- Unbenannte komplexe Typen werden durch separate „Typklassen“ ausgedrückt.
- **XML-Attribute** werden in Attribute der erzeugten Java-Klassen überführt.

Nachfolgend wird die zuvor abstrakt dargestellte Generierung am Beispiel des Projektverwaltungsschemas diskutiert.



Die Abbildung 54 stellt vereinfacht den strukturellen Aufbau des Projektverwaltungsschemas dar. Dabei wurden alle Elemente, Attribute sowie die Definitionen komplexer und simpler Typen berücksichtigt. Syntaktische Besonderheiten ohne Beitrag zur Struktur, wie die Unterscheidung zwischen direkt eingebetteter Definition und Referenzierung, wurden nicht berücksichtigt.

Der Schema-Compiler erzeugt daher aus dem Projektverwaltungsschema folgende Java-Quellcode-Dateien (JavaDoc-Dokumentation der erzeugten Klassen):

Aus den Elementen:

- `Projektverwaltung.java`
- `Person.java`
- `Projekt.java`
- `Qualifikationsprofil.java`

Aus den komplexen Typen:

- `PersonType.java`
- `ProjektType.java`
- `QualifikationsprofilType.java`

Zusätzlich wird für den Inhalt des Elements `Qualifikationsprofil` die Klasse `QualifikationsprofilTypeItem.java` erzeugt. Sie resultiert aus dem im XML-Schema definierten Auswahlinhaltsmodell, das entweder ein Element des Typs `Qualifikation` oder `Leistungsstufe` zuläßt.

Die im XML-Schema verwendeten Datentypen werden durch den Generierungsprozeß auf die verfügbaren Java-internen Datentypen abgebildet, soweit sinnvoll möglich. Für alle anderen XSD-Typen liefert die Castor-Laufzeitbibliothek eigene Implementierungen.
Die Abbildungen der XML-Schematypen auf die entsprechenden Programmierspachenkonstrukte kann Tabelle 28 entnommen werden.

Tabelle 28: Abbildung der XML-Schemadatentypen auf die der Castor-Laufzeitbibliothek

XML-Schemadatentyp	Java-Castor-Typ
string	java.lang.String
boolean	boolean (Java-Primitivtyp)
decimal	java.math.BigInteger
float	float (Java-Primitivtyp)
double	double (Java-Primitivtyp)
dateTime	java.util.Date
time	org.exolab.castor.types.Time
gYearMonth	org.exolab.castor.types.GYearMonth
gYear	org.exolab.castor.types.GYear
gMonthDay	org.exolab.castor.types.GMonthDay
gDay	org.exolab.castor.types.GDay
gMonth	org.exolab.castor.types.GMonth
hexBinary	byte[] (Array des Java-Primitivtypen)
base64Binary	byte[] (Array des Java-Primitivtypen)
anyURI	java.lang.String
QName	java.lang.String
normalizedString	java.lang.String
NMTOKEN	java.lang.String
NMTOKENS	java.util.Vecotor mit Elementen des Typs NMTOKEN
NCName	java.lang.String
ID	java.lang.String
IDREF	java.lang.Object
IDREFS	java.util.Vector mit Elementen des Typs IDREF
integer	int
nonPositiveInteger	int
negativeInteger	int
long	long
int	int
short	int
byte	byte
nonNegativeInteger	int
positiveInteger	int

Darüberhinaus erzeugt der Schema-Compiler aus der Typdefinition der Gehaltsgruppe eine separate Klasse zur Behandlung dieses selbstdefinierten Datentyps.

Die Applikation des Beispiels 118 zeigt die Implementierung des Einlesevorganges mit den durch Castor generierten Rahmenklassen.
Der gesamte Lese- und Validierungsvogang wird durch die Methode unmarshal realisiert. Sie liest eine XML-Datei aus einem Eingabestrom und erzeugt dabei die notwendigen Speicherobjekte und belegt sie mit der eingelesenen Information.
Tritt während dieses Vorganges ein Fehler auf, so wird eine MarshalException ausgelöst. Diese Situation entspricht einem Validierungsfehler beim Prüfvorgang des zugrundeliegenden XML-Schemas.
Dual zum Lesevorgang erlaubt die Methode marshal das Ausschreiben der Speicherobjekte als XML-Datei. Hierbei ist es unerheblich, ob diese Hauptspeicherstrukturen durch einen vorhergehenden unmarshal-Aufruf erzeugt wurden, oder mit programmiersprachlichen Mitteln innerhalb der Applikation instanziiert wurden.

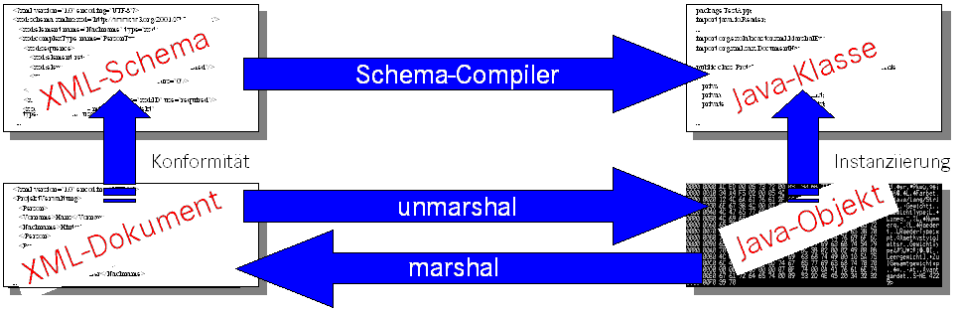
Beispiel 118: Einlesen einer XML-Datei mit Castor

```
(1)import java.io.*;
(2)import org.exolab.castor.xml.MarshalException;
(3)import TestApp.*;
(4)
(5)public class CastorExample1 {
(6)    public static void main(String[] argv) {
(7)        try {
(8)            ProjektVerwaltung myPv = ProjektVerwaltung.unmarshal(new FileReader(argv[0]));
(9)            //...
(10)            myPv.marshal(new FileWriter(argv[1]));
(11)        } //try
(12)        catch (MarshalException e) {
(13)            Exception originatingException = e.getException();
(14)            if (originatingException != null) {
(15)                System.out.println(originatingException);
(16)            } //if
(17)        } //catch
(18)        catch (Exception e) {
(19)            System.out.println("EXCEPTION: " + e);
```

```
(20)        }
(21)    } //main()
(22)} //class CastorExample1
```

Download des Beispiels

Die Graphik 55 faßt nochmals die beiden Einzelschritte des Quellcodes aus Beispiel 118 und den Klassengenerierungsschritt zusammen.



Zur Verarbeitung der speicherresidenten Javaobjekte werden durch den Schema-Compiler eine Reihe Standardmethoden erzeugt, welche den einfachen und uniformen Zugriff auf die verschiedenen Informationen ermöglichen.
Im Einzelnen sind dies:

Tabelle 29: Leseoperationen des Castor Data Bindings

Operation	Beispiel	Funktionalität
`get...()`	`getBudget()`	Liefert den Inhalt des Attributs oder Elements. Der Ergebnisdatentyp entspricht hierbei dem Datentyp gemäß der Abbildungsvorschriften aus Tabelle 28
`getContent()`	`getContent()`	Liefert den textartigen Inhalt eines Knotens. Diese Form findet ausschließlich für Elemente mit gemischtem Inhalt Anwendung.
`get...Count()`	`getPersonCount()`	Liefert die Anzahl der Kindknoten eines gegebenen Typs.
`get...(int index)`	`getMitarbeiter(1)`	Liefert das *n*-te Element des entsprechenden Typs zurück. Im XML-Dokument entspricht dies dem *n*-ten Kindknoten.
`enumerate...()`	`enumeratePerson()`	Liefert alle Kindknoten des gegebenen Typs als Java-Collection.
`has...()`	`hasBudget()`	Liefert für Attribute mit Vorgabewertdefinition die Information, ob der Wert aus dem Dokument übernommen wurde oder anhand der im XML-Schema definierten Vorgabebelegung gesetzt wurde.
`isValid()`	`isValid()`	Liefert als Wahrheitswert, ob das Objekt einschließlich seiner referenzierten Objekte in ein Schema-gültiges XML-Dokument abgebildet werden kann.

Beispiel 119 zeigt die verschiedenen vorgestellten Leseoperationen.
Zunächst wird mittels der Methode `getPersonCount()` die Anzahl der Kindknoten des Typs `Person` unterhalb des Wurzelknotens `Projektverwaltung` ermittelt. Anschließend werden alle Knoten des genannten Typs einzeln mittels der Methode `getPerson(...)` nach dem Inhalt des Attributs `PersID` angefragt.
Der Zugriff auf die Elemente des Typs `Vorname` vollzieht sich in analoger Weise. Aufgrund des möglichen mehrfachen Auftretens der `Vornamen`-Elemente innerhalb eines `Personen`-Knotens erfolgt hier eine zweite Iteration innerhalb der Schleife über alle Personen.
Abschließend zeigt der Zugriff auf die Elemente des Typs `Qualifikationsprofil` die Verwendung der Java-Collection-API, welche ein vereinfachtes Durchlaufen aller verwalteten Elemente gestattet. Zusätzlich veranschaulicht die letzte Sequenz den Lesevorgang for ein Element mit gemischtem Inhaltsmodell.

Beispiel 119: Auslesen von Attribut- und Elementinformation

```
(1)import TestApp.*;
(2)import TestApp.types.*;
(3)import org.exolab.castor.xml.MarshalException;
(4)import java.io.FileReader;
(5)import java.util.Enumeration;
(6)
(7)public class CastorExample2 {
(8)    public static void main(String[] argv) {
(9)        try {
(10)            ProjektVerwaltung myPv = ProjektVerwaltung.unmarshal(new FileReader(argv[0]));
(11)
(12)            //attribute
(13)            for (int i = 0; i < myPv.getPersonCount(); i++)
(14)                System.out.println(myPv.getPerson(i).getPersID());
(15)
(16)            //element
(17)            Person pers;
(18)            for (int i = 0; i < myPv.getPersonCount(); i++) {
(19)                pers = myPv.getPerson(i);
(20)                for (int j = 0; j < pers.getVornameCount(); j++)
```

```
(21)                    System.out.println(pers.getVorname(j));
(22)            } //for
(23)            pers = null;
(24)
(25)            //mixed content
(26)            System.out.println(myPv.getPerson(1).getQualifikationsprofil().getContent());
(27)
(28)            Enumeration e = myPv.getPerson(1).getQualifikationsprofil().
enumerateQualifikationsprofilTypeItem();
(29)            Object qti;
(30)            while (e.hasMoreElements()) {
(31)                qti = e.nextElement();
(32)                System.out.println(((QualifikationsprofilTypeItem) qti).getLeistungsstufe());
(33)                System.out.println(((QualifikationsprofilTypeItem) qti).getQualifikation());
(34)            } //while
(35)        } //try
(36)        catch (MarshalException e) {
(37)            Exception originatingException = e.getException();
(38)            if (originatingException != null) {
(39)                System.out.println(originatingException);
(40)            } //if
(41)        } //catch
(42)        catch (Exception e) {
(43)            System.out.println("EXCEPTION: " + e);
(44)        }
(45)    } //main()
(46)} //class CastorExample2
```

Download des Beispiels

Tabelle 30: Schreiboperationen des Castor Data Bindings

Operation	Beispiel	Funktionalität
set...(...)	setBudget(100.00)	Setzt den Inhalt des Attributs. Der Übergabedatentyp entspricht hierbei dem Datentyp gemäß der Abbildungsvorschriften aus Tabelle 28
add...(...)	addMitarbeiter (aMitarbeiter)	Fügt ein neues Element hinzu.
set...(int index, ...)	setPerson(1, aPerson)	Fügt ein neues Element an der gegebenen Indexposition ein.
setContent(s java. lang.String)	setContent()	Setzt den textartigen Inhalt eines Knotens. Diese Form findet ausschließlich für Elemente mit gemischtem Inhalt Anwendung.
remove...(...)	removePerson(rPerson)	Löscht das übergebene Element.
clear...()	clearPeson()	Leert das angegebene Element.

Der Code des Beispiels 120 zeigt die verschiedenen Operationen zur Erzeugung von Attribut- und Elementinhalten. Als Besonderheit wird zur Belegung des Attributs Gehaltsgruppe auf eine durch das Rahmenwerk erzeugte Konstante zurückgegriffen, welche die typkonforme Belegung dieses Attributs sicherstellt.

Beispiel 120: Erzeugung von Attribut- und Elementinhalten

```
(1)import TestApp.*;
(2)import TestApp.types.*;
(3)import org.exolab.castor.xml.MarshalException;
(4)import java.io.FileReader;
(5)import java.io.FileWriter;
(6)
(7)public class CastorExample3 {
(8)    public static void main(String[] argv) {
(9)        try {
(10)            ProjektVerwaltung myPv = ProjektVerwaltung.unmarshal(new FileReader(argv[0]));
(11)
(12)            //set existing attribute
(13)            myPv.getPerson(1).setPersID("ID001");
(14)
(15)            //set existing element
(16)            myPv.getPerson(0).setNachname(0, "Schulze");
(17)
(18)            //create attribute
(19)            myPv.getPerson(1).setGehaltsgruppe(GehaltsgruppeType.VALUE_1);
(20)
(21)            myPv.marshal(new FileWriter(argv[1]));
(22)        } //try
(23)        catch (MarshalException e) {
(24)            Exception originatingException = e.getException();
(25)            if (originatingException != null) {
(26)                System.out.println(originatingException);
(27)            } //if
```

```
(28)        } //catch
(29)        catch (Exception e) {
(30)            System.out.println("EXCEPTION: " + e);
(31)        }
(32)    } //main()
(33)} //class CastorExample3
```

Download des Beispiels

Abschlußbemerkungen und Einsatzempfehlungen

Konzeptionell entwickelt der Data Binding-Ansatz die bereits mit dem *Document Object Model* verwirklichte Grundidee einer Speicherabbildung von XML-Dokumenten fort. Anders als im DOM sind jedoch auch die Schnittstellen zum Einlesen und Schreiben der XML-Dateien definiert.
Jedoch stellt der vorgestellte Ansatz als Teil des *Java Community Process* keinen offenen Standard dar, sondern lediglich einen im SUN-Umfeld verfolgten Vorschlag, der bisher nur für die Programmiersprache Java Umsetzung gefunden hat.
In der Anwendung offenbart sich der Ansatz zur Generierung problemspezifischer Klassenstrukturen aus XML-Schemata als durchaus gangbar. Zusätzlich positiv wirken sich die erzeugten Objekte in der Handhabung aus, da aufwendige Typkonversionen (wie bei den generischen DOM-Typen in der Regel immer notwendig) zwischen Serialisierungsformat und Applikationsdatenhaltung entfällt.

Web-Referenzen 20: Weiterführende Links

•Castor
•JAXB @ SUN
•JSR 31
Ähnliche Ansätze:
•Zeus
•Schema2Java
•Übersicht von Ronald Bourret
•Dare Obasanjo: *A Survey of APIs and Techniques for Processing XML*

3.5 Web Services

Motivation und Hintergrund

Stand Datenaustausch in der Betrachtung der Metasprache XML bisher nahezu ausschließlich im Zeichen datei- oder strombasierter Integration, so eröffnet das Einsatzgebiet der *Web Services* den Einsatz XML-basierter Sprachen zur Abwicklung Internet-basierter online Kommunikation zwischen Anwendungssystemen.

Dem Begriff *Web Service* war in jüngerer Zeit eine bemerkenswerte Karriere beschieden, so daß es ihm gelang in der Praxis beachtliches Interesse auf sich zu ziehen. Allerdings fehlt bisher eine einheitliche Definition dieses Terminus hinsichtlich Zielsetzung und technischer Inhalte. Vielmehr versuchten und versuchen namhafte Herstelle durch eine Reihe verschiedener Definitionen, die am Markt offen zueinander in Konkurrenz treten, bisherige Techniken mit dem Ansatz der Web Services zu verschmelzen.
Jedoch führt(e) diese Vorgehensweise weder zu einer einheitlichen Begriffsübereinkunft und daher in der Folge auch zu keiner brauchbaren Abgrenzung gegenüber existierenden Techniken --- wie CORBA, RMI oder DCOM --- im Umfeld.
Nachfolgend wird daher eine an die Ergebnisse der Web Service Architecture-Arbeitsgruppe des World Wide Web Konsortiums angelehnte Definition zugrunde gelegt:

Definition 17: Web Service

Ein Web Service ist ein durch eine URI (gemäß RFC 2396) identifiziertes Softwaresystem, dessen öffentliche Schnittstellen und Protokollbindungen durch XML definiert und beschrieben sind.
Diese Definitionen können durch andere Softwaresysteme ermittelt und bezogen werden. Auf der Basis der publizierten Schnittstellendefinitionen können diese Systeme dann mit dem Web Service in der durch die Schnittstelle festgelegten Weise interagieren. Diese Interaktion geschieht durch den Austausch XML-basierter Nachrichten, die mittels Internetprotokollen übertragen werden.

Der Begriff des *Service* ist hierbei durchaus in Anlehnung an den klassischen Dienstleistungsbegriff gemeint. Dieser beschreibt Handlungen, die im Auftrag eines Dritten vorgenommen werden, welche kein physisches Gut produzieren, sondern deren Charakter ausschließlich durch die Handlungserbringung selbst definiert ist.

Die Definition enthält ferner bereits Aussagen über die Komponenten einer Web Service basierten Architektur (einer sog. *serviceorientierten Architektur* (SOA)) und deren technischer Realisierung. Grundlegend Eigenschaften einer solchen Architektur ist neben der XML-Basiertheit die Verwendung von Universal Resource Identifkatoren zur eineindeutigen Benennung eines angebotenen Dienstes sowie die Nutzung der bestehenden Internetinfrastruktur zur Datenübertragung. Der Begriff *Internetinfrastruktur* soll hierbei nicht verengt auf die Techniken des WWW, d.h. HTTP auf Basis TCP/IP, verstanden werden, sondern durchaus im Sinne des der Internetprotokollhierarchie gebraucht werden.

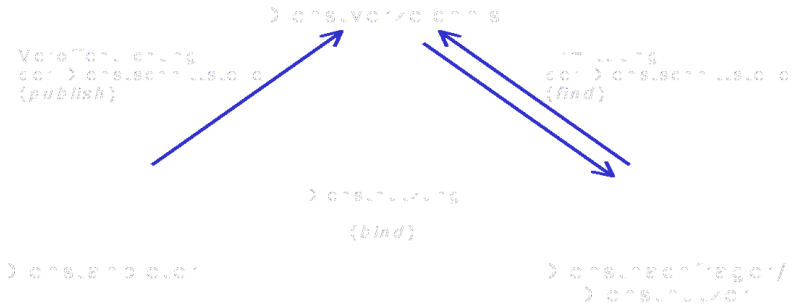
Gleichzeitig postuliert die Definition drei grundlegende Elemente einer serviceorientierten Architektur:

1. Ein XML-basierte Protokoll zur Darstellung und zum Austausch der Daten zwischen Diensten und ihren Nutzern.
2. Eine XML-basierte Beschreibungssprache zur Publikation von Dienstschnittstellen.
3. Einen Dienst zur Verwaltung der publizierten Schnittstellenbeschreibungen.

Diese drei aufeinander aufsetzenden Dienstschichten deuten auch bereits drei typische Rollen innerhalb einer serviceorientierten Architektur an:

1. Dienstanbieter.
   Er übernimmt die physische Bereitstellung des Dienstes, d.h. eine Implementierung und Publikation auf einem über das Internet zugänglichen Server.
   Zusätzlich kann er die Aufrufschnittstelle des Dienstes dokumentieren.
2. Dienstnachfrager/-nutzer.
   Er nutzt den angebotenen Dienst durch Übermittlung XML-codierter Nachrichten über Internetprotokolle.
   Zusätzlich kann er vor der Erstnutzung die Schnittstellenbeschreibung aus einem allgemein zugänglichen Dienstverzeichnis beziehen.
3. Dienstverzeichnis.
   Es verwaltet die durch die Dienstanbieter bereitgestellten kategorisierten Schnittstellenbeschreibungen.

Ausgehend von den Grundrollen und ihren Interaktionsbeziehungen ergeben sich die zwei in Abbildung 56 Abläufe für die Abwicklung einer Web Service-basierten Kommunikation.



Die Abbildung hebt die optionalen Schritte zur Ermittlung der Schnittstellenbeschreibung blau hervor. Liegt diese dem Aufrufwilligen bereits vor oder ist die Schnittstelle ihm bereits bekannt, so kann auf den Bezug der im Dienstverzeichnis abgelegten Beschreibungsdaten verzichtet werden und der Dienstaufruf direkt erfolgen.
Die Schnittstellenbeschreibung nimmt damit keine herausragende Rolle innerhalb der Architekturerstellung ein, wie dies beispielsweise für CORBA, RMI oder DCOM der Fall wäre (dort ist die Schnittstellenbeschreibung Teil des Entwicklungszyklus und muß zum Übersetzungzeitpunkt des Aufrufclients zwingend vorliegen).

Aus der Kombination der Architekturelemente und der Interaktionsszenarien haben sich drei Standards am Markt etabliert, deren Implementierungen die dargestellten Grundaufgaben innerhalb einer serviceorientierten Architektur erfüllen:

- SOAP --- Das Kommunikationsprotokoll zur Kommunikationsabwicklung zwischen Dienstanbieter und -nutzer.
- WSDL (Web Service Description Language) --- Die XML-basierte Schnittstellenbeschreibungssprache.
- UDDI (Universal Service Description, Discovery and Integration) --- Ein Verzeichnisdienst zur Verwaltung von Schnittstellenbeschreibungen, der selbst als Web Service realisiert ist.

Anmerkung: Ursprünglich wurde das Akronym *SOAP* zu *Simple Object Access Protocol* expandiert. Aufgrund der irreführenden Nähe zur objektorientierten Programmierung wurde jedoch im Verlauf des Standardisierungsprozesses beschlossen etablierte Akronym als Namen beizubehalten, jedoch von der weiteren Expansion abzusehen.

SOAP

SOAP, welches als Version 1.0 im Herbst 1999 als Ergebnis der Kooperation zwischen den Firmen *DevelopMentor*, *IBM*, *Microsoft*, *Lotus* und *UserLand Software* vorgestellt wurde markiert weniger den Beginn der Idee der Web Services, als vielmehr einen weiteren evolutionären Entwicklungsschritt.
Die Uridee die zur Abwicklung synchroner entfernter Funktionsaufrufe (*Remote Procedure Calls* (RPC)) zu übermittelnden Über- und Rückgabeparameter durch ein XML-Vokabular auszudrücken findet sich bereits im Protokoll *XML-RPC* verwirklicht.

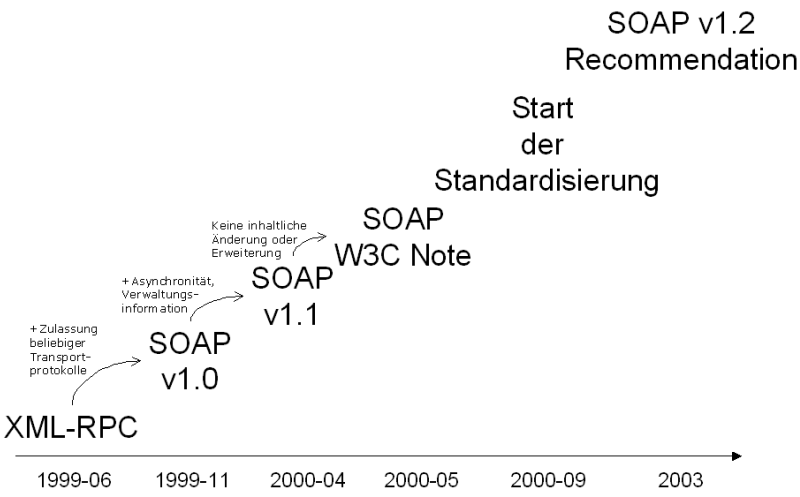
Ausgehend von diesem Ansatz definiert SOAP in der Version 1.0 ein vollständiges Protokoll, welches neben der Übertragung von Nutzdaten auch die Darstellung von dekriptiver Verwaltungsinformation vorsieht. Wie bereits für XML-RPC ist auch bei SOAP v1.0 ausschließlich die Verwendung von HTTP als Transportprotokoll definiert.

Diese Beschränkung wird durch die Anfang 2001 freigegebene SOAP Version 1.1 aufgehoben, welche SOAP als vollständig von der zu tatsächlichen Übertragung gewählten Protokollschicht entkoppelt und damit als eigenständige Protokollabstraktion etabliert. Gleichzeitig führt diese Version die Möglichkeit asynchroner Aufrufe ein.

Um den SOAP-Ansatz einer breiten Interessentenschicht zugänglich werden zu lassen und auch die Sensibilisierung für eine mögliche Standardisierung des Protokolls zu schaffen reichen die beteiligten Partner die Spezifikationsversion 1.1 unverändert als W3C Note ein.

Die im Herbst 2000 begonne Standardisierungsarbeit durch die W3C XML Protocol Arbeitsgruppe übernahm den eingereichten Spezifikationsvorschlag der Version 1.1 weitestgehend und führte, aus Gründen der Interoperabiltität behutsame Korrekturen sowie umfangreiche Erweiterungen ein. Mit der Verabschiedung des endgültigen W3C-Standards ist im Laufe des Jahres 2003 zu rechnen.

Die Abbildung 57 stellt nochmals die einzelnen SOAP-Versionen, ihre Unterschiede und den Vorläufer XML-RPC in der chronologischen Übersicht zusammen.



Aufbau einer SOAP-Nachricht

Jede SOAP-Nachricht, unabhängig davon ob es sich um eine synchron oder asynchron übermittelte Anfrage oder Antwort handelt besteht generell aus zwei Teilen:

1. Dem Rumpfelement (`Body`) zur Aufnahme der zu übermittelnden Nutzdaten.
2. Optional einem oder mehreren Kopfelementen (`Header`) zur Aufnahme von Metainformation.

Das Beispiel 121 zeigt einen vollständigen SOAP-Aufruf.
Er besteht aus einem *Umschlag* (dargestellt durch das Element `Envelope`), der das beschreibende Kopfelement `altertcontrol` sowie die im `Body`-Element zusammengefaßten Nutzdaten enthält.

Alle durch den Standard vorgegebenen Elemente und Attribute sind dem Namensraum `http://www.w3.org/2003/05/soap-envelope` zugeordnet und alle Anwenderdefinierten den entsprechenden eigenen Namensräumen.

Beispiel 121: Ein vollständiger SOAP-Aufruf

```
(1)<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
(2) <env:Header>
(3)  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
(4)   <n:priority>1</n:priority>
(5)   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
(6)  </n:alertcontrol>
(7) </env:Header>
(8) <env:Body>
(9)  <my:message xmlns:my="http://www.mycompany.com">
(10)   <my:text>Hello World!</my:text>
(11)  </my:message>
(12) </env:Body>
(13)</env:Envelope>
```

Die Aufgabe der *SOAP-Header* ist es Verwaltungsinformation aufzunehmen, die nicht ausschließlich durch den Empfänger des SOAP-Aufrufes zu verarbeiten ist und die nicht Teil der Nutzdaten ist.
Beispiele hierfür sind Daten über das Routingverhalten, gesetzte Sperren oder Transaktionen aber auch Zugriffsdaten wie Nutzername und Passwort (diese natürlich verschlüsselt!).

Die Verarbeitung eines Kopfelements ist vorgabegemäß optional, außer das zusätzliche gesetzte Attribut `mustUnderstand` weißt den Wert `1` oder `true` (die beiden durch XML Schema zugelassenen lexikalischen Repräsentationen für logisch *wahr*) auf. In diesem Falle muß ein *SOAP-Knoten* das Kopfelement verarbeiten. Kann die Verarbeitung nicht vorgenommen werden, so muß an den Sender der SOAP-Nachricht eine Fehlermeldung übermittelt werden.
Ein *SOAP-Knoten* ist hierbei jeder Rechner entlang des Nachrichtenpfades zwischen Sender und Empfänger, der das SOAP-Protokoll verarbeiten kann.

Soll die Verarbeitung der SOAP-Kopfelemente auf einen bestimmten Rechner (beispielsweise einen zwischenspeichernden Proxy-Knoten) beschränkt werden, so kann durch das im Standard vorhergesehene Attribut `role` eine eineindeutige Adressierung eines durch URI identifizierten (Zwischen-)Knotens erreicht werden.
Das Beispiel 122 zeigt die Nutzung des `role`- und des `mustUnderstand`-Attributs.

Beispiel 122: Nutzung der SOAP-Kopfelemente

```
(1)<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
(2)     <env:Header>
(3)     <abc:Extension1
(4)             xmlns:abc="http://example.org/2001/06/ext"
(5)             env:mustUnderstand="true"
(6)             role="http://www.example.com/xyz"/>
(7)     <def:Extension2 xmlns:def="http://example.com/stuff"
(8)             env:mustUnderstand="true"
(9)             env:role="http://www.w3.org/2003/05/soap-envelope/role/next"/>
(10)    </env:Header>
(11)    <env:Body>
(12)            ...
(13)    </env:Body>
(14)</env:Envelope>
```

Das Beispiel zeigt die Nutzung zweier Kopfelemente (`Extension1` und `Extension2`) in „eigenen“, d.h. nicht den im Standard vorgesehenen, Namensräumen.
Für beide Kopfelemente ist das Attribut `mustUnderstand` mit `true` belegt, was sie als zwingend zu prozessieren ausweist.
Zusätzlich ist für `Extension2` das Attribut `role` auf `http://www.example.com/xyz` gesetzt. Diese Belegung charakterisiert die durch `mustUnderstand` formulierte verpflichtende Verarbeitung näher. Im vorliegenden Falle muß sie ausschließlich durch den mit der URI `http://www.example.com/xyz` bezeichnenden Zwischenknoten erfolgen, für alle anderen Knoten des Nachrichtenpfades --- einschließlich des endgültigen Empfängers --- kann dieses Kopfelement ignoriert werden. Der dieses Kopfelement verarbeitende SOAP-Knoten darf es nach erfolgreicher Prozessierung aus der SOAP-Nachricht entfernen.
Bei der für das zweite Kopfelement (`Extension2`) gewählten Rolle (`http://www.w3.org/2003/05/soap-envelope/role/next`) handelt es sich um eine durch die SOAP-Spezifikation vordefinierte URI, welche alle Knoten entlang des Nachrichtenpfades einschließlich dem endgültigen Empfänger selbst bezeichnet. In diesem Fall darf das Kopfelement, selbst nach erfolgreicher Verarbeitung durch einen Knoten nicht entfernt werden, da weitere Verarbeitungsschritte durch andere Knoten folgen können. (Sofern es sich nicht um den endgültigen Empfänger handelt, müssen sogar weitere Verarbeitungsschritte folgen.)

Neben der dargestellten Sonderbelegung des Rollenattributes definiert der SOAP-Standard des W3C mit `http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver` eine Attributbelegung, die Kopfelemente kennzeichnet die ausschließlich durch den endgültigen Empfänger einer Nachricht verarbeitet werden dürfen und durch `http://www.w3.org/2003/05/soap-envelope/role/none` Kopfelemente, die durch einen SOAP-Knoten nicht verarbeitet werden dürfen.

Durch die Kopfelemente können Einzelknoten entlang des Vermittlungspfades einer SOAP-Nachricht pauschal oder gezielt zu einer anwenderdefinierten Verarbeitung angeregt werden. Die aktiven Zwischenknoten übernehmen hierbei nicht mehr nur reine vermittelnde Aufgaben auf den tieferliegenden (Transport-)Protokollschichten, sondern werden zu aktiven Elementen der SOAP-Nachrichtenkette. Aufgrund ihrer Stellung im Nachrichtenpfad zwischen Sender und endgültigem Empfänger werden sie auch mit dem Begriff *SOAP Intermediäre* belegt.

Die Übertragung der zwischen Dienstnutzer und Dienst ausgetauschten Daten (d.h. Aufruf- und Rückgabeparameter oder Einwegbotschaft) erfolgt durch das `Body`-Element der SOAP-Nachricht.
Beispiel 123 zeigt den erzeugten SOAP-Datenstrom zum Aufruf eines Dienstes der die beiden `int`-Parameter *a* und *b* addiert und das Ergebnis zurückliefert.

Beispiel 123: Nutzung des SOAP-Rumpfelements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<env:Envelope
(3)     xmlns:env="http://www.w3.org/2003/05/soap-envelope"
(4)     xmlns:b="http://www.example.org/AddService>
(5)     <env:Body>
(6)             <b:add env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
(7)             <b:a>2</b:a>
(8)             <b:b>3</b:b>
(9)             </b:add>
(10)    </env:Body>
(11)</env:Envelope>
```

Innerhalb des `Body`-Elements werden die benannten Parameter durch jeweils ein Element, das den Namen des Parameters trägt, dargestellt. Jedes Parameterelement enthält die lexikalische Repräsentation des zu übertragenden Wertes. Der aufgerufene Dienst (im Beispiel: *add*) fungiert als gemeinsames Elternelement der einzelnen Parameterelemente.
Zur Unterscheidung der verschiedenen Vokabularelemente des SOAP-Aufrufes müssen alle anwenderdefinierten Teile des `Body`-Elements einem eigenen, vom SOAP-Vorgabenamensraum abweichenden, Namensraum zugeordnet sein. Dies geschieht vor dem Hintergrund sowohl der gewünschten Abgrenzung von den durch den Standard vorgegebenen Elementen und Attributen, als auch mit der Zielsetzung zur Validierung der SOAP-Aufrufe eine XML-Schemainstanz einsetzen zu können. Hierzu sind die abweichenden Namensräume zwingend erforderlich, um mithilfe des any-Elements die Validierung der durch den SOAP-Standard strukturell nicht festlegbaren `Body`-Kindelemente zu ermöglichen.
Die Struktur dieser Kindelemente orientiert sich ausschließlich an der XML-Darstellung der für einen Service notwendigen Parameter und kann daher im allgemeinen Falle nicht durch den Standard vorgegeben werden. Aus

diesem Grunde legt die SOAP-Spezifikation lediglich Encodierungsregeln zur Transformation hauptspeicherresidenter Objekte und Strukturen in eine XML-Darstellung fest.
Das Beispiel verwendet diese vordefinierten Encodierungsregeln, welche die Abbildung allgemeiner Objektgraphen in XML-Strukturen gestatten. Neben dieser, an der Belegung des Attributs encodingStyle mit http://www.w3.org/2003/05/soap-encoding erkennbaren, Serialisierungsform können durch den Anwender beliebige eigene Regeln zur Gewinnung der XML-Darstellung festgelegt werden. Einzig das im Standard definierte SOAP-Encoding muß jedoch zwingend von allen Implementierungen unterstützt werden.

In ähnlicher Darstellungsform der Anfrage findet sich auch die Übermittlung der durch den aufgerufenen Dienst berechneten Resultate codiert:

Beispiel 124: Nutzung des SOAP-Rumpfelements

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<env:Envelope
(3)     xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
(4)     xmlns:b="http://www.example.org/AddService">
(5)     <env:Body>
(6)             <b:result env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">5</b:result>
(7)     </env:Body>
(8)</env:Envelope>
```

Beispiel 124 zeigt die rückübermittelte Antwort des Addier-Dienstes.
Auch sie folgt den selben Struktur- und Erstellungsprinzipien. Daher weist auch Antwortnachricht die Trennung in (optionale und in diesem Beispiel nicht genutzte) Kopfelemente und nutzdatentragende Rumpfelemente als Kindelemente des Body-Elements auf.
Die Erstellung der XML-Darstellung erfolgt, wie bereits für den Aufruf, gemäß festgelegter bzw. anwenderdefiniert festlegbarer Richtlinien einer Encodierungsdarstellung. Auch in diesem Beispiel wurde die Standardencodierung gewählt, weshalb das encodingStyle-Attribut abermals mit dem Wert http://www.w3.org/2003/05/soap-encoding versehen ist.
Zur Abgrenzung von den durch den W3C-Standard vordefinierten Elementen muß auch zur Darstellung der Antwort eines Dienstaufrufs jedes Kindelement des Body-Knotens einen vom SOAP-Standard abweichenden Namensraum besitzen.
Die Organisation der Darstellung des Dienstergebnisses als XML-Dokument ist dem Diensterbringer überlassen. Im Beispiel wird ein Element result übertragen.

Quellcode des Web Service
Quellcode eines Web Serviceaufrufes

Fallstudie: Implementierung eines Web Services

Die nachfolgende Fallstudie betrachteten einen einfachen Web Service zur Realisierung der mathematischen Operationen auf dem Körper der komplexen Zahlen.
Die Diskussion wird an der frei verfügbaren SOAP-Implementierung Axis geführt, welche im Rahmen des Apache-Projekts gepflegt wird. Axis stellt eine vollständig in Java realisierte Bibliothek zur Umsetzung von Web Services die als Java-Klassen angeboten werden zur Verfügung.
Zur Ausführung wird eine beliebige Servlet-Umgebung benötigt. Für das Beispiel wurde die ebenfalls kostenfrei verfügbare Jakarta-Tomcat eingesetzt, welche als Ausführungsumgebung der Web Services dient.

Implementierung des Web Service:

Beispiel 125: Beispielwebdienst

```
(1)public class Complex {
(2)     private double re;
(3)     private double im;
(4)
(5)     public Complex(double re, double im) {
(6)             this.re = re;
(7)             this.im = im;
(8)     }
(9)     public Complex() {
(10)            this.re = 0;
(11)            this.im = 0;
(12)    }
(13)
(14)    public void setRe(double re) {
(15)            this.re = re;
(16)    }
(17)    public void setIm(double im) {
(18)            this.im = im;
(19)    }
(20)    public double getIm() {
(21)            return im;
(22)    }
(23)    public double getRe() {
(24)            return re;
(25)    }
(26)    public double modulus(Complex c1) {
(27)            return Math.sqrt(Math.pow(c1.getRe(), 2) + Math.pow(c1.getIm(), 2));
```

```
(28)    }
(29)    public Complex add(Complex c1, Complex c2) {
(30)            return new Complex(
(31)                    c1.getRe() + c2.getRe(),
(32)                    c1.getIm() + c2.getIm());
(33)    }
(34)    public Complex mult(Complex c1, Complex c2) {
(35)            return new Complex(
(36)                    c1.getRe() * c2.getRe() - c1.getIm() * c2.getIm(),
(37)                    c1.getRe() * c2.getIm() + c1.getIm() * c2.getRe());
(38)    }
(39)    public Complex div(Complex c1, Complex c2) {
(40)            double div = Math.pow(c2.getRe(), 2) + Math.pow(c2.getIm(), 2);
(41)            return new Complex(
(42)                    (c1.getRe() * c2.getRe() + c1.getIm() * c2.getIm()) / div,
(43)                    (c1.getIm() * c2.getRe() - c1.getRe() * c2.getIm()) / div);
(44)    }
(45)    public Complex pow(Complex c1, int n){
(46)            if (n==0) {
(47)                    return new Complex(1,0);
(48)            }
(49)            Complex result = c1;
(50)            for (int i=1;i<n;i++){
(51)                    result = result.mult(c1, c1);
(52)            }
(53)            return result;
(54)    }
(55)    public String toString() {
(56)            String result = new String();
(57)            result += re;
(58)            if (im < 0) {
(59)                    result += "-";
(60)            } else {
(61)                    result += "+";
(62)            }
(63)            result += "i" + Math.abs(im);
(64)            return result;
(65)    }
(66)}
```

Beispiel 125 zeigt die Implementierung der Klasse `Complex` deren Methoden als Web Service angeboten werden. Augenscheinlich unterschiedet sich die Implementierung als Web Service nicht von der, die für die statische lokale Bereitstellung leistenden.
Bei der Umsetzung ist jedoch darauf zu achten, auf `this`-Verweise in Methodenrümpfen zu verzichten, da kein Objektkontext durch das SOAP-Protokoll mitversandt wird. Als pragmatisches Hilfsmittel zur Einhaltung dieser Einschränkung bietet es sich an die als Dienst bereitzustellenden Methoden mit dem Schlüsselwort `static` zu versehen.
Zusätzlich ist auf die Definition eines parameterlosen Vorgabekonstruktors zu achten, da dieser serverseitig zur durch das Framework zur Objekterzeugung herangezogen wird.

Die serverseitige Bereitstellung des Dienstes kann im Rahmen des Axis-Frameworks auf zwei Arten geschehen. Zum einen durch Bereitstellung des Quellcodes aus Beispiel 125 in einem dafür gesondert vorgesehen Serververzeichnis (typischerweise `../tomcat/webapps/axis`). Zusätzlich ist die Datei mit der Extension `jws` (statt dem üblichen `java`) zu versehen. Zum Aufrufzeitpunkt wird der Quellcode durch das Axis-Framework für den Aufrufer transparent übersetzt und zur Ausführung gebracht.
Andererseits sieht das Framework die Möglichkeit der Ablage von vorübersetztem Java-Code vor. Dieser in der Axis-Terminologie als *Web Service Deployment* bezeichnete Vorgang gestattet dem Anwender weiterrechenden Einfluß auf Umstände der Dienstbereitstellung als die zuvor gezeigte Vorgehensweise.

Notwendige Voraussetzung des Deploymentvorganges ist die Bereitstellung einer separaten Konfigurationsdatei, dem sog. *Deployment-Deskriptor* unter zwingend vorgegebenen dem Dateinamen `deploy.wsdd`. Beispiel 126 zeigt eine solche Konfigurationsdatei für den Beispieldienst.

Beispiel 126: Deploymentdeskriptor des Beispieldienstes

```
(1)<deployment
(2)     xmlns="http://xml.apache.org/axis/wsdd/"
(3)     xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
(4)   <service name="ComplexCalc" style="RPC">
(5)     <parameter name="className" value="Complex"/>
(6)      <parameter name="allowedMethods" value="add mult modulus div pow"/>
(7)      <beanMapping
(8)             qname="myNS:Complex"
(9)             xmlns:myNS="urn:Complex"
(10)            languageSpecificType="java:Complex"/>
(11)    </service>
(12)</deployment>
```

Innerhalb der Konfigurationsdatei werden Aussagen über das Interaktionsmuster, welches der Kommunikation mit dem Dienst zugrunde liegt. Im Beispiel wird durch die Belegung des Attributs `style` mit dem Wert `RPC` synchrone Kommunikation im Stile entfernter Methodenaufrufe gewählt.
Zusätzlich kann ein vom Namen des Compilats abweichender Dienstname frei festgelegt werden (im Beispiel `ComplexCalc` als Belegung des Attributs `name` des `service`-Elements).
Innerhalb der Kindelemente des Elements `service` wird die dienstimplementierende Klasse identifiziert (`parameter`-Element dessen `name`-Attribut den Wert `className` trägt, das zugehörige `value`-Attribut desselben Elements enthält dann den vollqualifizierten Klassennamen) sowie die per SOAP zugreifbaren Methoden mit separierenden Leerzeichen gemäß XML-Konvention aufgelistet (im Beispiel `add mult modulus div pow`).
Hierdurch eröffnet die Konfiguration des Webdienstes durch den Deploymentdeskriptor bereits einen Eingriffspunkt, der bei der reinen serverseitigen Ablage des Quellcodes, mit automatischer Freigabe aller öffentlich zugänglichen Methoden, nicht bestand.
Ferner können vermöge des Elements `beanMapping` anwenderdefinierte strukturierte Datentypen, die als Java-Bean-konforme Klasse umgesetzt sind definiert werden. Im Beispiel ist dies die Klasse zur Implementierung der komplexen Zahl.

Die Installation und Konfiguration (Deployment) des Web Service geschieht durch Aufruf des Administrationswerkeuges `java org.apache.axis.client.AdminClient deploy.wsdd`.
Zusätzlich ist die Dienstklasse im Verzeichnis `.../tomcat/webapps/axis/WEB-INF/classes` abzulegen.

Implementierung des Web Service Aufrufers:

Beispiel 127: Aufruf des Beispielwebdiensts

```
(1)import java.rmi.RemoteException;
(2)import javax.xml.namespace.QName;
(3)import javax.xml.rpc.ParameterMode;
(4)import javax.xml.rpc.encoding.XMLType;
(5)import org.apache.axis.client.Call;
(6)import org.apache.axis.client.Service;
(7)import org.apache.axis.encoding.ser.BeanSerializerFactory;
(8)import org.apache.axis.encoding.ser.BeanDeserializerFactory;
(9)
(10)public class callService {
(11)    public static void main(String[] args) {
(12)            Call call = new Call( new Service() );
(13)
(14)            QName qn = new QName("urn:Complex","Complex");
(15)
(16)            call.setTargetEndpointAddress("http://10.0.0.1:8080/axis/services/ComplexCalc");
(17)            call.setOperationName("pow");
(18)            call.registerTypeMapping(Complex.class, qn,
(19)                                    new BeanSerializerFactory(Complex.class, qn),
(20)                                    new BeanDeserializerFactory(Complex.class, qn));
(21)
(22)            call.addParameter("c1", qn, ParameterMode.IN);
(23)            call.addParameter("n", XMLType.XSD_INT, ParameterMode.IN);
(24)            Object[] param = new Object[2];
(25)            param[0] = new Complex(2,3);
(26)            param[1] = new Integer(3);
(27)
(28)            call.setReturnType( qn );
(29)
(30)            try {
(31)                    Complex result = (Complex) call.invoke(param);
(32)                    System.out.println("result=" + result);
(33)            } catch (RemoteException re) {
(34)                    re.printStackTrace();
(35)            }
(36)    }
(37)}
```

Beispiel 127 zeigt die Implementierung eines Aufrufs des Beispielwebdienstes.
Zentrales Objekt des Aufrufs eines Web Service ist die Klasse `Call`. Sie kapselt die gesamten zur Kommunikation notwendigen Daten und übernimmt den synchronen Aufruf des entfernten Dienstes. Zur Verwaltung von Verbindungsdaten, die für verschieden Einzelaufrufe desselben Dienstes gleichermaßen Verwendung finden können diese zusätzlich durch einen `Service` repräsentiert werden. Im Beispiel wird hierauf jedoch aus Übersichtlichkeitsgründen verzichtet und alle alle Einstellungen direkt für das `Call`-Objekt vorgenommen.
Zu diesen Einstellungen zählt die dargestellte Angabe des *Service Endpunktes*, derjenigen Adresse, die konform zum gewählten Übertragungsprotkoll (im Beispiel ist dies HTTP) die physische Übergabestelle des SOAP-Aufrufs an den Web Service identifiziert.
Daher ist bei Änderung der Dienstbereitstellung, etwa durch Verlagerung auf einen anderen Server oder auch die sich aus dem oben gezeigten geänderten Deployment ergebende Endpunktadresse, lediglich der Parameter der Methode `setTargetEndpointAddress` geeignet anzupassen.

Zusätzlich zur Angabe der aufzurufenden Operation innerhalb des angesprochenen Dienstes, durch die Methode `setOperationName` erfolgt durch `addParameter` die Definition der Übergabeparameter des Dienstes.
Hierbei muß jeder Übergabeparameter in Name, Typ und Übergabeart spezifiziert werden. Im Beispiel sind dies die beiden Parameter `c1` und `n`, die als Eingabe (d.h. nur zum lesenden Zugriff) der Methode `pow` dienen.

Diese Methode implementiert die Potenzierung komplexer Zahlen durch fortwährende Multiplikation. Ihre Signatur erfordert daher die Übergabe der komplexen Basis sowie des ganzzahligen Exponenten.

Zur Übertragung der speicherresidenten Wertdarstellung ordnet das Axis-Framework den Java-Primtivdatentypen und einigen ausgewählten als Klasse realisierten Datentypen eindeutige Abbildungen in die in XML-Schema definierten Datentypen zu.
Die Reihenfolge der Aufrufe der `addParameter`-Methode muß der Auftrittsreihenfolge in der Signatur des aufzurufenden Dienstes entsprechen, um die Kompatibilität zur Programmiersprachen, die ausschließlich über Stellungsparameter verfügen zu wahren.
Tabelle 31 stellt die durch das Axis-Framework Typäquivalenzen zusammen:

Tabelle 31: Abbildung der Java-Datentypen auf XML-Schema

Java	XML-Schema	Bemerkung
`byte[]`	`base64Binary` `hexBinary`	Beide Darstellungsweisen sind gleichwertig möglich.
`boolean`	`boolean`	
`byte`	`byte`	
`java.util.Calendar`	`dateTime`	
`java.math.BigDecimal`	`decimal`	
`double`	`double`	
`float`	`float`	
`int`	`int`	
`java.math.BigInteger`	`integer`	
`long`	`long`	
`javax.xml.namespace.QName`	`QName`	Diese Javaklasse ist Bestandteil der API-Erweiterung für XML, die separat bezogen werden muß.
`short`	`short`	
`java.lang.String`	`string`	

Bei der zu übergebenden komplexen Zahl handelt es sich um eine anwenderdefinierte Klasse, die einen neuen Java-Typ etabliert für den (naturgemäß) keine Entsprechung in XML-Schema zur Verfügung steht.
Aus diesem Grund muß auf Seiten des Aufrufers und des aufgerufenen Dienstes eine eigene Abbildungsvorschrift für die Erzeugung der SOAP-Darstellung bereitgestellt werden.

Durch die Axis-Serviceausführungsumgebung kann automatisiert eine solche Abbildung erzeugt werden, sofern der anwenderdefinierte Typ (d.h. die implementierende Klasse) gemäß der Java-Beans-Programmierkonvention umgesetzt ist. Diese ist eingehalten, sobald für jedes datenhaltende Attribut eine get- und set-Methode umgesetzt wird. Hintergrund dieses Vorgehens ist die Möglichkeit der Abfrage aller objektinternen Datenfelder durch Nutzung der Java-Reflection-API.
Die hierfür notwendige Zuordnung zwischen neuem Java- und XML-Typ geschieht im durch Beispiel 126 dargestellten Deploymentdeskriptor ab Zeile sieben. Dort gibt das Attribut `qname` den Namen des XML-Typs und `languageSpecificType` den Namen der implementierenden Java-Klasse (`Complex`) an.

Entsprechend vollzieht der Dienstaufrufer die Typabbildung im Code nach. Hierzu muß ihm der gewählte Name des neuen XML-Typen bekannt sein (im Beispiel: `Complex` im Namensraum `urn:Complex`). Durch den Aufruf der Methode `registerTypeMapping` gibt er dem Framework die Implementierungen der beiden Abbildungsrichtungen (Java-Hauptspeicherobjekte in SOAP bzw. SOAP zu Hauptspeicherobjekten) bekannt. Das Beispiel verwendet hierzu die im Axis-Framework mitgelieferten Klassen `BeanSerializerFactory` und `BeanDeserializerFactory` welche das Gewünschte auf Basis einer als Java-Bean codierten Klasse leisten.

Die Festlegung der tatsächlichen Übergabewerte erfolgt im Beispiel durch Erzeugung eines Arrays zur Aufnahme von `Object`-typisierten Elementen in die die zuvor hinsichtlich ihres Typs definierten Aktualparameter eingefügt werden. Aufgrund der Einschränkung der Programmiersprache Java (bis zur Version 1.4), die Ausprägungen von Primitivtypen nicht als Objekte behandeln kann, erfolgt im Beispiel die Kapselung des `int`-Wertes für n durch ein Objekt der Klasse `Integer`.

Für die Definition des Typs des erwarteten Rückgabewertes gelten dieselben Gesetztmäßigkeiten hinsichtlich Bekanntmachung und Zugriffsabwicklung. Die Definition des erwarteten Rückgabetyps erfolgt durch Aufruf der Methode `setReturnType`.

Der synchrone Aufruf des entfernten Dienstes wird dann durch die Methode `invoke` des erzeugten und entsprechend parametrisierten `Call`-Objektes vollzogen.
Dieser Aufruf ist blockierend und läßt den Aufrufer bis zum Eintreffen des berechneten Ergebnisses warten.

Die Rückgabe wird, trotz der Definition des spezielleren Rückgabetyps `Complex` generell als `Object`-Ausprägung geliefert um eine strikt typprüfbare Signatur der Methode `invoke` zu erhalten. Daher muß das durch den Web Service gelieferte Resultat geeignet, d.h. konform zur durch `setReturnType` getroffenen Festlegung, typgewandelt werden.

Beispiel 128 zeigt den SOAP-Datenverkehr beim Aufruf des Dienstes zur Berechnung der Potenz einer komplexen Zahl. Aufgrund des derzeitigen Entwicklungsstandes des Axis-Frameworks weicht der Leitungsmittschnitt von der zu Eingang dieses Kapitels vorgestellten SOAP-Struktur des W3C-Standards ab. Gegenwärtig (d.h. zum Zeitpunkt der Verfügbarkeit der Version 1.1) unterstützt das Framework nur eine Untermenge des neuen Standards und codiert die XML-Nachrichten noch nach dem Stand der Vorgängerspezifikation.

Beispiel 128: SOAP-Nachricht zum Aufruf des Beispieldienstes

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<soapenv:Envelope
(3)     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
(4)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(5)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
(6)     <soapenv:Body>
(7)             <pow soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
(8)             <c1 href="#id0"/>
(9)             <n xsi:type="xsd:int">3</n>
(10)            </pow>
(11)            <multiRef
(12)                    id="id0"
(13)                    soapenc:root="0"
(14)                    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
(15)                    xsi:type="ns1:Complex"
(16)                    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
(17)                    xmlns:ns1="urn:Complex">
(18)            <im xsi:type="xsd:double">3.0</im>
(19)            <re xsi:type="xsd:double">2.0</re>
(20)            </multiRef>
(21)    </soapenv:Body>
(22)</soapenv:Envelope>
```

Im Code des Beispiels gut zu sehen ist die Darstellung der beiden Übergabeparameter `c1` und `n` für die Basis der Potenzierung und den ganzzahligen Exponenten. Für `n` wird die in Tabelle 128 Abbildung in den äquivalenten Typen `int` aus XML-Schema durchgeführt. Zusätzlich überträgt die durch Axis gewählte Codierung die Typisierung explizit mit (Attribut `xsi:type="xsd:int"`) wodurch die Typabbildung offenbar wird.
Für den nicht direkt nach XML-Schema transformierbaren strukturierten (Java-)Datentypen `Complex` wird durch den Serialisierungsmechanismus ein eigenes mit `multiRef` bezeichnetes und durch das Attribut `id` identifiziertes Element erzeugt. Dieses Vorgehen entspricht der durch die SOAP-Spezifikation vorgesehenen Serialisierungs allgemeiner Graphen, welche die speicherresidenten Datenobjekte als Knoten interpretiert und hierfür wiederverwendbare (der Name des Elements deutet dies an) Elemente erzeugt. Die tatsächliche Wiederverwendung innerhalb des SOAP-Stromes findet durch Mehrfachnennung des im `id`-Attribut festgelegten identifizierenden Wertes im `href`-Attribut eines Verweiselementes statt.
Im Beispiel wird die Berechnungsbasis als ein solches `multiRef`-Element ausgedrückt, welches die Wertbelegungen der durch get-Methoden zugänglichen Javafelder (`im` und `re`) enthält.
Das den Übergabeparameter `c1` repräsentierende XML-Element enthält daher lediglich im `href`-Attribut einen Verweis auf die `multiRef`-Struktur.

Beispiel 129 zeigt den für die Übermittlung des Dienstergebnisses übertragenen SOAP-Datenverkehr.
Auch er weicht, aufgrund der Konformität zur älteren SOAP-Spezifikationsversion, geringfügig vom aktuellen Standard ab.

Beispiel 129: SOAP-Nachricht zur Übermittlung des Berechnungsergebnisses des Beispieldienstes

```
(1)<?xml version="1.0" encoding="UTF-8"?>
(2)<soapenv:Envelope
(3)     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
(4)     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(5)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
(6)     <soapenv:Body>
(7)             <powResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
(8)     <powReturn href="#id0"/>
(9)     </powResponse>
(10)    <multiRef
(11)            id="id0"
(12)            soapenc:root="0"
(13)            soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
(14)            xsi:type="ns1:Complex"
(15)            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
(16)            xmlns:ns1="urn:Complex">
(17)    <im xsi:type="xsd:double">12.0</im>
(18)    <re xsi:type="xsd:double">-5.0</re>
(19)    </multiRef>
(20)    </soapenv:Body>
(21)</soapenv:Envelope>
```

Wie bereits beim Aufruf realisiert wird die XML-Darstellung des strukturierten Datentyps `Complex`, der hier als Rückgabetyp dient, durch Nutzung der allgemeinen Graphenserialisierung erzeugt.

WSDL

Die Grundidee der Web Services fordert nicht zwingend die Darstellung der Dienstschnittstellen für Dritte, wie dies beispielsweise Verteilungsarchitekturen wie CORBA und DCOM zwingend als Teil des Entwicklungszyklus vorgeben. Dennoch erweist es sich auch für Web Services sinnvoll die Schnittstellenbeschreibungen in einem maschinenlesbaren Format bereitzustellen um Werkzeugen die automatisierte Verarbeitung zu ermöglichen. Hierunter fallen beispielsweise Entwicklungsumgebungen, die aus Schnittstellenbeschreibungen erste Rohgerüste für

die Abwicklung der Dienstaufrufe generieren können. Zusätzlich stellen formalisiert dokumentierte Schnittstellen einen wichtigen Bestandteil der Dokumentation eines Softwaresystems dar.
Zur Schnittstellendokumentation von Web Services etabliert sich gegenwärtig der Standard der *Web Service Description Language*, der durch eine Arbeitsgruppe des World Wide Web Konsortiums vorangetrieben wird.

Eine WSDL-Beschreibung selbst ist eine XML-Datei, WSDL mithin als XML-Schema-basiertes XML-Vokabular realisiert und zerfällt in sechs Teile:

1. **Service**: Zur allgemeinen Beschreibung des angebotenen Dienstes
2. **Operations**: Zur Beschreibung der angebotenen Operationen
3. **Messages**: Zur Beschreibung Signatur der einzelnen Nachrichten (Aufrufe und Antworten)
4. **PortTypes**: Zur Verknüpfung von *Operations* und *Messages*
5. **Bindings**: Zur Verknüpfung einer *Operation* mit einem Transportprotkoll
6. **Types**: Zur Definition anwenderdefinierter Übergabe- und Rückgabetypen

Die Angaben zur Struktur der anwenderdefinierten Typen ist optional und muß nur im Falle der Existenz solcher Typen erfolgen.

Das Beispiel 130 zeigt die vollständige WSDL-Dienstbeschreibung des aus der Fallstudie bekannten Dienstes:

Beispiel 130: WSDL-Beschreibung des Beispieldienstes

```
(1)<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.
org/xml-soap" xmlns:impl="http://10.0.0.1:8080/axis/services/Complex" xmlns:
intf="http://10.0.0.1:8080/axis/services/Complex" xmlns:soapenc="http://schemas.xmlsoap.org/soap/
encoding/" xmlns:tns1="urn:Complex" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:
wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://10.0.0.1:8080/axis/services/Complex">
(2)     <wsdl:types>
(3)             <schema targetNamespace="urn:Complex" xmlns="http://www.w3.org/2001/XMLSchema">
(4)                     <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
(5)                     <complexType name="Complex">
(6)                             <sequence>
(7)                                     <element name="im" type="xsd:double"/>
(8)                                     <element name="re" type="xsd:double"/>
(9)                             </sequence>
(10)                    </complexType>
(11)            </schema>
(12)    </wsdl:types>
(13)    <wsdl:message name="addResponse">
(14)            <wsdl:part name="addReturn" type="tns1:Complex"/>
(15)    </wsdl:message>
(16)    <wsdl:message name="divRequest">
(17)            <wsdl:part name="c1" type="tns1:Complex"/>
(18)            <wsdl:part name="c2" type="tns1:Complex"/>
(19)    </wsdl:message>
(20)    <wsdl:message name="modulusRequest">
(21)            <wsdl:part name="c1" type="tns1:Complex"/>
(22)    </wsdl:message>
(23)    <wsdl:message name="divResponse">
(24)            <wsdl:part name="divReturn" type="tns1:Complex"/>
(25)    </wsdl:message>
(26)    <wsdl:message name="multResponse">
(27)            <wsdl:part name="multReturn" type="tns1:Complex"/>
(28)    </wsdl:message>
(29)    <wsdl:message name="powResponse">
(30)            <wsdl:part name="powReturn" type="tns1:Complex"/>
(31)    </wsdl:message>
(32)    <wsdl:message name="addRequest">
(33)            <wsdl:part name="c1" type="tns1:Complex"/>
(34)            <wsdl:part name="c2" type="tns1:Complex"/>
(35)    </wsdl:message>
(36)    <wsdl:message name="modulusResponse">
(37)            <wsdl:part name="modulusReturn" type="xsd:double"/>
(38)    </wsdl:message>
(39)    <wsdl:message name="powRequest">
(40)            <wsdl:part name="c1" type="tns1:Complex"/>
(41)            <wsdl:part name="n" type="xsd:int"/>
(42)    </wsdl:message>
(43)    <wsdl:message name="multRequest">
(44)            <wsdl:part name="c1" type="tns1:Complex"/>
(45)            <wsdl:part name="c2" type="tns1:Complex"/>
(46)    </wsdl:message>
(47)    <wsdl:portType name="Complex">
(48)            <wsdl:operation name="pow" parameterOrder="c1 n">
(49)                    <wsdl:input name="powRequest" message="impl:powRequest"/>
(50)                    <wsdl:output name="powResponse" message="impl:powResponse"/>
(51)            </wsdl:operation>
(52)            <wsdl:operation name="add" parameterOrder="c1 c2">
(53)                    <wsdl:input name="addRequest" message="impl:addRequest"/>
(54)                    <wsdl:output name="addResponse" message="impl:addResponse"/>
(55)            </wsdl:operation>
(56)            <wsdl:operation name="mult" parameterOrder="c1 c2">
```

```
(57)                    <wsdl:input name="multRequest" message="impl:multRequest"/>
(58)                    <wsdl:output name="multResponse" message="impl:multResponse"/>
(59)            </wsdl:operation>
(60)            <wsdl:operation name="modulus" parameterOrder="c1">
(61)                    <wsdl:input name="modulusRequest" message="impl:modulusRequest"/>
(62)                    <wsdl:output name="modulusResponse" message="impl:modulusResponse"/>
(63)            </wsdl:operation>
(64)            <wsdl:operation name="div" parameterOrder="c1 c2">
(65)                    <wsdl:input name="divRequest" message="impl:divRequest"/>
(66)                    <wsdl:output name="divResponse" message="impl:divResponse"/>
(67)            </wsdl:operation>
(68)    </wsdl:portType>
(69)    <wsdl:binding name="ComplexSoapBinding" type="impl:Complex">
(70)            <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
(71)            <wsdl:operation name="pow">
(72)                    <wsdlsoap:operation/>
(73)                    <wsdl:input>
(74)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(75)                    </wsdl:input>
(76)                    <wsdl:output>
(77)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(78)                    </wsdl:output>
(79)            </wsdl:operation>
(80)            <wsdl:operation name="add">
(81)                    <wsdlsoap:operation/>
(82)                    <wsdl:input>
(83)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(84)                    </wsdl:input>
(85)                    <wsdl:output>
(86)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(87)                    </wsdl:output>
(88)            </wsdl:operation>
(89)            <wsdl:operation name="mult">
(90)                    <wsdlsoap:operation/>
(91)                    <wsdl:input>
(92)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(93)                    </wsdl:input>
(94)                    <wsdl:output>
(95)                            <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(96)                    </wsdl:output>
(97)            </wsdl:operation>
(98)            <wsdl:operation name="modulus">
(99)                    <wsdlsoap:operation/>
(100)                   <wsdl:input>
(101)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(102)                   </wsdl:input>
(103)                   <wsdl:output>
(104)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(105)                   </wsdl:output>
(106)           </wsdl:operation>
(107)           <wsdl:operation name="div">
(108)                   <wsdlsoap:operation/>
(109)                   <wsdl:input>
(110)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(111)                   </wsdl:input>
(112)                   <wsdl:output>
(113)                           <wsdlsoap:body use="encoded" encodingStyle="http://schemas.xmlsoap.
org/soap/encoding/" namespace="http://10.0.0.1:8080/axis/services/Complex"/>
(114)                   </wsdl:output>
(115)           </wsdl:operation>
(116)   </wsdl:binding>
(117)   <wsdl:service name="ComplexService">
(118)           <wsdl:port name="Complex" binding="impl:ComplexSoapBinding">
(119)                   <wsdlsoap:address location="http://10.0.0.1:8080/axis/services/Complex"/>
(120)           </wsdl:port>
(121)   </wsdl:service>
(122)</wsdl:definitions>
```

Das Beispiel spiegelt im **Service**-Element die bereits durch den Deploymentdeskriptor getroffenen Festlegungen wieder und macht sie so dem (potentiellen) Dienstnutzer zugänglich. So enthält das Attribut `name` mit dem Wert `ComplexService` den gewählten Namen des Dienstes. Zusätzlich der als Kindelement realisierte `address`-Eintrag

unter `location` die Adresse des SOAP-Endpunktes wieder welche SOAP-Botschaften zum Dienstaufruf entgegennimmt.
Da derselbe Dienst durch verschiedene Endpunkte zur Verfügung gestellt werden kann, ist das mehrfache Auftreten von `address`-Element zugelassen. Aus Gründen der Eindeutigkeit und Zuordnung des Endpunktes zum jeweils unterstützten Transportprotokoll ist jedes `address`-Element durch ein `port`-Element umschlossen welches auf das definierte Protokoll verweist.

Jedes **Operation**-Element definiert die zum Aufruf zu übermittelnden Parameter hinsichtlich Reihenfolge und referenziert die eine Interaktion konstituierenden Interaktionsschritte.
Im Beispiel legt die Operation `pow` (die Benennung wird durch das `name`-Attribut festgelegt) fest, daß die beiden Parameter `c1` und `n` benötigt werden. Zusätzlich erfolgt durch das Element `input` eine Referenz auf die Eingabenachricht bzw. `output` auf die Ausgabenachricht die zur Initiierung der Operation bzw. nach deren Ende versandt werden.

Innerhalb des **Message**-Elements werden die Übergabe Parameter inhaltlich hinsichtlich Typ und Name ausdefiniert. Für die Nachricht `powRequest` (Aufruf der Methode `pow`) werden daher `c1` vom Typ eigendefinierten Typ `Complex` sowie `n` vom Standardtyp `int` genannt.

Durch das WSDL-Element `PortType` erfolgt die Aggregation der zuvor definierten Operationen, deren Nachrichten, an den in der Service-Definition genannten Endpunkt.

Die technischen Details wie gewähltes Encodierungsschema und genutzter Namensraum werden im Rahmen des **Binding**-Elements für jeden Nichtstandardtypen --- im Beispiel ist dies `Complex` --- definiert.

Die zur Erzeugung der über die Netzwerkleitung zu transportierenden XML-Darstellung dieser Nichtstandardtypen wird durch ein XML-Schemafragment beschrieben, welches als Kindelement des WSDL-Elements **Types** abgelegt ist.

UDDI

Zwar bietet WSDL eine wertvolle Möglichkeit zur Beschreibung der technischen Schnittstellencharakteristika eines Web Service, jedoch bleiben andere Fragestellungen --- etwa die nach der Natur und menschenlesbaren Beschreibung eines Dienstes oder seinen Nutzungs- und Abrechnungsbedingungen --- durch diesen Standard vollkommen offen. Überdies enthält das offizielle Spezifikationsdokument keinerlei Aussagen über einen präferierten oder auch nur sinnvollen Bereitstellungsort der WSDL-Beschreibungen.
Einen Ansatz zur Antwort auf diese Fragestellungen versucht der im Rahmen des OASIS-Standardisierungsprozeß vorangetriebene Verzeichnisdienst *Universal Service Description, Discovery and Integration* zu liefern.

Dieses, selbst als SOAP-basierter Web-Dienst angebotene, Verzeichnisdienst stellt eine Verwaltungsstruktur zur Ablage von WSDL-Beschreibungen und anderer dienstbezogener deskriptiver Metadaten bereit die durch eine definierte Schnittstelle abgefragt werden kann.

Die Grundprimitive des UDDI-Dienstes sind:

- **Business Entity**: Der Anbieter eines Web Service
- **Business Service**: Der angebotene Dienst
- **tModel**: Die Beschreibung des Dienstes
- **BindingTemplate**: Verknüpfung von tModel und Business Service

Ziel der **Business Entity**-Struktur ist es telephonbuchartig beschreibende Metadaten zum Dienstanbieter, wie Firmenname und Erreichbarkeitsdaten in einer alphabetisch sortierten Struktur anzubieten.

Jedem Business Entity-Eintrag können mehrere **Business Services** zugeordnet sein, welche die angebotenen Web Services repräsentieren.

Jedes **tModel** (Abkürzung für *technical model*) kann eine WSDL-Beschreibung oder beliebige durch den Anwender festlegbare dienstbezogene deskriptive Inhalte aufnehmen.
Insbesondere kann ein tModel auch Kategorisierungen eines Dienstes, etwa die Zuordnung zu einer Dienstfamilie, die Herstellung eines bestimmten Typ-Kontexts wie Erbringungsort des Dienstes, aufnehmen.

Alle tModels eines Dienstes werden mit diesem durch **BindingTemplate**-Elemente verbunden.

Die Interaktion mit einem UDDI-Verzeichnisdienst kann SOAP-basiert oder durch manuelle Erfassung der abzulegenden Daten über eine Webseite erfolgen.
Ziel dieser Interaktion ist zumeist einer der global angebotenen UDDI-Verzeichnisdienste, die gegenwärtig innerhalb eines Tages für die vollständige Inhaltsreplizierung sorgen.

Aufgrund immernoch offener Sicherheitsfragestellungen und der als ungelöst anzusehenden Abrechnungsproblematik liegen jedoch in den aktuell zugänglichen UDDI-Diensten kaum kommerzielle Dienstangebote vor, sondern überwiegend Testeinträge.

Web-Referenzen 21: Weiterführende Links

- SOAP-Standard des W3C
  - Primer
  - Messaging Framework
  - Adjuncts
- Spezifikation der Web Service Description Language
- UDDI-Spezifikation
- Axis-Framework Eine Open-Source SOAP-Implementierung
- Aktuelle Informationen rund um Web Services

3.6 XML und Datenbanken

Seit dem Aufkommen XMLs als universellem Format zur Darstellung beliebiger Daten wird die Fragestellung nach einer adäquaten Speicherung von XML-codierten Daten diskutiert. Die Anforderungen beschränken sich hierbei nicht auf die Betrachtung der reinen Ablage im Dateisystem oder der Verwaltung durch ein Datenbankmanagementsystem, sondern beziehen auch Aspekte der Performance der lesenden und schreiben Operationen auf diesen Daten sowie der Formulierung flexibler Anfragen ein. Gleichzeitig ist jedoch die aufgeworfene Frage nach der sinnvoll(st)en Repräsentation XML-artiger semistrukturierter Daten innerhalb einer Persistenzschicht noch nicht eindeutig und letztgültig beantwortet worden, sofern sie dies überhaupt ist.

Generell bieten sich zur Ablage von XML drei Klassen von Speicherungsmechanismen an, die gegenwärtig alle in der Praxis verwendet werden:

1. Ablage als unstrukturierter (Text-artiger oder binärer) Datenstrom im Datei- oder einem Datenbanksystem.
2. Transformation in logische Datenbankstrukturen.
3. Speicherung in einem „nativen“ XML-Datenbanksystem.

Diese drei Klassen stellen jedoch nur eine erste taugliche Grobeinteilung dar und sind intern durchaus unterstrukturiert und teilweise höchst fragmentiert.

Speicherung von XML-Inhalten als unstrukturierte Datenströme

Grundidee und Konzepte
Naheliegendste Form der Speicherung von XML-Inhalten ist zweifellos die Ablage des XML-Stromes, der ohnehin zumeist als Datei vorliegt, in einem klassischen Dateisystem (z.B. ext2, NTFS, FAT).
In dieselbe Klasse fällt auch die Speicherung als unstrukturierter Inhalt in einem Datenbankmanagementsystem. Hierzu werden zumeist Datentypen wie BLOB (*binary large object*) oder CLOB (*character large object*) eingesetzt, die zur Aufnahme großer Mengen binärer oder textartiger Daten geeignet sind.

Beispiel
Die Speicherung im Dateisystem erfolgt (trivialerweise) in Dateien, d.h. die Erzeugung der XML-Daten beschränkt sich auf die Anlage, Benennung und Katalogszuordnung der Datei. Die XML-Daten werden aus einem Applikation mittels einer XML-Programmierschnittstelle direkt geschrieben oder durch per Netzwerk erhalten.

Die Speicherung in einem DBMS erfordert die Definition einer entsprechenden logischen Struktur zur Aufnahme der XML-Daten, die datenbankintern auf physische Speicherstrukturen abgebildet wird.
131 zeigt eine Implementierung für das relationale Datenbankmanagementsystem MySQL, die eine per Kommandozeilenargument übergebene Datei im Feld `DATA` der Datenbanktabelle `XML` ablegt. Das Feld mit `LONGTEXT` typisiert, was im verwendenten DBMS die Aufnahme von 2^{32}-1 Zeichen gestattet. Physisch wird `LONGTEXT` auf einen BLOB mit case-insensitiver Suchmöglichkeit abgebildet.

Beispiel 131: Unstrukturierte Speicherung von XML-Dokumenten in einer Datenbank

```
(1)import java.io.File;
(2)import java.io.FileInputStream;
(3)import java.io.IOException;
(4)import java.sql.Connection;
(5)import java.sql.DriverManager;
(6)import java.sql.PreparedStatement;
(7)import java.sql.SQLException;
(8)import java.sql.Statement;
(9)
(10)public class XML2DB {
(11)    public static void main(String[] args)
(12)    throws ClassNotFoundException, SQLException, IOException {
(13)            Class.forName("com.mysql.jdbc.Driver");
(14)            Connection con =
(15)                            (Connection) DriverManager.getConnection(
(16)                                    "jdbc:mysql://10.0.0.1/XMLunstructured/",
(17)                                    "mario",
(18)                                    "thePassword");
(19)
(20)            Statement stmt = con.createStatement();
(21)            stmt.executeUpdate("CREATE TABLE IF NOT EXISTS XML(Data LONGTEXT);");
(22)
(23)            File file = new File(args[0]);
(24)            FileInputStream fis = new FileInputStream(args[0]);
(25)            PreparedStatement pstmt =
(26)                    con.prepareStatement("INSERT INTO XML VALUES(?)");
```

```
(27)            pstmt.setBinaryStream(1, fis, (int) file.length());
(28)            pstmt.executeUpdate();
(29)            fis.close();
(30)    }
(31)}
```

Download des Beispiels

Zugriffe auf Inhalte (beispielsweise Extraktion eines Teilbaumes per XPath) des abgelegten XML-Dokumentes können datenbankseitig nicht direkt unterstützt werden, da das gesamte Dokument als (große) Zeichenkette interpretiert wird. Alle gewünschten Zugriffe müssen daher auf die vorhandenen Funktionen zur Auswertung von Zeichenketten abgebildet werden. Teilweise stellen die Datenankhersteller angepaßte Module zum Zugriff auf Volltextinformation zur Verfügung, die diesen Schritt vollziehen.

Das Auslesen des in der Datenbank abgelegten Dokuments vollzieht sich invers zur Ablage ebenfalls in Form von reinen Operationen zur Verarbeitung des gespeicherten Binärstroms, ohne Berücksichtigung seiner Natur als XML-Dokument. Beispiel 132 zeigt dies:

Beispiel 132: Auslesen eines unstrukturierte gespeicherten XML-Dokuments

```
(1)import java.io.FileDescriptor;
(2)import java.io.FileOutputStream;
(3)import java.io.IOException;
(4)import java.sql.Connection;
(5)import java.sql.DriverManager;
(6)import java.sql.ResultSet;
(7)import java.sql.SQLException;
(8)import java.sql.Statement;
(9)
(10)public class DB2XML {
(11)
(12)    public static void main(String[] args)
(13)    throws ClassNotFoundException, SQLException, IOException {
(14)            Class.forName("com.mysql.jdbc.Driver");
(15)            Connection con  =
(16)                    (Connection) DriverManager.getConnection(
(17)                            "jdbc:mysql://10.0.0.1/XMLunstructured/",
(18)                            "mario",
(19)                            "thePassword");
(20)            Statement stmt = (Statement) con.createStatement();
(21)            ResultSet rs =
(22)                    stmt.executeQuery(
(23)                            "SELECT Data FROM XML;");
(24)
(25)            FileOutputStream fos = new FileOutputStream(FileDescriptor.out);
(26)            if (rs.next())
(27)                    fos.write(rs.getBytes(1));
(28)            fos.close();
(29)    }
(30)}
```

Download des Beispiels

Diskussion der Vor- und Nachteile

Für das Konzept der unstrukturierten binären oder zeichenorientierten Ablage spricht die **Verfügbarkeit** seitens der bestehenden Datenbankmanagementsysteme, da die Einführung einer „XML-Unterstützung“ in dieser Form keinerlei Modifikationen am bestehenden System bedarf. Im selben Sinne können XML-Dokumente in Form von Textdateien durch bestehende Betriebssysteme verwaltet werden.
Gleichzeitig können die Lese- und Schreiboperationen in vergleichsweise **großer Geschwindigkeit** durchgeführt werden, da keine Abbildungen in interne Speicherstrukturen erfolgen muß.
Andererseits können XML-typische **Zugriffsoperationen**, durch die Lokatorsprache XPath oder die Anfragesprache XQuery für Dateien nur durch separate Lösungen und für Datenbankmanagementsysteme, nur durch Zeichenkettenverarbeitungsfunktionen oder darauf aufsetzende Module bereitgestellt werden. Im Allgemeinen müssen Zugriffsschritte, welche XML-Strukturen ausbeuten diese zunächst dynamisch zur Laufzeit im Hauptspeicher aus dem Zeichenstrom des Dokumentes re-generieren, was sich negativ auf die Ausführungszeit auswirkt.
Dieses mangelnde Wissen über internen Aufbau und Struktur eines XML-Dokuments erfordert zusätzliche Schritte zur Gewährleistung der **Konsistenz** der verwalteten Daten. So muß der Anwender Sorge dafür tragen, daß nur wohl-geformte XML-Dokumente als solche abgelegt werden, da weder das Dateisystem noch das DBMS für allgemeine Binärströme geeignete Validierungsfuntionen bereitstellen.

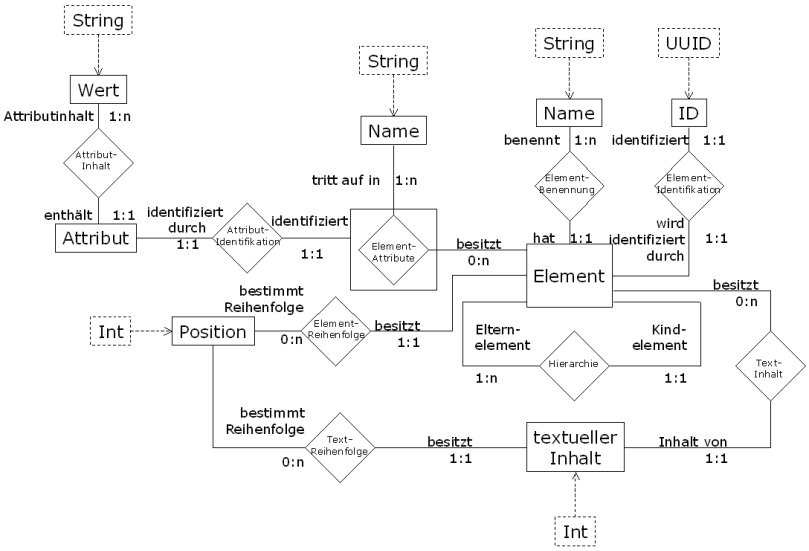
Abbildung von XML-Inhalten auf logische DB-Strukturen

Grundidee und Konzepte
Zur Behebung der formulierten Nachteile des naiven Speicherungsansatzes --- unter gleichzeitiger Erhaltung des Vorteils der Verfügbarkeit ohne Änderungen oder Erweiterungen an bestehenden Verwaltungssystemen vornehmen zu müssen --- bietet sich die Speicherung in eigens geschaffenen logischen Strukturen konform zu einem bestehenden logischen Modell an. Typischerweise wird hierzu eine generische oder dokumenttypsspezifische Abbildung der XML-Strukturen in das den verwaltbaren Datenbanken zugrundeliegende Modell definiert. Hierbei

finden überwiegend relationale Systeme Einsatz, Objektorientiere oder Hierarchische sind hingegen nur selten anzutreffen.
Naturgemäß bedarf die Verfolgung dieses Ansatzes der Existenz eines Verwaltungssystems, welches die logischen Strukturen bereitstellt und verarbeitet. Aus diesem Grunde scheiden ausschließlich Datei-basierte Ansätze, die sich nicht zusätzlicher interpretierender Applikationen bedienen, welche die Semantik der verwalteten Strukturen kennen, aus.

Beispiel
Die Applikation der Beispiele 133 und 134 setzt eine bijektive Abbildung allgemeiner XML-Strukturen in logische Relationenstrukturen um.

Abbildung 58 zeigt einen Ausschnitt des konzeptuellen Schemas (es bedient sich der Notation des Semantically Enriched Extended Entity Relationship-Modells) welches der Bildung der notwendigen Relationenstrukturen zugrunde liegt.
Die Strukturen sind dabei so gehalten, daß sie für beliebige XML-Dokumente dienen können. Hierzu orientiert sich das Schema an den strukturellen und begrifflichen Gegebenheiten des XML Information Sets.



Zur hierarchieunabhängigen Identifikation der einzelnen verwalteten Elemente wird ein Surrogatattribut (ID) eingeführt, welches einen automatisch erzeugten Schlüssel in Form eines zeit- und raumunabhängigen Identifikators enthält.
Mit den in der Abbildung dargestellten Strukturen können wohlgeformte XML-Dokumente die ausschließlich Elemente, Attribute und textuellen Elementinhalt umfassen verwaltet werden. Aus Gründen der Übersichtlichkeit sind weitere XML-Strukturprimitive weggelassen.
Durch die aus dem konzeptuellen Schema erzeugten Relationenstrukturen können Teile der strukturellen Anforderungen an die Wohlgeformtheit eines Dokuments automatisiert durch das verwendete DBMS geprüft werden. Hierzu zählt die Anforderung der eindeutigen Benennung von Attributen innerhalb eines Elements sowie die streng hierarchische Schachtelung der XML-Elemente.

Beispiel 133 zeigt die Implementierung zur Erzeugung der Relationenstrukturen sowie zu deren Befüllung mit den Inhalten eines per Kommandozeilenargument übergebenen XML-Dokuments.

Beispiel 133: Speichern eines XML-Dokuments in einer relationalen Datenbank

```
(1)package de.jeckle.tinyXMLDB;
(2)
(3)import java.io.IOException;
(4)import java.sql.Connection;
(5)import java.sql.DriverManager;
(6)import java.sql.ResultSet;
(7)import java.sql.SQLException;
(8)import java.sql.Statement;
(9)import java.util.Stack;
(10)
(11)import javax.xml.parsers.ParserConfigurationException;
(12)import javax.xml.parsers.SAXParser;
(13)import javax.xml.parsers.SAXParserFactory;
(14)
(15)import org.doomdark.uuid.UUIDGenerator;
(16)import org.xml.sax.Attributes;
(17)import org.xml.sax.SAXException;
(18)import org.xml.sax.helpers.DefaultHandler;
(19)
(20)public class XML2DB extends DefaultHandler {
(21)    private Connection con = null;
(22)    private Stack elementStack;
(23)    private String inputFile;
(24)    private UUIDGenerator gen;
(25)
(26)    public static void main(String[] args)
(27)            throws
```

```
(28)                    ParserConfigurationException,
(29)                    SAXException,
(30)                    IOException,
(31)                    ClassNotFoundException,
(32)                    SQLException {
(33)            XML2DB xml2db = new XML2DB();
(34)            xml2db.con = xml2db.connectDB();
(35)            for (int i = 0; i < args.length; i++) {
(36)                    if (args[i].compareTo("-createDB") == 0) {
(37)                            xml2db.createDB(xml2db.con);
(38)                    }
(39)                    if (args[i].compareTo("-store") == 0) {
(40)                            xml2db.elementStack = new Stack();
(41)                            xml2db.inputFile = args[i + 1];
(42)                    }
(43)            }
(44)
(45)            xml2db.gen = UUIDGenerator.getInstance();
(46)
(47)            SAXParserFactory spf = SAXParserFactory.newInstance();
(48)            SAXParser sp = spf.newSAXParser();
(49)            System.out.print("storing document ...");
(50)            sp.parse(xml2db.inputFile, xml2db);
(51)            System.out.println("done");
(52)    }
(53)
(54)    private Connection connectDB()
(55)            throws ClassNotFoundException, SQLException {
(56)            Class.forName("com.mysql.jdbc.Driver");
(57)            return (Connection) DriverManager.getConnection(
(58)                    "jdbc:mysql://10.0.0.1/XML/",
(59)                    "mario",
(60)                    "thePassword");
(61)    }
(62)
(63)    private void createDB(Connection con) throws SQLException {
(64)            System.out.print("creating database ...");
(65)            Statement stmt = con.createStatement();
(66)
(67)            stmt.executeUpdate(
(68)                    "CREATE TABLE IF NOT EXISTS Element("
(69)                            + "ID VARCHAR(36) PRIMARY KEY,"
(70)                            + "Name VARCHAR(50) NOT NULL,"
(71)                            + "ParentElement VARCHAR(36),"
(72)                            + "SeqNo int);");
(73)            stmt.executeUpdate(
(74)                    "ALTER TABLE Element ADD INDEX ParentIdx (ParentElement);");
(75)            stmt.executeUpdate(
(76)                    "ALTER TABLE Element ADD CONSTRAINT ParentElementFK FOREIGN KEY
(ParentElement) REFERENCES Element(ID);");
(77)
(78)            stmt.executeUpdate(
(79)                    "CREATE TABLE IF NOT EXISTS Text("
(80)                            + "TextualContent Text NOT NULL,"
(81)                            + "ParentElement VARCHAR(36) NOT NULL,"
(82)                            + "SeqNo int NOT NULL,"
(83)                            + "PRIMARY KEY (SeqNo, ParentElement));");
(84)            stmt.executeUpdate(
(85)                    "ALTER TABLE Text ADD INDEX ParentIdx (ParentElement);");
(86)            stmt.executeUpdate(
(87)                    "ALTER TABLE Text ADD CONSTRAINT ParentElementFK FOREIGN KEY
(ParentElement) REFERENCES Element(ID);");
(88)
(89)            stmt.executeUpdate(
(90)                    "CREATE TABLE IF NOT EXISTS Attribute("
(91)                            + "Name VARCHAR(20) NOT NULL,"
(92)                            + "ParentElement VARCHAR(36) NOT NULL,"
(93)                            + "Value VARCHAR(50) NOT NULL,"
(94)                            + "PRIMARY KEY (Name, ParentElement));");
(95)            stmt.executeUpdate(
(96)                    "ALTER TABLE Attribute ADD INDEX ParentIdx (ParentElement);");
(97)            stmt.executeUpdate(
(98)                    "ALTER TABLE Attribute ADD CONSTRAINT ParentElementFK FOREIGN KEY
(ParentElement) REFERENCES Element(ID);");
(99)
(100)           System.out.println("done");
(101)   }
(102)
(103)   public void startDocument() {
(104)           elementStack.push(null);
(105)   }
(106)
```

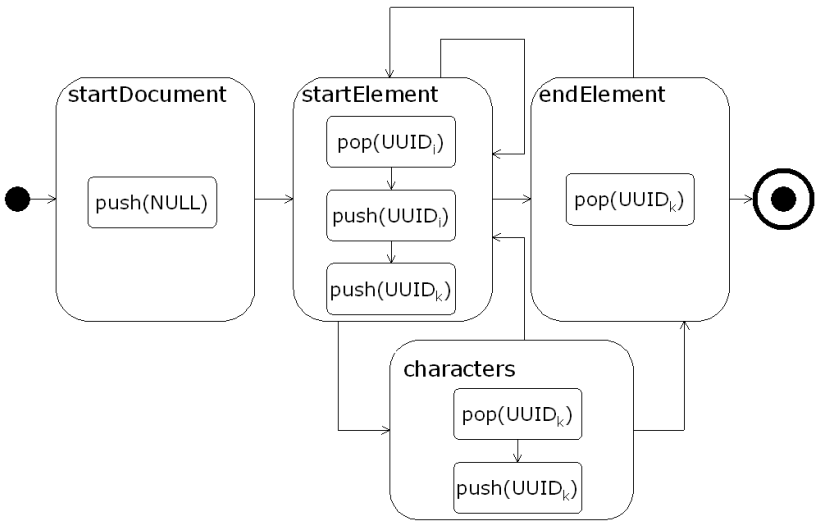
```
(107)   public void startElement(
(108)           String namespaceURI,
(109)           String localName,
(110)           String qName,
(111)           Attributes atts) {
(112)           String uuid = gen.generateTimeBasedUUID().toString();
(113)           String parentID = (String) elementStack.peek();
(114)
(115)           try {
(116)                   Statement stmt = con.createStatement();
(117)                   String st = null;
(118)                   ResultSet rs =
(119)                           stmt.executeQuery(
(120)                                   "(SELECT max(SeqNo) AS max FROM Element WHERE ParentElement=
\""
(121)                                           + parentID
(122)                                           + "\")"
(123)                                           + " UNION "
(124)                                           + "(SELECT max(SeqNo) AS max FROM Text WHERE
ParentElement=\""
(125)                                           + parentID
(126)                                           + "\")"
(127)                                           + "ORDER BY 1 DESC;");
(128)                   rs.first();
(129)                   int childSeq = rs.getInt("max");
(130)
(131)                   if (parentID == null) {
(132)                           st =
(133)                                   "INSERT INTO Element (ID, Name, SeqNo) VALUES("
(134)                                           + "\""
(135)                                           + uuid
(136)                                           + "\","
(137)                                           + "\""
(138)                                           + qName
(139)                                           + "\","
(140)                                           + ++childSeq
(141)                                           + ");";
(142)                   } else {
(143)                           st =
(144)                                   "INSERT INTO Element (ID, Name, SeqNo, ParentElement) VALUES
("
(145)                                           + "\""
(146)                                           + uuid
(147)                                           + "\",\""
(148)                                           + qName
(149)                                           + "\","
(150)                                           + ++childSeq
(151)                                           + ",\""
(152)                                           + parentID
(153)                                           + "\""
(154)                                           + ");";
(155)                   }
(156)                   stmt.executeUpdate(st);
(157)
(158)                   //storing attributes
(159)                   for (int i = 0; i < atts.getLength(); i++) {
(160)                           stmt.executeUpdate(
(161)                                   "INSERT INTO Attribute VALUES(\""
(162)                                           + atts.getQName(i)
(163)                                           + "\",\""
(164)                                           + uuid
(165)                                           + "\",\""
(166)                                           + atts.getValue(i)
(167)                                           + "\");");
(168)                   }
(169)
(170)           } catch (SQLException e) {
(171)                   e.printStackTrace();
(172)                   System.exit(1);
(173)           }
(174)
(175)           elementStack.push(uuid);
(176)   }
(177)
(178)   public void characters(char[] ch, int start, int length) {
(179)           String currentElement = (String) elementStack.peek();
(180)
(181)           String content = new String(ch, start, length);
(182)
(183)           try {
(184)                   Statement stmt = con.createStatement();
(185)                   String st = null;
```

```
(186)                   ResultSet rs =
(187)                           stmt.executeQuery(
(188)                                   "(SELECT max(SeqNo) AS max FROM Element WHERE ParentElement=
\""
(189)                                           + currentElement
(190)                                           + "\")"
(191)                                           + " UNION "
(192)                                           + "(SELECT max(SeqNo) AS max FROM Text WHERE
ParentElement=\""
(193)                                           + currentElement
(194)                                           + "\")"
(195)                                           + "ORDER BY 1 DESC;");
(196)                   rs.first();
(197)                   int childSeq = rs.getInt("max");
(198)
(199)                   st =
(200)                           "INSERT INTO Text VALUES("
(201)                                   + "\""
(202)                                   + content
(203)                                   + "\","
(204)                                   + "\""
(205)                                   + currentElement
(206)                                   + "\","
(207)                                   + ++childSeq
(208)                                   + ");";
(209)                   stmt.executeUpdate(st);
(210)           } catch (SQLException e) {
(211)                   e.printStackTrace();
(212)           }
(213)   }
(214)
(215)   public void endElement(
(216)           String namespaceURI,
(217)           String localName,
(218)           String qName) {
(219)           elementStack.pop();
(220)   }
(221)}
```

Download des Beispiels

Die Logik der Abbildung der XML-Strukturen und Inhalte stellt jedes Element des Dokuments durch einen Tupel der Tabelle `Element` dar. Dieser Tupel enthält eine eineindeutige Identifikation für das Element (`UUID`, seinen qualifizierten Namen, die Identifikation des Elternelements (`ParentElement`) und die Position (`SeqNo`) des aktuellen Elements innerhalb der Kindelemente des Elternelements.
XML-Attribute bedürfen keine eigenständigen Identifikation durch einen generischen Schlüssel, sondern können --- aufgrund ihrer Eindeutigkeit für das umgebende Elternelement --- direkt durch ihren Namen identifiziert werden. Zusätzlich hierzu wird der unstrukturierte Inhalt des Attributs als Zeichenkette abgelegt. Da die Auftrittsreichenfolge von Attributen spezifikationsgemäß nicht signifikant ist, wird diese nicht mitverwaltet.
Der Inhalt eines Elements kann sich im betrachteten Ausschnitt des Information Sets ausschließlich aus weiteren Elementen, den sog. Kindelementen, oder unstrukturiertem textuellem Inhalt zusammensetzen. Für Kindelemente werden keine zusätzlichen Verwaltungsstrukturen benötigt, da diese selbst wieder als vollständige Elemente dargestellt werden können. Zur Aufnahme der textuellen Anteile genügt die Ablage der Inhalte sowie der Reihenfolgeposition innerhalb der Sequenz der Kindelemente.

Die Realisierung bedient sich eines SAX-basierten Parsers, der jedes Ereignis des Typs `startElement` einen Eintrag in die Tabelle `Element` sowie im Bedarfsfalle zu `Attribute` hinzufügt. Die Erzeugung der Datentupel für die textuellen Inhalte geschieht im Rahmen der Verarbeitung des `characters`-Ereignisses.
Zur korrekten Behandlung der XML-Hierarchie wird ein Verarbeitungskeller eingeführt, die den einzelnen Ereignisbehandlungsroutinen `startElement` und `characters` die Identifikation des aktuell verarbeiteten Elements (d. h. des Elternelements der durch die beiden Behandlungsroutinen verarbeiteten XML-Elements) zur Verfügung stellt. Die Befüllung dieses Kellers geschieht durch die Behandlungsroutine `startElement` sowie `startDocument` für die initiale Ablage von `NULL` als Identifikation des Elternelement-losen Wurzelelements. Entfernt werden Kellerelemente ausschließlich durch die Behandlungsroutine `endElement`.
Das Aktivitätsdiagramm der Abbildung 59 zeigt die Interaktion der durch den Keller gekoppelten Ereignisbehandlungsroutinen.



Das Auslesen eines in die erzeugten Relationen aufgespaltenen Dokuments vollzieht sich als datenbankgestützter dynamischer Rekombinationprozeß gemäß dem durch Beispiel 134 dargestellten Beispielcode.

Beispiel 134: Erzeugen eines XML-Dokuments aus einer relationalen Datenbank

```
(1)package de.jeckle.tinyXMLDB;
(2)
(3)import java.sql.Connection;
(4)import java.sql.DriverManager;
(5)import java.sql.ResultSet;
(6)import java.sql.SQLException;
(7)import java.sql.Statement;
(8)
(9)public class DB2XML {
(10)    public static void main(String[] args)
(11)            throws ClassNotFoundException, SQLException {
(12)            Class.forName("com.mysql.jdbc.Driver");
(13)            Connection con =
(14)                    (Connection) DriverManager.getConnection(
(15)                            "jdbc:mysql://10.0.0.1/XML/",
(16)                            "mario",
(17)                            "thePassword");
(18)            Statement stmt = con.createStatement();
(19)            ResultSet rs =
(20)                    stmt.executeQuery(
(21)                            "SELECT ID FROM Element WHERE ParentElement IS NULL;");
(22)            rs.first();
(23)            printElement(rs.getString("ID"), con);
(24)    }
(25)    private static void printElement(String elementID, Connection con)
(26)            throws SQLException {
(27)            Statement stmt = con.createStatement();
(28)            ResultSet rs =
(29)                    stmt.executeQuery(
(30)                            "SELECT Name FROM Element WHERE ID=\"" + elementID + "\";");
(31)            rs.first();
(32)            String elementName = rs.getString("Name");
(33)
(34)            System.out.print("<" + elementName);
(35)            //process attributes
(36)            rs =
(37)                    stmt.executeQuery(
(38)                            "SELECT Name, Value FROM Attribute WHERE ParentElement=\""
(39)                                    + elementID
(40)                                    + "\";");
(41)            rs.beforeFirst();
(42)            while (rs.next()) {
(43)                    System.out.print(
(44)                            " "
(45)                                    + rs.getString("Name")
(46)                                    + "=\""
(47)                                    + rs.getString("Value")
(48)                                    + "\"");
(49)            }
(50)            System.out.print(">");
(51)            //process content
(52)            String st =
(53)                    "(SELECT SeqNo FROM Element WHERE ParentElement=\""
(54)                            + elementID
(55)                            + "\")"
(56)                            + "UNION"
(57)                            + "(SELECT SeqNo FROM Text WHERE "
```

```
(58)                            + "ParentElement=\""
(59)                            + elementID
(60)                            + "\")"
(61)                            + " ORDER BY 1 DESC;";
(62)            rs = stmt.executeQuery(st);
(63)            rs.first();
(64)
(65)            int noChilds = rs.getInt("SeqNo");
(66)
(67)            ResultSet crs[] = new ResultSet[noChilds];
(68)            for (int i = 1; i <= noChilds; i++) {
(69)                    crs[i - 1] =
(70)                            stmt.executeQuery(
(71)                                    "SELECT Name, ID FROM Element WHERE ParentElement=\""
(72)                                            + elementID
(73)                                            + "\" AND SeqNo="
(74)                                            + i
(75)                                            + ";");
(76)                    crs[i - 1].last();
(77)                    if (crs[i - 1].getRow() == 1) {
(78)                            //got an element
(79)                            printElement(crs[i - 1].getString("ID"), con);
(80)                    }
(81)                    crs[i - 1] =
(82)                            stmt.executeQuery(
(83)                                    "SELECT TextualContent FROM Text WHERE ParentElement=\""
(84)                                            + elementID
(85)                                            + "\" AND SeqNo ="
(86)                                            + i
(87)                                            + ";");
(88)                    crs[i - 1].last();
(89)                    if (crs[i - 1].getRow() == 1) {
(90)                            //got text child node
(91)                            System.out.print(crs[i - 1].getString("TextualContent"));
(92)                    }
(93)            }
(94)            System.out.print("</" + elementName + ">");
(95)    }
(96)}
```

Download des Beispiels

Diskussion der Vor- und Nachteile
Die Transformation der strukturellen Eigenschaften von XML in die Strukturen eines logischen Modells erweist sich als **konzeptionell leicht begehbarer Weg** um XML-Daten in bestehenden DBMS abzulegen, ohne diese für die Aufnahme des neuen Datentypen zu modifizieren. Gleichzeitig gestattet es die **Nutzung der vorhandenen Datenbanksysteme** auch für XML-Daten. Aus diesem Grund bieten viele Hersteller, insbesondere relationaler Systeme, inzwischen Module an, die die im Rahmen der Beispiele manuell vorgenommene Abbildung bereits datenbankseitig vornehmen und so den Abbildungsschritt automatisieren.
Die hierfür notwendigen **Abbildungsoperationen sind jedoch zeitintensiv**, so daß die Geschwindigkeit für den Zugriff auf aufgespaltene XML-Daten deutlich hinter dem auf native (d.h. beispielsweise relationale) zurückfällt. Dasselbe gilt für Ausleseschritte unter Anwendung von XPath oder XQuery. Dafür kann zumeist durch **Erweiterungen der nativen Anfragesprache** (zumeist ist dies SQL) vergleichsweise performant auf die Inhalte der früheren XML-Strukturen zugegriffen werden.
Maßnahmen zur Wahrung der strukturellen **Konsistenz** der verwalteten XML-Daten lassen sich teilweise bereits in Organisation der gewählten logische Repräsentation ergreifen und können zusätzlich im Rahmen des Abbildungsprozesses erfolgen.

Verwaltung von XML-Inhalten durch native XML-Datenbanken

Grundidee und Konzepte
Den im Vergleich zu den im vorhergehenden betrachteten Ansätzen jüngsten Beitrag zu den Speicherungsalternativen liefern sog. „XML-Datenbanken“, die sich das Ziel setzen XML-Daten „nativ“, d.h. ohne für den Anwender spür- und sichtbare Abbildung in logische Strukturen abzulegen.
Zwar ist dieser Ansatz durchaus konzeptionell diskussionswürdig, da er die Strukturen des logischen Modells XML auf die physische Speicherung zu übertragen sucht und damit die physische Datenunabhängigkeit verletzt, jedoch sprechen die Erfolge --- hauptsächlich hinsichtlich des Laufzeitverhaltens --- für diesen Speicherungstyp.

Beispiel
Bisher konnte sich für XML-Datenbanken noch kein Standard oder zumindest eine breite Übereinkunft hinsichtlich des Zugriffes oder der Administration herausbilden. Lediglich XQuery wird in breiten Teilen der Industrie als zukünftiger Standard einer Anfragesprache angesehen, der sich jedoch noch kaum in kommerziellen Produkten umgesetzt findet. Bis zur entgültigen Verabschiedung dieser W3C-Spezifikation werden vielfach noch XPath-basierenden proprietäre Anfragesprachen angeboten.
Das nachfolgende Beispiel beruht auf der open-source Implementierung Xindice, die im Rahmen des Apache XML-Projektes frei verfügbar ist.

Xindice organisiert die verwalteten Inhalte generell zweistufig. Die oberste Organisationseben bilden die sog. Dokumentsammlungen (*Collections*), welche die benannt abgelegten XML-Dokumente enthalten.

Zur Erzeugung von Sammlungen werden Administrationsrechte in Form von Ausführungsrechten für das Kommandozeilenwerkzeug `xindiceadmin` benötigt.
Der Aufruf `xindiceadmin ac -c /db -n myCol1` erzeugt eine Sammlung unter dem Namen `col1` in der Hierarchie der Dokumentsammlungen unterhalb des Astes `myCollection`.

Die Speicherung von Dokumenten in der Datenbank geschieht durch das Kommando `xindice` mit dem Parameter `ad` (Abkürzung für `add document`) und Angabe der Sammlung in die das Dokument aufgenommen werden soll (`-c`) , sowie des des Dokumentnamens (`-n`) und der Lokation des Quelldokuments im Dateisystem (`-f`).
So legt `xindice ad -c /db/myCol1 -f /home/mario/test.xml -n test` Das Dokument `test.xml` im Systemkatalog `/home/mario` in der Sammlung `/db/myCol1` unter dem Namen `test` ab.

Zur Entnahme eines abgelegten Dokuments steht der Parameter `rd` (Abkürzung für *retrieve for storage elsewhere*) zur Verfügung. Die Angabe von `xindiceadmin rd -c /db/myCol1 -n test` liefert das zuvor abgelegte Dokument als Resultat in die Standardausgebe.

Zur Anfrage auf Dokumentsammlungen, d.h. auf alle Dokumente einer Sammlung gleichzeitig, stellt das Werkzeug `xindice` den Parameter `xpath` bereit. Daher liefert die Anfrage `xindice xpath -c /db/myCol1 -q /` alle Wurzelelemente aller in der Sammlung `/db/myCol1` verwalteten Dokumente.

Diskussion der Vor- und Nachteile
XML-Datenbankmanagementsysteme offerieren einige Vorteile XML-spezifische Vorteile, wie die **native Unterstützung** einer auf XML-Inhalte abgestimmten Anfragesprache sowie auf diese Inhalte ausgerichtete Manipulations- und Verarbeitungsmechanismen wie XSLT-Transformationen, erfordern jedoch die **Investition** in ein neues DBMS, das im Betrieb zusätzlich zu den bestehenden Systemen betrieben werden muß.
Typischerweise erzielen XML-DBMS eine **höhere Zugriffsgeschwindigkeit** als Ansätze die XML-Daten in andere logische Strukturen überführen.
Die Verwendung von Systemen dieses Typs erfordert keine Kenntnis über den internen Aufbau und die physischen Ablageform der XML-Daten, ist jedoch in der Möglichkeit der Datenspeicherung ausschließlich auf diese Daten beschränkt, was durchaus **Zweifel an der langfristigen Zukunftsfähigkeit** dieser Systeme angebracht erscheinen läßt.

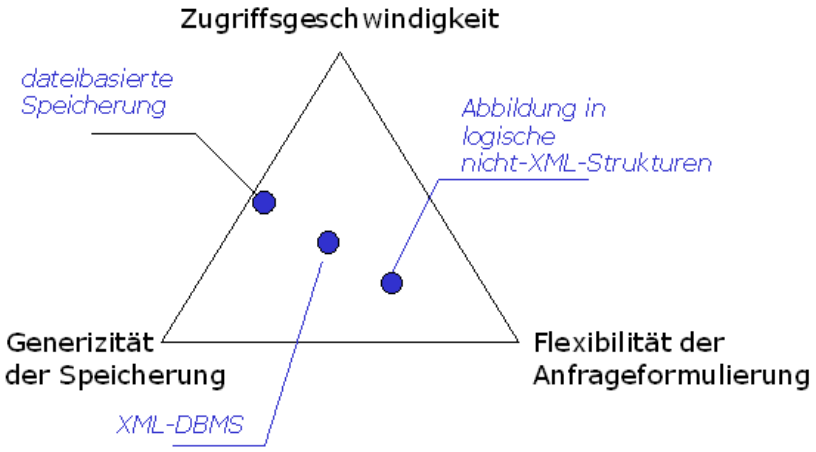
Vergleich der drei vorgestellten Konzepte

Der Vergleich der drei unterschiedlichen Speicherungs- und Zugriffsphilosophien offenbart, daß generell die Generizität der Speicherung, die mögliche erzielbare Flexibilität der Anfrageformulierung und die durch das System erreichbare Zugriffsgeschwindigkeit antagonistische Ziele darstellen.

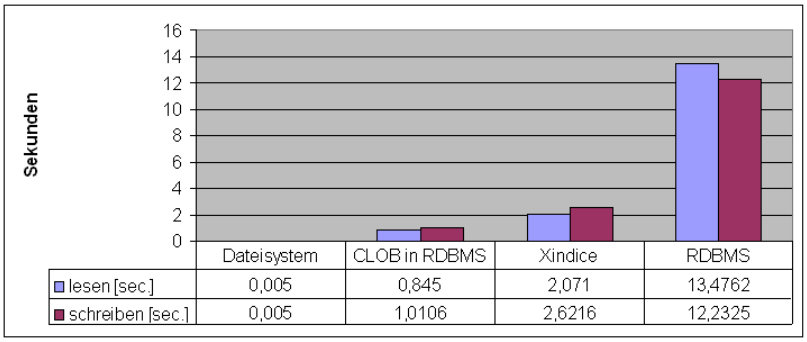
Im Detail streben die Ziele die gleichzeitige Erfüllung folgender Eigenschaften an:

- **Zugriffsgeschwindigkeit**: Die lesenden- und schreibenden Operationen auf dem in der Datenbank gehaltenen Datenbestand sollen idealerweise möglichst nahe an der ohne Verwendung eines Datenbanksystems erzielbaren Geschwindigkeit ablaufen.
- **Generizität der Speicherung**: Das Datenbanksystem soll in der Lage sein verschiedene XML-Formate in derselben Weise zu behandeln und abzulegen, ohne manuelle Eingriffe zu benötigen.
  Gleichzeitig sollen die erzeugten internen Strukturen so zukunftsfähig sein, das die Unterstützung neuer logischer Formate keine Änderungen an den existierenden Datenbeständen nach sich zieht.
- **Flexibilität der Anfrageformulierung**: Jedes einzelne Datenatom, d.h. jeder für den Benutzer potentiell informationstragender Bestandteil der verwalteten Daten, sollte einzeln abfragbar sein.
  Gleichzeitig sollte die Verwendung der Anfragesprache keinerlei Kenntnis über die interne Ablage der verwalteten Daten verlangen und durch die Datenbanksysteme verschiedener Hersteller unterstützt werden.

Abbildung 60 illustriert schematisch die quantitative Erfüllung dieser Ziele durch die drei vorgestellten Systemtypen. Hinsichtlich der Zugriffsgeschwindikeit erzielt die dateibasierte Speicherung zweifellos die besten Ergebnisse, da sie keinerlei Einschränkung hinsichtlich der unterstützten Datenformate trifft, jedoch gleichzeitig existierende Formate auch nicht strukturell ausbeuten kann. Als Resultat hiervon können Anfragen auf als Datei veralteten XML-Dokumenten nur durch zusätzliche Werkzeuge, die außerhalb des Dateisystems als Applikation zur Verfügung stehen, formuliert werden.
Durch den Ansatz XML in andere (zumeist relationale) logische Strukturen abzubilden verschiebst sich das Gewicht signifikant zu einer Begünstigung der Anfrageforderung. Üblicherweise werden für die gemäß dem logischen Modell des verwendeten DBMS zerlegte XML-Dokumente auch die Anfragesprachen des logischen Modells angeboten. Dieser Ansatz vernachlässigt jedoch die Forderung nach generischer Speicherung, da er eine explizite Transformation in die logischen Zielstrukturen erfordert. Durch diesen notwendigen Abbildungsschritt sind generell die Zugriffsgeschwindigkeit.
XML-Datenbanksysteme versuchen einen pragmatischen Kompromiß zwischen den drei widerstreitenden Zielen zu erreichen. Dies geschieht jedoch durch Einführung eines neuen DB-Systemtyps und neuer Anfragesprachen.



Abschließend soll als ein quantifizierbarer Vergleichsparameter die Geschwindigkeit lesender- und schreibender Operationen herausgegriffen werden. Die Graphik der Abbildung 61 stellt die an den Systemtypen dieses Kapitels gemessenen Werte bei der Speicherung und anschließenden Extraktion (und Ablage in einer Datei ohne vorherige Ausgabe am Bildschirm) der XML-Darstellung des Schauspiels *Macbeth* dar.



	Dateisystem	CLOB in RDBMS	Xindice	RDBMS
lesen [sec.]	0,005	0,845	2,071	13,4762
schreiben [sec.]	0,005	1,0106	2,6216	12,2325

Hervorstechend ist die auffallend niedrige (d.h. hoher Zeitbedarf) Zugriffsgeschwindigkeit der relationalen Abbildung aus den Beispielen 133 und 134. Diese erklärt sich einerseits durch die gewählte naive Implementierung, die auf zugriffsbeschleunigende datenbankseitige Optimierungen und entsprechende Applikationsseitige Mechanismen verzichtet aber gleichzeitig auch durch den inhärent zu leistendenen Abbildungsaufwand (insbesondere bei der Rekombination eines aufgespaltenen Dokuments während des lesenden Zugriffs), der auch durch Messungen an kommerziell verfügbaren Systemen gestützt wird.

Web-Referenzen 22: Weiterführende Links

- Viele weiterführende Informationen zum Thema
- Xindice, eine open-source XML-Datenbank
- Tamino, ein kommerzielles XML-Datenbankprodukt
- DB2, ein relationales DBMS mit XML-Erweiterungen
- Oracle, ein relationales DBMS mit XML-Erweiterungen

Lösungen zu den Übungsaufgaben

- **2a)** Liste aller `Nachnamen` unterhalb eines `Personen` Elements, in einem Dokument, in dem mindestens ein Element des Typs `Qualifikationsprofil` existiert
- **2b)** Alle ersten `Vornamen` eines `Personen`-Knotens unter `ProjektVerwaltung`, von `Personen`, die über einen zweiten `Vornamen` verfügen.
- **2c)** Alle `Nachnamen` von `Personen`, deren `PersID`-Attribut den Wert „`Pers01`“ enthält.
- **2d)** `//Person[Nachname.='Obermüller']`
- **2e)** `//Nachname[following-sibling::Qualifikationsprofil[count(//Qualifikation)>1]]`
- **2f)** `//Person[attribute::PersID=//Projekt/attribute::Projektleiter]/Nachname`
- **XSLT-Datei zu Übung vier**
- **5a)** Liste aller `Nachnamen` unterhalb eines `Personen`-Elements, in einem Dokument, in dem mindestens ein Element des Typs `Qualifikationsprofil` existiert.
- **5b)** Alle ersten `Vornamen` eines `Personen`-Knotens unterhalb `ProjektVerwaltung`, von `Personen`, die über einen zweiten `Vornamen` verfügen.
- **5c)** Alle `Nachnamen` von `Personen`, deren `PersID`-Attribut den Wert „`Pers01`“ enthält.
- **5d)**

```
FOR $x IN //Person W $x/Nachname = 'Obermüller' RETURN $x
```

- **5e)**

```
FOR $x IN //Person WHERE count($x//Qualifikation) > 1 RETURN $x/Nachname
```

- **5f)**

```
$x IN //Person     $y IN //Projekt WHERE $x/@PersID => $y/@Projektleiter
RETURN $x/Nachname
```

Definitionsverzeichnis

Schlagwortverzeichnis

Abbildungsverzeichnis

Verzeichnis der Beispiele

---

java.util

Class Calendar

java.lang.Object
  └ **java.util.Calendar**

All Implemented Interfaces:

Cloneable, Serializable

Direct Known Subclasses:

GregorianCalendar

---

public abstract class **Calendar**
extends Object
implements Serializable, Cloneable

`Calendar` is an abstract base class for converting between a `Date` object and a set of integer fields such as `YEAR`, `MONTH`, `DAY`, `HOUR`, and so on. (A `Date` object represents a specific instant in time with millisecond precision. See `Date` for information about the `Date` class.)

Subclasses of `Calendar` interpret a `Date` according to the rules of a specific calendar system. The platform provides one concrete subclass of `Calendar`: `GregorianCalendar`. Future subclasses could represent the various types of lunar calendars in use in many parts of the world.

Like other locale-sensitive classes, `Calendar` provides a class method, `getInstance`, for getting a generally useful object of this type. `Calendar`'s `getInstance` method returns a `Calendar` object whose time fields have been initialized with the current date and time:

```
Calendar rightNow = Calendar.getInstance();
```

A `Calendar` object can produce all the time field values needed to implement the date-time formatting for a particular language and calendar style (for example, Japanese-Gregorian, Japanese-Traditional). `Calendar` defines the range of values returned by certain fields, as well as their meaning. For example, the first month of the year has value `MONTH` == `JANUARY` for all calendars. Other values are defined by the concrete subclass, such as `ERA` and `YEAR`. See individual field

documentation and subclass documentation for details.

When a `Calendar` is *lenient*, it accepts a wider range of field values than it produces. For example, a lenient `GregorianCalendar` interprets `MONTH` == `JANUARY`, `DAY_OF_MONTH` == 32 as February 1. A non-lenient `GregorianCalendar` throws an exception when given out-of-range field settings. When calendars recompute field values for return by `get()`, they normalize them. For example, a `GregorianCalendar` always produces `DAY_OF_MONTH` values between 1 and the length of the month.

`Calendar` defines a locale-specific seven day week using two parameters: the first day of the week and the minimal days in first week (from 1 to 7). These numbers are taken from the locale resource data when a `Calendar` is constructed. They may also be specified explicitly through the API.

When setting or getting the `WEEK_OF_MONTH` or `WEEK_OF_YEAR` fields, `Calendar` must determine the first week of the month or year as a reference point. The first week of a month or year is defined as the earliest seven day period beginning on `getFirstDayOfWeek()` and containing at least `getMinimalDaysInFirstWeek()` days of that month or year. Weeks numbered ..., -1, 0 precede the first week; weeks numbered 2, 3,... follow it. Note that the normalized numbering returned by `get()` may be different. For example, a specific `Calendar` subclass may designate the week before week 1 of a year as week *n* of the previous year.

When computing a `Date` from time fields, two special circumstances may arise: there may be insufficient information to compute the `Date` (such as only year and month but no day in the month), or there may be inconsistent information (such as "Tuesday, July 15, 1996" -- July 15, 1996 is actually a Monday).

Insufficient information. The calendar will use default information to specify the missing fields. This may vary by calendar; for the Gregorian calendar, the default for a field is the same as that of the start of the epoch: i.e., YEAR = 1970, MONTH = JANUARY, DATE = 1, etc.

Inconsistent information. If fields conflict, the calendar will give preference to fields set more recently. For example, when determining the day, the calendar will look for one of the following combinations of fields. The most recent combination, as determined by the most recently set single field, will be used.

```
MONTH + DAY_OF_MONTH
MONTH + WEEK_OF_MONTH + DAY_OF_WEEK
MONTH + DAY_OF_WEEK_IN_MONTH + DAY_OF_WEEK
DAY_OF_YEAR
DAY_OF_WEEK + WEEK_OF_YEAR
```

For the time of day:

```
HOUR_OF_DAY
```

```
AM_PM + HOUR
```

Note: for some non-Gregorian calendars, different fields may be necessary for complete disambiguation. For example, a full specification of the historical Arabic astronomical calendar requires year, month, day-of-month *and* day-of-week in some cases.

Note: There are certain possible ambiguities in interpretation of certain singular times, which are resolved in the following ways:

1. 23:59 is the last minute of the day and 00:00 is the first minute of the next day. Thus, 23:59 on Dec 31, 1999 < 00:00 on Jan 1, 2000 < 00:01 on Jan 1, 2000.
2. Although historically not precise, midnight also belongs to "am", and noon belongs to "pm", so on the same day, 12:00 am (midnight) < 12:01 am, and 12:00 pm (noon) < 12:01 pm

The date or time format strings are not part of the definition of a calendar, as those must be modifiable or overridable by the user at runtime. Use `DateFormat` to format dates.

Field manipulation methods

`Calendar` fields can be changed using three methods: `set()`, `add()`, and `roll()`.

`set(f, value)` changes field `f` to `value`. In addition, it sets an internal member variable to indicate that field `f` has been changed. Although field `f` is changed immediately, the calendar's milliseconds is not recomputed until the next call to `get()`, `getTime()`, or `getTimeInMillis()` is made. Thus, multiple calls to `set()` do not trigger multiple, unnecessary computations. As a result of changing a field using `set()`, other fields may also change, depending on the field, the field value, and the calendar system. In addition, `get(f)` will not necessarily return `value` after the fields have been recomputed. The specifics are determined by the concrete calendar class.

Example: Consider a `GregorianCalendar` originally set to August 31, 1999. Calling `set(Calendar.MONTH, Calendar.SEPTEMBER)` sets the calendar to September 31, 1999. This is a temporary internal representation that resolves to October 1, 1999 if `getTime()`is then called. However, a call to `set(Calendar.DAY_OF_MONTH, 30)` before the call to `getTime()` sets the calendar to September 30, 1999, since no recomputation occurs after `set()` itself.

`add(f, delta)` adds `delta` to field `f`. This is equivalent to calling `set(f, get(f) + delta)` with two adjustments:

> **Add rule 1**. The value of field `f` after the call minus the value of field `f` before the call is `delta`, modulo any overflow that has occurred in field `f`. Overflow occurs when a field value exceeds its range and, as a result, the next larger field is incremented or decremented and the field value is adjusted back into its range.
>
> **Add rule 2**. If a smaller field is expected to be invariant, but it is impossible for it to

be equal to its prior value because of changes in its minimum or maximum after field `f` is changed, then its value is adjusted to be as close as possible to its expected value. A smaller field represents a smaller unit of time. `HOUR` is a smaller field than `DAY_OF_MONTH`. No adjustment is made to smaller fields that are not expected to be invariant. The calendar system determines what fields are expected to be invariant.

In addition, unlike `set()`, `add()` forces an immediate recomputation of the calendar's milliseconds and all fields.

Example: Consider a `GregorianCalendar` originally set to August 31, 1999. Calling `add(Calendar.MONTH, 13)` sets the calendar to September 30, 2000. **Add rule 1** sets the `MONTH` field to September, since adding 13 months to August gives September of the next year. Since `DAY_OF_MONTH` cannot be 31 in September in a `GregorianCalendar`, **add rule 2** sets the `DAY_OF_MONTH` to 30, the closest possible value. Although it is a smaller field, `DAY_OF_WEEK` is not adjusted by rule 2, since it is expected to change when the month changes in a `GregorianCalendar`.

`roll(f, delta)` adds `delta` to field `f` without changing larger fields. This is equivalent to calling `add(f, delta)` with the following adjustment:

Roll rule. Larger fields are unchanged after the call. A larger field represents a larger unit of time. `DAY_OF_MONTH` is a larger field than `HOUR`.

Example: See `GregorianCalendar.roll(int, int)`.

Usage model. To motivate the behavior of `add()` and `roll()`, consider a user interface component with increment and decrement buttons for the month, day, and year, and an underlying `GregorianCalendar`. If the interface reads January 31, 1999 and the user presses the month increment button, what should it read? If the underlying implementation uses `set()`, it might read March 3, 1999. A better result would be February 28, 1999. Furthermore, if the user presses the month increment button again, it should read March 31, 1999, not March 28, 1999. By saving the original date and using either `add()` or `roll()`, depending on whether larger fields should be affected, the user interface can behave as most users will intuitively expect.

Since:

JDK1.1

See Also:

`Date`, `GregorianCalendar`, `TimeZone`, `DateFormat`, Serialized Form

Field Summary

static int	**AM** Value of the AM_PM field indicating the period of the day from midnight to just before noon.
static int	**AM_PM** Field number for get and set indicating whether the HOUR is before or after noon.
static int	**APRIL** Value of the MONTH field indicating the fourth month of the year.
protected boolean	**areFieldsSet** True if fields[] are in sync with the currently set time.
static int	**AUGUST** Value of the MONTH field indicating the eighth month of the year.
static int	**DATE** Field number for get and set indicating the day of the month.
static int	**DAY_OF_MONTH** Field number for get and set indicating the day of the month.
static int	**DAY_OF_WEEK** Field number for get and set indicating the day of the week.
static int	**DAY_OF_WEEK_IN_MONTH** Field number for get and set indicating the ordinal number of the day of the week within the current month.
static int	**DAY_OF_YEAR** Field number for get and set indicating the day number within the current year.
static int	**DECEMBER** Value of the MONTH field indicating the twelfth month of the year.
static int	**DST_OFFSET** Field number for get and set indicating the daylight savings offset in milliseconds.
static int	**ERA** Field number for get and set indicating the era, e.g., AD or BC in the Julian calendar.
static int	**FEBRUARY** Value of the MONTH field indicating the second month of the year.
static int	**FIELD_COUNT** The number of distinct fields recognized by get and set.

protected int[]	**fields** The field values for the currently set time for this calendar.
static int	**FRIDAY** Value of the DAY_OF_WEEK field indicating Friday.
static int	**HOUR** Field number for get and set indicating the hour of the morning or afternoon.
static int	**HOUR_OF_DAY** Field number for get and set indicating the hour of the day.
protected boolean[]	**isSet** The flags which tell if a specified time field for the calendar is set.
protected boolean	**isTimeSet** True if then the value of time is valid.
static int	**JANUARY** Value of the MONTH field indicating the first month of the year.
static int	**JULY** Value of the MONTH field indicating the seventh month of the year.
static int	**JUNE** Value of the MONTH field indicating the sixth month of the year.
static int	**MARCH** Value of the MONTH field indicating the third month of the year.
static int	**MAY** Value of the MONTH field indicating the fifth month of the year.
static int	**MILLISECOND** Field number for get and set indicating the millisecond within the second.
static int	**MINUTE** Field number for get and set indicating the minute within the hour.
static int	**MONDAY** Value of the DAY_OF_WEEK field indicating Monday.
static int	**MONTH** Field number for get and set indicating the month.
static int	**NOVEMBER** Value of the MONTH field indicating the eleventh month of the year.
static int	**OCTOBER** Value of the MONTH field indicating the tenth month of the year.

static int	**PM** Value of the AM_PM field indicating the period of the day from noon to just before midnight.
static int	**SATURDAY** Value of the DAY_OF_WEEK field indicating Saturday.
static int	**SECOND** Field number for get and set indicating the second within the minute.
static int	**SEPTEMBER** Value of the MONTH field indicating the ninth month of the year.
static int	**SUNDAY** Value of the DAY_OF_WEEK field indicating Sunday.
static int	**THURSDAY** Value of the DAY_OF_WEEK field indicating Thursday.
protected long	**time** The currently set time for this calendar, expressed in milliseconds after January 1, 1970, 0:00:00 GMT.
static int	**TUESDAY** Value of the DAY_OF_WEEK field indicating Tuesday.
static int	**UNDECIMBER** Value of the MONTH field indicating the thirteenth month of the year.
static int	**WEDNESDAY** Value of the DAY_OF_WEEK field indicating Wednesday.
static int	**WEEK_OF_MONTH** Field number for get and set indicating the week number within the current month.
static int	**WEEK_OF_YEAR** Field number for get and set indicating the week number within the current year.
static int	**YEAR** Field number for get and set indicating the year.
static int	**ZONE_OFFSET** Field number for get and set indicating the raw offset from GMT in milliseconds.

Constructor Summary

protected	**Calendar**() Constructs a Calendar with the default time zone and locale.
protected	**Calendar**(TimeZone zone, Locale aLocale) Constructs a calendar with the specified time zone and locale.

Method Summary

abstract void	**add**(int field, int amount) Date Arithmetic function.
boolean	**after**(Object when) Compares the time field records.
boolean	**before**(Object when) Compares the time field records.
void	**clear**() Clears the values of all the time fields.
void	**clear**(int field) Clears the value in the given time field.
Object	**clone**() Overrides Cloneable
protected void	**complete**() Fills in any unset fields in the time field list.
protected abstract void	**computeFields**() Converts the current millisecond time value `time` to field values in `fields[]`.
protected abstract void	**computeTime**() Converts the current field values in `fields[]` to the millisecond time value `time`.
boolean	**equals**(Object obj) Compares this calendar to the specified object.
int	**get**(int field) Gets the value for a given time field.
int	**getActualMaximum**(int field) Return the maximum value that this field could have, given the current date.
int	**getActualMinimum**(int field) Return the minimum value that this field could have, given the current date.

static Locale[]	**getAvailableLocales**() Gets the list of locales for which Calendars are installed.
int	**getFirstDayOfWeek**() Gets what the first day of the week is; e.g., Sunday in US, Monday in France.
abstract int	**getGreatestMinimum**(int field) Gets the highest minimum value for the given field if varies.
static Calendar	**getInstance**() Gets a calendar using the default time zone and locale.
static Calendar	**getInstance**(Locale aLocale) Gets a calendar using the default time zone and specified locale.
static Calendar	**getInstance**(TimeZone zone) Gets a calendar using the specified time zone and default locale.
static Calendar	**getInstance**(TimeZone zone, Locale aLocale) Gets a calendar with the specified time zone and locale.
abstract int	**getLeastMaximum**(int field) Gets the lowest maximum value for the given field if varies.
abstract int	**getMaximum**(int field) Gets the maximum value for the given time field.
int	**getMinimalDaysInFirstWeek**() Gets what the minimal days required in the first week of the year are; e. g., if the first week is defined as one that contains the first day of the first month of a year, getMinimalDaysInFirstWeek returns 1.
abstract int	**getMinimum**(int field) Gets the minimum value for the given time field.
Date	**getTime**() Gets this Calendar's current time.
long	**getTimeInMillis**() Gets this Calendar's current time as a long.
TimeZone	**getTimeZone**() Gets the time zone.
int	**hashCode**() Returns a hash code for this calendar.
protected int	**internalGet**(int field) Gets the value for a given time field.

boolean	**isLenient**() Tell whether date/time interpretation is to be lenient.
boolean	**isSet**(int field) Determines if the given time field has a value set.
abstract void	**roll**(int field, boolean up) Time Field Rolling function.
void	**roll**(int field, int amount) Time Field Rolling function.
void	**set**(int field, int value) Sets the time field with the given value.
void	**set**(int year, int month, int date) Sets the values for the fields year, month, and date.
void	**set**(int year, int month, int date, int hour, int minute) Sets the values for the fields year, month, date, hour, and minute.
void	**set**(int year, int month, int date, int hour, int minute, int second) Sets the values for the fields year, month, date, hour, minute, and second.
void	**setFirstDayOfWeek**(int value) Sets what the first day of the week is; e.g., Sunday in US, Monday in France.
void	**setLenient**(boolean lenient) Specify whether or not date/time interpretation is to be lenient.
void	**setMinimalDaysInFirstWeek**(int value) Sets what the minimal days required in the first week of the year are; For example, if the first week is defined as one that contains the first day of the first month of a year, call the method with value 1.
void	**setTime**(Date date) Sets this Calendar's current time with the given Date.
void	**setTimeInMillis**(long millis) Sets this Calendar's current time from the given long value.
void	**setTimeZone**(TimeZone value) Sets the time zone with the given time zone value.
String	**toString**() Return a string representation of this calendar.

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

ERA

```
public static final int ERA
```

Field number for `get` and `set` indicating the era, e.g., AD or BC in the Julian calendar. This is a calendar-specific value; see subclass documentation.

See Also:
`GregorianCalendar.AD`, `GregorianCalendar.BC`, Constant Field Values

YEAR

```
public static final int YEAR
```

Field number for `get` and `set` indicating the year. This is a calendar-specific value; see subclass documentation.

See Also:
Constant Field Values

MONTH

```
public static final int MONTH
```

Field number for `get` and `set` indicating the month. This is a calendar-specific value. The first month of the year is `JANUARY` which is 0; the last depends on the number of months in a year.

See Also:
`JANUARY`, `FEBRUARY`, `MARCH`, `APRIL`, `MAY`, `JUNE`, `JULY`, `AUGUST`, `SEPTEMBER`, `OCTOBER`, `NOVEMBER`, `DECEMBER`, `UNDECIMBER`, Constant Field Values

---

WEEK_OF_YEAR

```
public static final int WEEK_OF_YEAR
```

Field number for `get` and `set` indicating the week number within the current year. The first week of the year, as defined by `getFirstDayOfWeek()` and `getMinimalDaysInFirstWeek()`, has value 1. Subclasses define the value of `WEEK_OF_YEAR` for days before the first week of the year.

See Also:

`getFirstDayOfWeek()`, `getMinimalDaysInFirstWeek()`, Constant Field Values

WEEK_OF_MONTH

```
public static final int WEEK_OF_MONTH
```

Field number for `get` and `set` indicating the week number within the current month. The first week of the month, as defined by `getFirstDayOfWeek()` and `getMinimalDaysInFirstWeek()`, has value 1. Subclasses define the value of `WEEK_OF_MONTH` for days before the first week of the month.

See Also:

`getFirstDayOfWeek()`, `getMinimalDaysInFirstWeek()`, Constant Field Values

DATE

```
public static final int DATE
```

Field number for `get` and `set` indicating the day of the month. This is a synonym for `DAY_OF_MONTH`. The first day of the month has value 1.

See Also:

`DAY_OF_MONTH`, Constant Field Values

DAY_OF_MONTH

```
public static final int DAY_OF_MONTH
```

Field number for `get` and `set` indicating the day of the month. This is a synonym for `DATE`. The first day of the month has value 1.

See Also:
`DATE`, Constant Field Values

DAY_OF_YEAR

```
public static final int DAY_OF_YEAR
```

Field number for `get` and `set` indicating the day number within the current year. The first day of the year has value 1.

See Also:
Constant Field Values

DAY_OF_WEEK

```
public static final int DAY_OF_WEEK
```

Field number for `get` and `set` indicating the day of the week. This field takes values `SUNDAY`, `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY`, and `SATURDAY`.

See Also:
`SUNDAY`, `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY`, `SATURDAY`, Constant Field Values

DAY_OF_WEEK_IN_MONTH

```
public static final int DAY_OF_WEEK_IN_MONTH
```

Field number for `get` and `set` indicating the ordinal number of the day of the week within the current month. Together with the `DAY_OF_WEEK` field, this uniquely specifies a day within a month. Unlike `WEEK_OF_MONTH` and `WEEK_OF_YEAR`, this field's value does *not* depend on `getFirstDayOfWeek()` or `getMinimalDaysInFirstWeek()`. `DAY_OF_MONTH 1` through `7` always correspond to `DAY_OF_WEEK_IN_MONTH 1`; `8` through `14` correspond to `DAY_OF_WEEK_IN_MONTH 2`, and so on. `DAY_OF_WEEK_IN_MONTH 0` indicates the week before `DAY_OF_WEEK_IN_MONTH 1`. Negative values count back from the end of the month, so the last Sunday of a month is specified as `DAY_OF_WEEK = SUNDAY, DAY_OF_WEEK_IN_MONTH = -1`. Because negative values count backward they will usually be aligned differently within the month than positive values. For example, if a month has 31 days, `DAY_OF_WEEK_IN_MONTH -1` will overlap `DAY_OF_WEEK_IN_MONTH 5` and the end of `4`.

See Also:

`DAY_OF_WEEK`, `WEEK_OF_MONTH`, Constant Field Values

AM_PM

```
public static final int AM_PM
```

Field number for `get` and `set` indicating whether the `HOUR` is before or after noon. E.g., at 10:04:15.250 PM the `AM_PM` is `PM`.

See Also:

`AM`, `PM`, `HOUR`, Constant Field Values

HOUR

```
public static final int HOUR
```

Field number for `get` and `set` indicating the hour of the morning or afternoon. `HOUR` is used for the 12-hour clock. E.g., at 10:04:15.250 PM the `HOUR` is 10.

See Also:

`AM_PM`, `HOUR_OF_DAY`, Constant Field Values

HOUR_OF_DAY

`public static final int` **`HOUR_OF_DAY`**

Field number for `get` and `set` indicating the hour of the day. `HOUR_OF_DAY` is used for the 24-hour clock. E.g., at 10:04:15.250 PM the `HOUR_OF_DAY` is 22.

See Also:
`HOUR`, Constant Field Values

MINUTE

`public static final int` **`MINUTE`**

Field number for `get` and `set` indicating the minute within the hour. E.g., at 10:04:15.250 PM the `MINUTE` is 4.

See Also:
Constant Field Values

SECOND

`public static final int` **`SECOND`**

Field number for `get` and `set` indicating the second within the minute. E.g., at 10:04:15.250 PM the `SECOND` is 15.

See Also:
Constant Field Values

MILLISECOND

`public static final int` **`MILLISECOND`**

Field number for `get` and `set` indicating the millisecond within the second. E.g., at 10:04:15.250 PM the `MILLISECOND` is 250.

See Also:
Constant Field Values

ZONE_OFFSET

```
public static final int ZONE_OFFSET
```

Field number for `get` and `set` indicating the raw offset from GMT in milliseconds.

This field reflects the correct GMT offset value of the time zone of this `Calendar` if the `TimeZone` implementation subclass supports historical GMT offset changes.

See Also:

Constant Field Values

DST_OFFSET

```
public static final int DST_OFFSET
```

Field number for `get` and `set` indicating the daylight savings offset in milliseconds.

This field reflects the correct daylight saving offset value of the time zone of this `Calendar` if the `TimeZone` implementation subclass supports historical Daylight Saving Time schedule changes.

See Also:

Constant Field Values

FIELD_COUNT

```
public static final int FIELD_COUNT
```

The number of distinct fields recognized by `get` and `set`. Field numbers range from `0..FIELD_COUNT-1`.

See Also:

Constant Field Values

SUNDAY

```
public static final int SUNDAY
```

Value of the DAY_OF_WEEK field indicating Sunday.

See Also:
Constant Field Values

MONDAY

```
public static final int MONDAY
```

Value of the DAY_OF_WEEK field indicating Monday.

See Also:
Constant Field Values

TUESDAY

```
public static final int TUESDAY
```

Value of the DAY_OF_WEEK field indicating Tuesday.

See Also:
Constant Field Values

WEDNESDAY

```
public static final int WEDNESDAY
```

Value of the DAY_OF_WEEK field indicating Wednesday.

See Also:
Constant Field Values

THURSDAY

```
public static final int THURSDAY
```

Value of the DAY_OF_WEEK field indicating Thursday.

See Also:

Constant Field Values

FRIDAY

```
public static final int FRIDAY
```

Value of the DAY_OF_WEEK field indicating Friday.

See Also:

Constant Field Values

SATURDAY

```
public static final int SATURDAY
```

Value of the DAY_OF_WEEK field indicating Saturday.

See Also:

Constant Field Values

JANUARY

```
public static final int JANUARY
```

Value of the MONTH field indicating the first month of the year.

See Also:

Constant Field Values

FEBRUARY

```
public static final int FEBRUARY
```

Value of the MONTH field indicating the second month of the year.

See Also:
Constant Field Values

MARCH

```
public static final int MARCH
```

Value of the MONTH field indicating the third month of the year.

See Also:
Constant Field Values

APRIL

```
public static final int APRIL
```

Value of the MONTH field indicating the fourth month of the year.

See Also:
Constant Field Values

MAY

```
public static final int MAY
```

Value of the MONTH field indicating the fifth month of the year.

See Also:

Constant Field Values

JUNE

```
public static final int JUNE
```

Value of the MONTH field indicating the sixth month of the year.

See Also:
Constant Field Values

JULY

```
public static final int JULY
```

Value of the MONTH field indicating the seventh month of the year.

See Also:
Constant Field Values

AUGUST

```
public static final int AUGUST
```

Value of the MONTH field indicating the eighth month of the year.

See Also:
Constant Field Values

SEPTEMBER

```
public static final int SEPTEMBER
```

Value of the MONTH field indicating the ninth month of the year.

See Also:
Constant Field Values

OCTOBER

```
public static final int OCTOBER
```

Value of the MONTH field indicating the tenth month of the year.

See Also:
Constant Field Values

NOVEMBER

```
public static final int NOVEMBER
```

Value of the MONTH field indicating the eleventh month of the year.

See Also:
Constant Field Values

DECEMBER

```
public static final int DECEMBER
```

Value of the MONTH field indicating the twelfth month of the year.

See Also:
Constant Field Values

UNDECIMBER

```
public static final int UNDECIMBER
```

Value of the MONTH field indicating the thirteenth month of the year. Although

`GregorianCalendar` does not use this value, lunar calendars do.

See Also:
Constant Field Values

AM

```
public static final int AM
```

Value of the `AM_PM` field indicating the period of the day from midnight to just before noon.

See Also:
Constant Field Values

PM

```
public static final int PM
```

Value of the `AM_PM` field indicating the period of the day from noon to just before midnight.

See Also:
Constant Field Values

---

fields

```
protected int[] fields
```

The field values for the currently set time for this calendar. This is an array of `FIELD_COUNT` integers, with index values `ERA` through `DST_OFFSET`.

---

isSet

```
protected boolean[] isSet
```

The flags which tell if a specified time field for the calendar is set. A new object has no fields

set. After the first call to a method which generates the fields, they all remain set after that. This is an array of `FIELD_COUNT` booleans, with index values `ERA` through `DST_OFFSET`.

---

time

```
protected long time
```

The currently set time for this calendar, expressed in milliseconds after January 1, 1970, 0:00:00 GMT.

See Also:
`isTimeSet`

---

isTimeSet

```
protected boolean isTimeSet
```

True if then the value of `time` is valid. The time is made invalid by a change to an item of `field[]`.

See Also:
`time`

---

areFieldsSet

```
protected boolean areFieldsSet
```

True if `fields[]` are in sync with the currently set time. If false, then the next attempt to get the value of a field will force a recomputation of all fields from the current value of `time`.

Constructor Detail

Calendar

```
protected Calendar()
```

Constructs a Calendar with the default time zone and locale.

See Also:

`TimeZone.getDefault()`

Calendar

```
protected Calendar(TimeZone zone,
                   Locale aLocale)
```

Constructs a calendar with the specified time zone and locale.

Parameters:

`zone` - the time zone to use
`aLocale` - the locale for the week data

Method Detail

getInstance

```
public static Calendar getInstance()
```

Gets a calendar using the default time zone and locale. The `Calendar` returned is based on the current time in the default time zone with the default locale.

Returns:

a Calendar.

---

getInstance

```
public static Calendar getInstance(TimeZone zone)
```

Gets a calendar using the specified time zone and default locale. The `Calendar` returned is based on the current time in the given time zone with the default locale.

Parameters:

`zone` - the time zone to use

Returns:

a Calendar.

---

getInstance

```
public static Calendar getInstance(Locale aLocale)
```

Gets a calendar using the default time zone and specified locale. The `Calendar` returned is based on the current time in the default time zone with the given locale.

Parameters:
`aLocale` - the locale for the week data
Returns:
a Calendar.

---

getInstance

```
public static Calendar getInstance(TimeZone zone,
                                   Locale aLocale)
```

Gets a calendar with the specified time zone and locale. The `Calendar` returned is based on the current time in the given time zone with the given locale.

Parameters:
`zone` - the time zone to use
`aLocale` - the locale for the week data
Returns:
a Calendar.

---

getAvailableLocales

```
public static Locale[] getAvailableLocales()
```

Gets the list of locales for which Calendars are installed.

Returns:
the list of locales for which Calendars are installed.

---

computeTime

```
protected abstract void computeTime()
```

Converts the current field values in `fields[]` to the millisecond time value `time`.

---

computeFields

```
protected abstract void computeFields()
```

Converts the current millisecond time value `time` to field values in `fields[]`. This allows you to sync up the time field values with a new time that is set for the calendar. The time is *not* recomputed first; to recompute the time, then the fields, call the `complete` method.

See Also:
`complete()`

---

getTime

```
public final Date getTime()
```

Gets this Calendar's current time.

Returns:
the current time.
See Also:
`setTime(java.util.Date)`, `getTimeInMillis()`

---

setTime

```
public final void setTime(Date date)
```

Sets this Calendar's current time with the given Date.

Note: Calling `setTime()` with `Date(Long.MAX_VALUE)` or `Date(Long.`

MIN_VALUE) may yield incorrect field values from get().

Parameters:
date - the given Date.
See Also:
getTime(), setTimeInMillis(long)

---

getTimeInMillis

```
public long getTimeInMillis()
```

Gets this Calendar's current time as a long.

Returns:
the current time as UTC milliseconds from the epoch.
See Also:
getTime(), setTimeInMillis(long)

---

setTimeInMillis

```
public void setTimeInMillis(long millis)
```

Sets this Calendar's current time from the given long value.

Parameters:
millis - the new time in UTC milliseconds from the epoch.
See Also:
setTime(java.util.Date), getTimeInMillis()

---

get

```
public int get(int field)
```

Gets the value for a given time field.

Parameters:
field - the given time field.

Returns:
the value for the given time field.
Throws:
`ArrayIndexOutOfBoundsException` - if specified field is out of range (`field < 0 || field >= FIELD_COUNT`).

---

internalGet

```
protected final int internalGet(int field)
```

Gets the value for a given time field. This is an internal fast time field value getter for the subclasses.

Parameters:
`field` - the given time field.
Returns:
the value for the given time field.

---

set

```
public void set(int field,
                int value)
```

Sets the time field with the given value.

Parameters:
`field` - the given time field.
`value` - the value to be set for the given time field.
Throws:
`ArrayIndexOutOfBoundsException` - if specified field is out of range (`field < 0 || field >= FIELD_COUNT`).

---

set

```
public final void set(int year,
                      int month,
                      int date)
```

Sets the values for the fields year, month, and date. Previous values of other fields are retained. If this is not desired, call `clear` first.

Parameters:

`year` - the value used to set the YEAR time field.
`month` - the value used to set the MONTH time field. Month value is 0-based. e.g., 0 for January.
`date` - the value used to set the DATE time field.

---

set

```
public final void set(int year,
                      int month,
                      int date,
                      int hour,
                      int minute)
```

Sets the values for the fields year, month, date, hour, and minute. Previous values of other fields are retained. If this is not desired, call `clear` first.

Parameters:

`year` - the value used to set the YEAR time field.
`month` - the value used to set the MONTH time field. Month value is 0-based. e.g., 0 for January.
`date` - the value used to set the DATE time field.
`hour` - the value used to set the HOUR_OF_DAY time field.
`minute` - the value used to set the MINUTE time field.

---

set

```
public final void set(int year,
                      int month,
                      int date,
                      int hour,
                      int minute,
                      int second)
```

Sets the values for the fields year, month, date, hour, minute, and second. Previous values of other fields are retained. If this is not desired, call `clear` first.

Parameters:

`year` - the value used to set the YEAR time field.
`month` - the value used to set the MONTH time field. Month value is 0-based. e.g., 0 for January.
`date` - the value used to set the DATE time field.
`hour` - the value used to set the HOUR_OF_DAY time field.
`minute` - the value used to set the MINUTE time field.
`second` - the value used to set the SECOND time field.

---

clear

```
public final void clear()
```

Clears the values of all the time fields.

---

clear

```
public final void clear(int field)
```

Clears the value in the given time field.

Parameters:

`field` - the time field to be cleared.

---

isSet

```
public final boolean isSet(int field)
```

Determines if the given time field has a value set.

Returns:

true if the given time field has a value set; false otherwise.

---

complete

`protected void complete()`

Fills in any unset fields in the time field list.

---

equals

`public boolean equals(Object obj)`

Compares this calendar to the specified object. The result is `true` if and only if the argument is not `null` and is a `Calendar` object that represents the same calendar as this object.

Overrides:
`equals` in class `Object`

Parameters:
`obj` - the object to compare with.

Returns:
`true` if the objects are the same; `false` otherwise.

See Also:
`Object.hashCode()`, `Hashtable`

---

hashCode

`public int hashCode()`

Returns a hash code for this calendar.

Overrides:
`hashCode` in class `Object`

Returns:
a hash code value for this object.

Since:
1.2

See Also:
`Object.equals(java.lang.Object)`, `Hashtable`

---

before

```
public boolean before(Object when)
```

Compares the time field records. Equivalent to comparing result of conversion to UTC.

Parameters:
`when` - the Calendar to be compared with this Calendar.
Returns:
true if the current time of this Calendar is before the time of Calendar when; false otherwise.

---

after

```
public boolean after(Object when)
```

Compares the time field records. Equivalent to comparing result of conversion to UTC.

Parameters:
`when` - the Calendar to be compared with this Calendar.
Returns:
true if the current time of this Calendar is after the time of Calendar when; false otherwise.

---

add

```
public abstract void add(int field,
                         int amount)
```

Date Arithmetic function. Adds the specified (signed) amount of time to the given time field, based on the calendar's rules. For example, to subtract 5 days from the current time of the calendar, you can achieve it by calling:

add(Calendar.DATE, -5).

Parameters:
`field` - the time field.
`amount` - the amount of date or time to be added to the field.

---

roll

```
public abstract void roll(int field,
                          boolean up)
```

Time Field Rolling function. Adds or subtracts (up/down) a single unit of time on the given time field without changing larger fields. For example, to roll the current date up by one day, you can achieve it by calling:

roll(Calendar.DATE, true). When rolling on the year or Calendar.YEAR field, it will roll the year value in the range between 1 and the value returned by calling getMaximum(Calendar.YEAR). When rolling on the month or Calendar.MONTH field, other fields like date might conflict and, need to be changed. For instance, rolling the month on the date 01/31/96 will result in 02/29/96. When rolling on the hour-in-day or Calendar.HOUR_OF_DAY field, it will roll the hour value in the range between 0 and 23, which is zero-based.

Parameters:

`field` - the time field.

`up` - indicates if the value of the specified time field is to be rolled up or rolled down. Use true if rolling up, false otherwise.

See Also:

`add(int, int)`, `set(int, int)`

---

roll

```
public void roll(int field,
                 int amount)
```

Time Field Rolling function. Add to field a signed amount without changing larger fields. A negative roll amount means to roll down. [NOTE: This default implementation on Calendar just repeatedly calls the version of roll() that takes a boolean and rolls by one unit. This may not always do the right thing. For example, if the DAY_OF_MONTH field is 31, rolling through February will leave it set to 28. The GregorianCalendar version of this function takes care of this problem. Other subclasses should also provide overrides of this function that do the right thing.

Parameters:

`field` - the time field.

`amount` - the signed amount to add to `field`.

Since:

1.2

See Also:

`add(int, int)`, `set(int, int)`

---

setTimeZone

```
public void setTimeZone(TimeZone value)
```

Sets the time zone with the given time zone value.

Parameters:
`value` - the given time zone.

---

getTimeZone

```
public TimeZone getTimeZone()
```

Gets the time zone.

Returns:
the time zone object associated with this calendar.

---

setLenient

```
public void setLenient(boolean lenient)
```

Specify whether or not date/time interpretation is to be lenient. With lenient interpretation, a date such as "February 942, 1996" will be treated as being equivalent to the 941st day after February 1, 1996. With strict interpretation, such dates will cause an exception to be thrown.

See Also:
`DateFormat.setLenient(boolean)`

---

isLenient

```
public boolean isLenient()
```

Tell whether date/time interpretation is to be lenient.

---

setFirstDayOfWeek

```
public void setFirstDayOfWeek(int value)
```

Sets what the first day of the week is; e.g., Sunday in US, Monday in France.

Parameters:
`value` - the given first day of the week.

---

getFirstDayOfWeek

```
public int getFirstDayOfWeek()
```

Gets what the first day of the week is; e.g., Sunday in US, Monday in France.

Returns:
the first day of the week.

---

setMinimalDaysInFirstWeek

```
public void setMinimalDaysInFirstWeek(int value)
```

Sets what the minimal days required in the first week of the year are; For example, if the first week is defined as one that contains the first day of the first month of a year, call the method with value 1. If it must be a full week, use value 7.

Parameters:
`value` - the given minimal days required in the first week of the year.

---

getMinimalDaysInFirstWeek

```
public int getMinimalDaysInFirstWeek()
```

Gets what the minimal days required in the first week of the year are; e.g., if the first week is defined as one that contains the first day of the first month of a year,

getMinimalDaysInFirstWeek returns 1. If the minimal days required must be a full week, getMinimalDaysInFirstWeek returns 7.

Returns:
the minimal days required in the first week of the year.

---

getMinimum

```
public abstract int getMinimum(int field)
```

Gets the minimum value for the given time field. e.g., for Gregorian DAY_OF_MONTH, 1.

Parameters:
`field` - the given time field.
Returns:
the minimum value for the given time field.

---

getMaximum

```
public abstract int getMaximum(int field)
```

Gets the maximum value for the given time field. e.g. for Gregorian DAY_OF_MONTH, 31.

Parameters:
`field` - the given time field.
Returns:
the maximum value for the given time field.

---

getGreatestMinimum

```
public abstract int getGreatestMinimum(int field)
```

Gets the highest minimum value for the given field if varies. Otherwise same as getMinimum (). For Gregorian, no difference.

Parameters:
`field` - the given time field.
Returns:

the highest minimum value for the given time field.

---

getLeastMaximum

```
public abstract int getLeastMaximum(int field)
```

Gets the lowest maximum value for the given field if varies. Otherwise same as getMaximum (). e.g., for Gregorian DAY_OF_MONTH, 28.

Parameters:
`field` - the given time field.
Returns:
the lowest maximum value for the given time field.

---

getActualMinimum

```
public int getActualMinimum(int field)
```

Return the minimum value that this field could have, given the current date. For the Gregorian calendar, this is the same as getMinimum() and getGreatestMinimum(). The version of this function on Calendar uses an iterative algorithm to determine the actual minimum value for the field. There is almost always a more efficient way to accomplish this (in most cases, you can simply return getMinimum()). GregorianCalendar overrides this function with a more efficient implementation.

Parameters:
`field` - the field to determine the minimum of
Returns:
the minimum of the given field for the current date of this Calendar
Since:
1.2

---

getActualMaximum

```
public int getActualMaximum(int field)
```

Return the maximum value that this field could have, given the current date. For example, with the date "Feb 3, 1997" and the DAY_OF_MONTH field, the actual maximum would be

28; for "Feb 3, 1996" it s 29. Similarly for a Hebrew calendar, for some years the actual maximum for MONTH is 12, and for others 13. The version of this function on Calendar uses an iterative algorithm to determine the actual maximum value for the field. There is almost always a more efficient way to accomplish this (in most cases, you can simply return getMaximum()). GregorianCalendar overrides this function with a more efficient implementation.

Parameters:
`field` - the field to determine the maximum of
Returns:
the maximum of the given field for the current date of this Calendar
Since:
1.2

---

clone

`public Object clone()`

Overrides Cloneable

Overrides:
`clone` in class `Object`
Returns:
a clone of this instance.
See Also:
`Cloneable`

---

toString

`public String toString()`

Return a string representation of this calendar. This method is intended to be used only for debugging purposes, and the format of the returned string may vary between implementations. The returned string may be empty but may not be `null`.

Overrides:
`toString` in class `Object`
Returns:
a string representation of this calendar.

Submit a bug or feature

For further API reference and developer documentation, see Java 2 SDK SE Developer Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

java.math

Class BigDecimal

```
java.lang.Object
  └ java.lang.Number
      └ java.math.BigDecimal
```

All Implemented Interfaces:

Comparable, Serializable

---

public class **BigDecimal**
extends Number
implements Comparable

Immutable, arbitrary-precision signed decimal numbers. A BigDecimal consists of an arbitrary precision integer *unscaled value* and a non-negative 32-bit integer *scale*, which represents the number of digits to the right of the decimal point. The number represented by the BigDecimal is `(unscaledValue/10^scale)`. BigDecimal provides operations for basic arithmetic, scale manipulation, comparison, hashing, and format conversion.

The BigDecimal class gives its user complete control over rounding behavior, forcing the user to explicitly specify a rounding behavior for operations capable of discarding precision (`divide(BigDecimal, int)`, `divide(BigDecimal, int, int)`, and `setScale(int, int)`). Eight *rounding modes* are provided for this purpose.

Two types of operations are provided for manipulating the scale of a BigDecimal: scaling/rounding operations and decimal point motion operations. Scaling/rounding operations (`setScale`) return a BigDecimal whose value is approximately (or exactly) equal to that of the operand, but whose scale is the specified value; that is, they increase or decrease the precision of the number with minimal effect on its value. Decimal point motion operations (`movePointLeft(int)` and `movePointRight(int)`) return a BigDecimal created from the operand by moving the decimal point a specified distance in the specified direction; that is, they change a number's value without affecting its precision.

For the sake of brevity and clarity, pseudo-code is used throughout the descriptions of BigDecimal

methods. The pseudo-code expression `(i + j)` is shorthand for "a BigDecimal whose value is that of the BigDecimal `i` plus that of the BigDecimal `j`." The pseudo-code expression `(i == j)` is shorthand for "`true` if and only if the BigDecimal `i` represents the same value as the the BigDecimal `j`." Other pseudo-code expressions are interpreted similarly.

Note: care should be exercised if BigDecimals are to be used as keys in a `SortedMap` or elements in a `SortedSet`, as BigDecimal's *natural ordering* is *inconsistent with equals*. See `Comparable`, `SortedMap` or `SortedSet` for more information.

All methods and constructors for this class throw `NullPointerException` when passed a null object reference for any input parameter.

See Also:

`BigInteger`, `SortedMap`, `SortedSet`, Serialized Form

Field Summary

`static int`	**`ROUND_CEILING`** Rounding mode to round towards positive infinity.
`static int`	**`ROUND_DOWN`** Rounding mode to round towards zero.
`static int`	**`ROUND_FLOOR`** Rounding mode to round towards negative infinity.
`static int`	**`ROUND_HALF_DOWN`** Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round down.
`static int`	**`ROUND_HALF_EVEN`** Rounding mode to round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round towards the even neighbor.
`static int`	**`ROUND_HALF_UP`** Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round up.
`static int`	**`ROUND_UNNECESSARY`** Rounding mode to assert that the requested operation has an exact result, hence no rounding is necessary.
`static int`	**`ROUND_UP`** Rounding mode to round away from zero.

Constructor Summary

BigDecimal(BigInteger val) Translates a BigInteger into a BigDecimal.
BigDecimal(BigInteger unscaledVal, int scale) Translates a BigInteger unscaled value and an int scale into a BigDecimal.
BigDecimal(double val) Translates a double into a BigDecimal.
BigDecimal(String val) Translates the String representation of a BigDecimal into a BigDecimal.

Method Summary

BigDecimal	**abs**() Returns a BigDecimal whose value is the absolute value of this BigDecimal, and whose scale is this.scale().
BigDecimal	**add**(BigDecimal val) Returns a BigDecimal whose value is (this + val), and whose scale is max(this.scale(), val.scale()).
int	**compareTo**(BigDecimal val) Compares this BigDecimal with the specified BigDecimal.
int	**compareTo**(Object o) Compares this BigDecimal with the specified Object.
BigDecimal	**divide**(BigDecimal val, int roundingMode) Returns a BigDecimal whose value is (this / val), and whose scale is this.scale().
BigDecimal	**divide**(BigDecimal val, int scale, int roundingMode) Returns a BigDecimal whose value is (this / val), and whose scale is as specified.
double	**doubleValue**() Converts this BigDecimal to a double.
boolean	**equals**(Object x) Compares this BigDecimal with the specified Object for equality.
float	**floatValue**() Converts this BigDecimal to a float.
int	**hashCode**() Returns the hash code for this BigDecimal.

int	**intValue**() Converts this BigDecimal to an int.
long	**longValue**() Converts this BigDecimal to a long.
BigDecimal	**max**(BigDecimal val) Returns the maximum of this BigDecimal and val.
BigDecimal	**min**(BigDecimal val) Returns the minimum of this BigDecimal and val.
BigDecimal	**movePointLeft**(int n) Returns a BigDecimal which is equivalent to this one with the decimal point moved n places to the left.
BigDecimal	**movePointRight**(int n) Moves the decimal point the specified number of places to the right.
BigDecimal	**multiply**(BigDecimal val) Returns a BigDecimal whose value is (this * val), and whose scale is (this.scale() + val.scale()).
BigDecimal	**negate**() Returns a BigDecimal whose value is (-this), and whose scale is this.scale().
int	**scale**() Returns the *scale* of this BigDecimal.
BigDecimal	**setScale**(int scale) Returns a BigDecimal whose scale is the specified value, and whose value is numerically equal to this BigDecimal's.
BigDecimal	**setScale**(int scale, int roundingMode) Returns a BigDecimal whose scale is the specified value, and whose unscaled value is determined by multiplying or dividing this BigDecimal's unscaled value by the appropriate power of ten to maintain its overall value.
int	**signum**() Returns the signum function of this BigDecimal.
BigDecimal	**subtract**(BigDecimal val) Returns a BigDecimal whose value is (this - val), and whose scale is max(this.scale(), val.scale()).
BigInteger	**toBigInteger**() Converts this BigDecimal to a BigInteger.

String	**toString**() Returns the string representation of this BigDecimal.
BigInteger	**unscaledValue**() Returns a BigInteger whose value is the *unscaled value* of this BigDecimal.
static BigDecimal	**valueOf**(long val) Translates a long value into a BigDecimal with a scale of zero.
static BigDecimal	**valueOf**(long unscaledVal, int scale) Translates a long unscaled value and an int scale into a BigDecimal.

Methods inherited from class java.lang.Number
byteValue, shortValue

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

ROUND_UP

```
public static final int ROUND_UP
```

Rounding mode to round away from zero. Always increments the digit prior to a non-zero discarded fraction. Note that this rounding mode never decreases the magnitude of the calculated value.

See Also:

Constant Field Values

ROUND_DOWN

```
public static final int ROUND_DOWN
```

Rounding mode to round towards zero. Never increments the digit prior to a discarded fraction

(i.e., truncates). Note that this rounding mode never increases the magnitude of the calculated value.

See Also:

Constant Field Values

ROUND_CEILING

```
public static final int ROUND_CEILING
```

Rounding mode to round towards positive infinity. If the BigDecimal is positive, behaves as for `ROUND_UP`; if negative, behaves as for `ROUND_DOWN`. Note that this rounding mode never decreases the calculated value.

See Also:

Constant Field Values

ROUND_FLOOR

```
public static final int ROUND_FLOOR
```

Rounding mode to round towards negative infinity. If the BigDecimal is positive, behave as for `ROUND_DOWN`; if negative, behave as for `ROUND_UP`. Note that this rounding mode never increases the calculated value.

See Also:

Constant Field Values

ROUND_HALF_UP

```
public static final int ROUND_HALF_UP
```

Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round up. Behaves as for `ROUND_UP` if the discarded fraction is >= .5; otherwise, behaves as for `ROUND_DOWN`. Note that this is the rounding mode that most of us were taught in grade school.

See Also:

Constant Field Values

ROUND_HALF_DOWN

```
public static final int ROUND_HALF_DOWN
```

Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round down. Behaves as for `ROUND_UP` if the discarded fraction is > .5; otherwise, behaves as for `ROUND_DOWN`.

See Also:

Constant Field Values

ROUND_HALF_EVEN

```
public static final int ROUND_HALF_EVEN
```

Rounding mode to round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round towards the even neighbor. Behaves as for ROUND_HALF_UP if the digit to the left of the discarded fraction is odd; behaves as for ROUND_HALF_DOWN if it's even. Note that this is the rounding mode that minimizes cumulative error when applied repeatedly over a sequence of calculations.

See Also:

Constant Field Values

ROUND_UNNECESSARY

```
public static final int ROUND_UNNECESSARY
```

Rounding mode to assert that the requested operation has an exact result, hence no rounding is necessary. If this rounding mode is specified on an operation that yields an inexact result, an `ArithmeticException` is thrown.

See Also:

Constant Field Values

Constructor Detail

BigDecimal

`public BigDecimal(String val)`

Translates the String representation of a BigDecimal into a BigDecimal. The String representation consists of an optional sign, `'+'` (`'\u002B'`) or `'-'` (`'\u002D'`), followed by a sequence of zero or more decimal digits ("the integer"), optionally followed by a fraction, optionally followed by an exponent.

The fraction consists of of a decimal point followed by zero or more decimal digits. The string must contain at least one digit in either the integer or the fraction. The number formed by the sign, the integer and the fraction is referred to as the *significand*.

The exponent consists of the character `'e'` (`'\u0075'`) or `'E'` (`'\u0045'`) followed by one or more decimal digits. The value of the exponent must lie between -`Integer.MAX_VALUE` (`Integer.MIN_VALUE`+1) and `Integer.MAX_VALUE`, inclusive.

More formally, the strings this constructor accepts are described by the following grammar:

BigDecimalString:
Sign$_{opt}$ *Significand Exponent*$_{opt}$

Sign:
+
-

Significand:
IntegerPart . *FractionPart*$_{opt}$
. *FractionPart*
IntegerPart

IntegerPart:
Digits

FractionPart:
Digits

Exponent:
ExponentIndicator SignedInteger

ExponentIndicator:
`e`
`E`

SignedInteger:
$Sign_{opt}$ *Digits*

Digits:
Digit
Digits Digit

Digit:
any character for which `Character.isDigit(char)` returns `true`, including 0, 1, 2 ...

The scale of the returned BigDecimal will be the number of digits in the fraction, or zero if the string contains no decimal point, subject to adjustment for any exponent: If the string contains an exponent, the exponent is subtracted from the scale. If the resulting scale is negative, the scale of the returned BigDecimal is zero and the unscaled value is multiplied by the appropriate power of ten so that, in every case, the resulting BigDecimal is equal to *significand* × $10^{exponent}$. (If in the future this specification is amended to permit negative scales, the final step of zeroing the scale and adjusting the unscaled value will be eliminated.)

The character-to-digit mapping is provided by `Character.digit(char, int)` set to convert to radix 10. The String may not contain any extraneous characters (whitespace, for example).

Note: For values other `float` and `double` NaN and ±Infinity, this constructor is compatible with the values returned by `Float.toString(float)` and `Double.toString(double)`. This is generally the preferred way to convert a `float` or `double` into a BigDecimal, as it doesn't suffer from the unpredictability of the `BigDecimal(double)` constructor.

Note: the optional leading plus sign and trailing exponent were added in release 1.3.

Parameters:
`val` - String representation of BigDecimal.
Throws:
`NumberFormatException` - `val` is not a valid representation of a BigDecimal.

BigDecimal

```
public BigDecimal(double val)
```

Translates a `double` into a BigDecimal. The scale of the BigDecimal is the smallest value such that `(10^scale * val)` is an integer.

Note: the results of this constructor can be somewhat unpredictable. One might assume that `new BigDecimal(.1)` is exactly equal to .1, but it is actually equal to .1000000000000000055511151231257827021181583404541015625. This is so because .1 cannot be represented exactly as a double (or, for that matter, as a binary fraction of any finite length). Thus, the long value that is being passed *in* to the constructor is not exactly equal to .1, appearances notwithstanding.

The (String) constructor, on the other hand, is perfectly predictable: `new BigDecimal(".1")` is *exactly* equal to .1, as one would expect. Therefore, it is generally recommended that the (String) constructor be used in preference to this one.

Parameters:

`val` - `double` value to be converted to BigDecimal.

Throws:

`NumberFormatException` - `val` if `val` is infinite or NaN.

BigDecimal

```
public BigDecimal(BigInteger val)
```

Translates a BigInteger into a BigDecimal. The scale of the BigDecimal is zero.

Parameters:

`val` - BigInteger value to be converted to BigDecimal.

BigDecimal

```
public BigDecimal(BigInteger unscaledVal,
                  int scale)
```

Translates a BigInteger unscaled value and an `int` scale into a BigDecimal. The value of the BigDecimal is `(unscaledVal/10^scale)`.

Parameters:

`unscaledVal` - unscaled value of the BigDecimal.
`scale` - scale of the BigDecimal.

Throws:

`NumberFormatException` - scale is negative

Method Detail

valueOf

```
public static BigDecimal valueOf(long unscaledVal,
                                 int scale)
```

Translates a `long` unscaled value and an `int` scale into a BigDecimal. This "static factory method" is provided in preference to a (`long`, `int`) constructor because it allows for reuse of frequently used BigDecimals.

Parameters:

`unscaledVal` - unscaled value of the BigDecimal.
`scale` - scale of the BigDecimal.

Returns:

a BigDecimal whose value is $(unscaledVal/10^{scale})$.

---

valueOf

```
public static BigDecimal valueOf(long val)
```

Translates a `long` value into a BigDecimal with a scale of zero. This "static factory method" is provided in preference to a (`long`) constructor because it allows for reuse of frequently used BigDecimals.

Parameters:

`val` - value of the BigDecimal.

Returns:

a BigDecimal whose value is `val`.

---

add

```
public BigDecimal add(BigDecimal val)
```

Returns a BigDecimal whose value is `(this + val)`, and whose scale is `max(this.scale(), val.scale())`.

Parameters:
`val` - value to be added to this BigDecimal.
Returns:
`this + val`

---

subtract

```
public BigDecimal subtract(BigDecimal val)
```

Returns a BigDecimal whose value is `(this - val)`, and whose scale is `max(this.scale(), val.scale())`.

Parameters:
`val` - value to be subtracted from this BigDecimal.
Returns:
`this - val`

---

multiply

```
public BigDecimal multiply(BigDecimal val)
```

Returns a BigDecimal whose value is `(this * val)`, and whose scale is `(this.scale() + val.scale())`.

Parameters:
`val` - value to be multiplied by this BigDecimal.
Returns:
`this * val`

---

divide

```
public BigDecimal divide(BigDecimal val,
                         int scale,
                         int roundingMode)
```

Returns a BigDecimal whose value is `(this / val)`, and whose scale is as specified. If rounding must be performed to generate a result with the specified scale, the specified rounding mode is applied.

Parameters:

`val` - value by which this BigDecimal is to be divided.
`scale` - scale of the BigDecimal quotient to be returned.
`roundingMode` - rounding mode to apply.

Returns:

`this / val`

Throws:

`ArithmeticException` - `val` is zero, `scale` is negative, or `roundingMode==ROUND_UNNECESSARY` and the specified scale is insufficient to represent the result of the division exactly.
`IllegalArgumentException` - `roundingMode` does not represent a valid rounding mode.

See Also:

`ROUND_UP`, `ROUND_DOWN`, `ROUND_CEILING`, `ROUND_FLOOR`, `ROUND_HALF_UP`, `ROUND_HALF_DOWN`, `ROUND_HALF_EVEN`, `ROUND_UNNECESSARY`

---

divide

```
public BigDecimal divide(BigDecimal val,
                         int roundingMode)
```

Returns a BigDecimal whose value is `(this / val)`, and whose scale is `this.scale()`. If rounding must be performed to generate a result with the given scale, the specified rounding mode is applied.

Parameters:

`val` - value by which this BigDecimal is to be divided.
`roundingMode` - rounding mode to apply.

Returns:

`this / val`

Throws:

`ArithmeticException` - `val==0`, or `roundingMode==ROUND_UNNECESSARY` and `this.scale()` is insufficient to represent the result of the division exactly.
`IllegalArgumentException` - `roundingMode` does not represent a valid rounding mode.

See Also:

ROUND_UP, ROUND_DOWN, ROUND_CEILING, ROUND_FLOOR, ROUND_HALF_UP, ROUND_HALF_DOWN, ROUND_HALF_EVEN, ROUND_UNNECESSARY

---

abs

```
public BigDecimal abs()
```

Returns a BigDecimal whose value is the absolute value of this BigDecimal, and whose scale is `this.scale()`.

Returns:

`abs(this)`

---

negate

```
public BigDecimal negate()
```

Returns a BigDecimal whose value is `(-this)`, and whose scale is `this.scale()`.

Returns:

`-this`

---

signum

```
public int signum()
```

Returns the signum function of this BigDecimal.

Returns:

-1, 0 or 1 as the value of this BigDecimal is negative, zero or positive.

---

scale

```
public int scale()
```

Returns the *scale* of this BigDecimal. (The scale is the number of digits to the right of the decimal point.)

Returns:
the scale of this BigDecimal.

---

unscaledValue

```
public BigInteger unscaledValue()
```

Returns a BigInteger whose value is the *unscaled value* of this BigDecimal. (Computes `(this * 10^this.scale())`.)

Returns:
the unscaled value of this BigDecimal.
Since:
1.2

---

setScale

```
public BigDecimal setScale(int scale,
                           int roundingMode)
```

Returns a BigDecimal whose scale is the specified value, and whose unscaled value is determined by multiplying or dividing this BigDecimal's unscaled value by the appropriate power of ten to maintain its overall value. If the scale is reduced by the operation, the unscaled value must be divided (rather than multiplied), and the value may be changed; in this case, the specified rounding mode is applied to the division.

Note that since BigDecimal objects are immutable, calls of this method do *not* result in the original object being modified, contrary to the usual convention of having methods named `set`*X* mutate field *X*. Instead, `setScale` returns an object with the proper scale; the returned object may or may not be newly allocated.

Parameters:
`scale` - scale of the BigDecimal value to be returned.
`roundingMode` - The rounding mode to apply.
Returns:
a BigDecimal whose scale is the specified value, and whose unscaled value is

determined by multiplying or dividing this BigDecimal's unscaled value by the appropriate power of ten to maintain its overall value.

Throws:

`ArithmeticException` - `scale` is negative, or `roundingMode==ROUND_UNNECESSARY` and the specified scaling operation would require rounding.

`IllegalArgumentException` - `roundingMode` does not represent a valid rounding mode.

See Also:

`ROUND_UP`, `ROUND_DOWN`, `ROUND_CEILING`, `ROUND_FLOOR`, `ROUND_HALF_UP`, `ROUND_HALF_DOWN`, `ROUND_HALF_EVEN`, `ROUND_UNNECESSARY`

---

setScale

```
public BigDecimal setScale(int scale)
```

Returns a BigDecimal whose scale is the specified value, and whose value is numerically equal to this BigDecimal's. Throws an ArithmeticException if this is not possible. This call is typically used to increase the scale, in which case it is guaranteed that there exists a BigDecimal of the specified scale and the correct value. The call can also be used to reduce the scale if the caller knows that the BigDecimal has sufficiently many zeros at the end of its fractional part (i.e., factors of ten in its integer value) to allow for the rescaling without loss of precision.

This method returns the same result as the two argument version of setScale, but saves the caller the trouble of specifying a rounding mode in cases where it is irrelevant.

Note that since BigDecimal objects are immutable, calls of this method do *not* result in the original object being modified, contrary to the usual convention of having methods named `set`*X* mutate field *X*. Instead, `setScale` returns an object with the proper scale; the returned object may or may not be newly allocated.

Parameters:

`scale` - scale of the BigDecimal value to be returned.

Returns:

a BigDecimal whose scale is the specified value, and whose unscaled value is determined by multiplying or dividing this BigDecimal's unscaled value by the appropriate power of ten to maintain its overall value.

Throws:

`ArithmeticException` - `scale` is negative, or the specified scaling operation would require rounding.

See Also:

setScale(int, int)

---

movePointLeft

```
public BigDecimal movePointLeft(int n)
```

Returns a BigDecimal which is equivalent to this one with the decimal point moved n places to the left. If n is non-negative, the call merely adds n to the scale. If n is negative, the call is equivalent to movePointRight(-n). (The BigDecimal returned by this call has value `(this * 10^-n)` and scale `max(this.scale()+n, 0)`.)

Parameters:

n - number of places to move the decimal point to the left.

Returns:

a BigDecimal which is equivalent to this one with the decimal point moved n places to the left.

---

movePointRight

```
public BigDecimal movePointRight(int n)
```

Moves the decimal point the specified number of places to the right. If this BigDecimal's scale is >= n, the call merely subtracts n from the scale; otherwise, it sets the scale to zero, and multiplies the integer value by `10^(n - this.scale)`. If n is negative, the call is equivalent to `movePointLeft(-n)`. (The BigDecimal returned by this call has value `(this * 10^n)` and scale `max(this.scale()-n, 0)`.)

Parameters:

n - number of places to move the decimal point to the right.

Returns:

a BigDecimal which is equivalent to this one with the decimal point moved n places to the right.

---

compareTo

```
public int compareTo(BigDecimal val)
```

Compares this BigDecimal with the specified BigDecimal. Two BigDecimals that are equal in value but have a different scale (like 2.0 and 2.00) are considered equal by this method. This method is provided in preference to individual methods for each of the six boolean comparison operators (<, ==, >, >=, !=, <=). The suggested idiom for performing these comparisons is: `(x.compareTo(y)` <*op*> `0)`, where <*op*> is one of the six comparison operators.

Parameters:

`val` - BigDecimal to which this BigDecimal is to be compared.

Returns:

-1, 0 or 1 as this BigDecimal is numerically less than, equal to, or greater than `val`.

---

compareTo

```
public int compareTo(Object o)
```

Compares this BigDecimal with the specified Object. If the Object is a BigDecimal, this method behaves like `compareTo`. Otherwise, it throws a `ClassCastException` (as BigDecimals are comparable only to other BigDecimals).

Specified by:

`compareTo` in interface `Comparable`

Parameters:

`o` - Object to which this BigDecimal is to be compared.

Returns:

a negative number, zero, or a positive number as this BigDecimal is numerically less than, equal to, or greater than `o`, which must be a BigDecimal.

Throws:

`ClassCastException` - `o` is not a BigDecimal.

Since:

1.2

See Also:

`compareTo(java.math.BigDecimal)`, `Comparable`

---

equals

```
public boolean equals(Object x)
```

Compares this BigDecimal with the specified Object for equality. Unlike `compareTo`, this

method considers two BigDecimals equal only if they are equal in value and scale (thus 2.0 is not equal to 2.00 when compared by this method).

Overrides:

`equals` in class `Object`

Parameters:

`x` - Object to which this BigDecimal is to be compared.

Returns:

`true` if and only if the specified Object is a BigDecimal whose value and scale are equal to this BigDecimal's.

See Also:

`compareTo(java.math.BigDecimal)`

---

min

```
public BigDecimal min(BigDecimal val)
```

Returns the minimum of this BigDecimal and `val`.

Parameters:

`val` - value with which the minimum is to be computed.

Returns:

the BigDecimal whose value is the lesser of this BigDecimal and `val`. If they are equal, as defined by the `compareTo` method, either may be returned.

See Also:

`compareTo(java.math.BigDecimal)`

---

max

```
public BigDecimal max(BigDecimal val)
```

Returns the maximum of this BigDecimal and `val`.

Parameters:

`val` - value with which the maximum is to be computed.

Returns:

the BigDecimal whose value is the greater of this BigDecimal and `val`. If they are equal, as defined by the `compareTo` method, either may be returned.

See Also:

compareTo(java.math.BigDecimal)

---

hashCode

```
public int hashCode()
```

Returns the hash code for this BigDecimal. Note that two BigDecimals that are numerically equal but differ in scale (like 2.0 and 2.00) will generally *not* have the same hash code.

Overrides:
hashCode in class Object

Returns:
hash code for this BigDecimal.

See Also:
Object.equals(java.lang.Object), Hashtable

---

toString

```
public String toString()
```

Returns the string representation of this BigDecimal. The digit-to- character mapping provided by Character.forDigit(int, int) is used. A leading minus sign is used to indicate sign, and the number of digits to the right of the decimal point is used to indicate scale. (This representation is compatible with the (String) constructor.)

Overrides:
toString in class Object

Returns:
String representation of this BigDecimal.

See Also:
Character.forDigit(int, int), BigDecimal(java.lang.String)

---

toBigInteger

```
public BigInteger toBigInteger()
```

Converts this BigDecimal to a BigInteger. This conversion is analogous to a *narrowing*

primitive conversion from `double` to `long` as defined in the Java Language Specification: any fractional part of this BigDecimal will be discarded. Note that this conversion can lose information about the precision of the BigDecimal value.

Returns:

this BigDecimal converted to a BigInteger.

---

intValue

```
public int intValue()
```

Converts this BigDecimal to an `int`. This conversion is analogous to a *narrowing primitive conversion* from `double` to `short` as defined in the Java Language Specification: any fractional part of this BigDecimal will be discarded, and if the resulting "BigInteger" is too big to fit in an `int`, only the low-order 32 bits are returned. Note that this conversion can lose information about the overall magnitude and precision of the BigDecimal value as well as return a result with the opposite sign.

Specified by:

`intValue` in class `Number`

Returns:

this BigDecimal converted to an `int`.

---

longValue

```
public long longValue()
```

Converts this BigDecimal to a `long`. This conversion is analogous to a *narrowing primitive conversion* from `double` to `short` as defined in the Java Language Specification: any fractional part of this BigDecimal will be discarded, and if the resulting "BigInteger" is too big to fit in a `long`, only the low-order 64 bits are returned. Note that this conversion can lose information about the overall magnitude and precision of the BigDecimal value as well as return a result with the opposite sign.

Specified by:

`longValue` in class `Number`

Returns:

this BigDecimal converted to an `long`.

---

floatValue

```
public float floatValue()
```

Converts this BigDecimal to a `float`. This conversion is similar to the *narrowing primitive conversion* from `double` to `float` defined in the Java Language Specification: if this BigDecimal has too great a magnitude to represent as a `float`, it will be converted to `Float.NEGATIVE_INFINITY` or `Float.POSITIVE_INFINITY` as appropriate. Note that even when the return value is finite, this conversion can lose information about the precision of the BigDecimal value.

Specified by:

`floatValue` in class `Number`

Returns:

this BigDecimal converted to a `float`.

---

doubleValue

```
public double doubleValue()
```

Converts this BigDecimal to a `double`. This conversion is similar to the *narrowing primitive conversion* from `double` to `float` as defined in the Java Language Specification: if this BigDecimal has too great a magnitude represent as a `double`, it will be converted to `Double.NEGATIVE_INFINITY` or `Double.POSITIVE_INFINITY` as appropriate. Note that even when the return value is finite, this conversion can lose information about the precision of the BigDecimal value.

Specified by:

`doubleValue` in class `Number`

Returns:

this BigDecimal converted to a `double`.

---

definitions of terms, workarounds, and working code examples.

---

java.math

Class BigInteger

```
java.lang.Object
  └ java.lang.Number
      └ java.math.BigInteger
```

All Implemented Interfaces:

Comparable, Serializable

---

public class **BigInteger**
extends Number
implements Comparable

Immutable arbitrary-precision integers. All operations behave as if BigIntegers were represented in two's-complement notation (like Java's primitive integer types). BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods from java.lang.Math. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations.

Semantics of arithmetic operations exactly mimic those of Java's integer arithmetic operators, as defined in *The Java Language Specification*. For example, division by zero throws an `ArithmeticException`, and division of a negative by a positive yields a negative (or zero) remainder. All of the details in the Spec concerning overflow are ignored, as BigIntegers are made as large as necessary to accommodate the results of an operation.

Semantics of shift operations extend those of Java's shift operators to allow for negative shift distances. A right-shift with a negative shift distance results in a left shift, and vice-versa. The unsigned right shift operator (>>>) is omitted, as this operation makes little sense in combination with the "infinite word size" abstraction provided by this class.

Semantics of bitwise logical operations exactly mimic those of Java's bitwise integer operators. The binary operators (`and`, `or`, `xor`) implicitly perform sign extension on the shorter of the two operands prior to performing the operation.

Comparison operations perform signed integer comparisons, analogous to those performed by Java's

relational and equality operators.

Modular arithmetic operations are provided to compute residues, perform exponentiation, and compute multiplicative inverses. These methods always return a non-negative result, between `0` and `(modulus - 1)`, inclusive.

Bit operations operate on a single bit of the two's-complement representation of their operand. If necessary, the operand is sign- extended so that it contains the designated bit. None of the single-bit operations can produce a BigInteger with a different sign from the BigInteger being operated on, as they affect only a single bit, and the "infinite word size" abstraction provided by this class ensures that there are infinitely many "virtual sign bits" preceding each BigInteger.

For the sake of brevity and clarity, pseudo-code is used throughout the descriptions of BigInteger methods. The pseudo-code expression `(i + j)` is shorthand for "a BigInteger whose value is that of the BigInteger `i` plus that of the BigInteger `j`." The pseudo-code expression `(i == j)` is shorthand for "`true` if and only if the BigInteger `i` represents the same value as the the BigInteger `j`." Other pseudo-code expressions are interpreted similarly.

All methods and constructors in this class throw `NullPointerException` when passed a null object reference for any input parameter.

Since:

JDK1.1

See Also:

`BigDecimal`, Serialized Form

Field Summary

`static BigInteger`	**`ONE`** The BigInteger constant one.
`static BigInteger`	**`ZERO`** The BigInteger constant zero.

Constructor Summary

`BigInteger``(byte[] val)` Translates a byte array containing the two's-complement binary representation of a BigInteger into a BigInteger.
`BigInteger``(int signum, byte[] magnitude)` Translates the sign-magnitude representation of a BigInteger into a BigInteger.

BigInteger(int bitLength, int certainty, Random rnd) Constructs a randomly generated positive BigInteger that is probably prime, with the specified bitLength.
BigInteger(int numBits, Random rnd) Constructs a randomly generated BigInteger, uniformly distributed over the range 0 to $(2^{numBits} - 1)$, inclusive.
BigInteger(String val) Translates the decimal String representation of a BigInteger into a BigInteger.
BigInteger(String val, int radix) Translates the String representation of a BigInteger in the specified radix into a BigInteger.

Method Summary

BigInteger	**abs**() Returns a BigInteger whose value is the absolute value of this BigInteger.
BigInteger	**add**(BigInteger val) Returns a BigInteger whose value is (this + val).
BigInteger	**and**(BigInteger val) Returns a BigInteger whose value is (this & val).
BigInteger	**andNot**(BigInteger val) Returns a BigInteger whose value is (this & ~val).
int	**bitCount**() Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit.
int	**bitLength**() Returns the number of bits in the minimal two's-complement representation of this BigInteger, *excluding* a sign bit.
BigInteger	**clearBit**(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared.
int	**compareTo**(BigInteger val) Compares this BigInteger with the specified BigInteger.
int	**compareTo**(Object o) Compares this BigInteger with the specified Object.
BigInteger	**divide**(BigInteger val) Returns a BigInteger whose value is (this / val).

BigInteger[]	**divideAndRemainder**(BigInteger val) Returns an array of two BigIntegers containing (this / val) followed by (this % val).
double	**doubleValue**() Converts this BigInteger to a double.
boolean	**equals**(Object x) Compares this BigInteger with the specified Object for equality.
BigInteger	**flipBit**(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped.
float	**floatValue**() Converts this BigInteger to a float.
BigInteger	**gcd**(BigInteger val) Returns a BigInteger whose value is the greatest common divisor of abs(this) and abs(val).
int	**getLowestSetBit**() Returns the index of the rightmost (lowest-order) one bit in this BigInteger (the number of zero bits to the right of the rightmost one bit).
int	**hashCode**() Returns the hash code for this BigInteger.
int	**intValue**() Converts this BigInteger to an int.
boolean	**isProbablePrime**(int certainty) Returns true if this BigInteger is probably prime, false if it's definitely composite.
long	**longValue**() Converts this BigInteger to a long.
BigInteger	**max**(BigInteger val) Returns the maximum of this BigInteger and val.
BigInteger	**min**(BigInteger val) Returns the minimum of this BigInteger and val.
BigInteger	**mod**(BigInteger m) Returns a BigInteger whose value is (this mod m).
BigInteger	**modInverse**(BigInteger m) Returns a BigInteger whose value is ($this^{-1}$ mod m).

BigInteger	**modPow**(BigInteger exponent, BigInteger m) Returns a BigInteger whose value is ($this^{exponent}$ mod m).
BigInteger	**multiply**(BigInteger val) Returns a BigInteger whose value is (this * val).
BigInteger	**negate**() Returns a BigInteger whose value is (-this).
BigInteger	**not**() Returns a BigInteger whose value is (~this).
BigInteger	**or**(BigInteger val) Returns a BigInteger whose value is (this | val).
BigInteger	**pow**(int exponent) Returns a BigInteger whose value is ($this^{exponent}$).
static BigInteger	**probablePrime**(int bitLength, Random rnd) Returns a positive BigInteger that is probably prime, with the specified bitLength.
BigInteger	**remainder**(BigInteger val) Returns a BigInteger whose value is (this % val).
BigInteger	**setBit**(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set.
BigInteger	**shiftLeft**(int n) Returns a BigInteger whose value is (this << n).
BigInteger	**shiftRight**(int n) Returns a BigInteger whose value is (this >> n).
int	**signum**() Returns the signum function of this BigInteger.
BigInteger	**subtract**(BigInteger val) Returns a BigInteger whose value is (this - val).
boolean	**testBit**(int n) Returns true if and only if the designated bit is set.
byte[]	**toByteArray**() Returns a byte array containing the two's-complement representation of this BigInteger.
String	**toString**() Returns the decimal String representation of this BigInteger.

String	**toString**(int radix) Returns the String representation of this BigInteger in the given radix.
static BigInteger	**valueOf**(long val) Returns a BigInteger whose value is equal to that of the specified long.
BigInteger	**xor**(BigInteger val) Returns a BigInteger whose value is (this ^ val).

Methods inherited from class java.lang.Number
byteValue, shortValue

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

ZERO

```
public static final BigInteger ZERO
```

The BigInteger constant zero.

Since:
1.2

ONE

```
public static final BigInteger ONE
```

The BigInteger constant one.

Since:
1.2

Constructor Detail

BigInteger

```
public BigInteger(byte[] val)
```

Translates a byte array containing the two's-complement binary representation of a BigInteger into a BigInteger. The input array is assumed to be in *big-endian* byte-order: the most significant byte is in the zeroth element.

Parameters:

`val` - big-endian two's-complement binary representation of BigInteger.

Throws:

`NumberFormatException` - `val` is zero bytes long.

BigInteger

```
public BigInteger(int signum,
                  byte[] magnitude)
```

Translates the sign-magnitude representation of a BigInteger into a BigInteger. The sign is represented as an integer signum value: -1 for negative, 0 for zero, or 1 for positive. The magnitude is a byte array in *big-endian* byte-order: the most significant byte is in the zeroth element. A zero-length magnitude array is permissible, and will result in in a BigInteger value of 0, whether signum is -1, 0 or 1.

Parameters:

`signum` - signum of the number (-1 for negative, 0 for zero, 1 for positive).
`magnitude` - big-endian binary representation of the magnitude of the number.

Throws:

`NumberFormatException` - `signum` is not one of the three legal values (-1, 0, and 1), or `signum` is 0 and `magnitude` contains one or more non-zero bytes.

BigInteger

```
public BigInteger(String val,
                  int radix)
```

Translates the String representation of a BigInteger in the specified radix into a BigInteger.

The String representation consists of an optional minus sign followed by a sequence of one or more digits in the specified radix. The character-to-digit mapping is provided by `Character.digit`. The String may not contain any extraneous characters (whitespace, for example).

Parameters:

`val` - String representation of BigInteger.
`radix` - radix to be used in interpreting `val`.

Throws:

`NumberFormatException` - `val` is not a valid representation of a BigInteger in the specified radix, or `radix` is outside the range from `Character.MIN_RADIX` to `Character.MAX_RADIX`, inclusive.

See Also:

`Character.digit(char, int)`

BigInteger

```
public BigInteger(String val)
```

Translates the decimal String representation of a BigInteger into a BigInteger. The String representation consists of an optional minus sign followed by a sequence of one or more decimal digits. The character-to-digit mapping is provided by `Character.digit`. The String may not contain any extraneous characters (whitespace, for example).

Parameters:

`val` - decimal String representation of BigInteger.

Throws:

`NumberFormatException` - `val` is not a valid representation of a BigInteger.

See Also:

`Character.digit(char, int)`

BigInteger

```
public BigInteger(int numBits,
                  Random rnd)
```

Constructs a randomly generated BigInteger, uniformly distributed over the range `0` to $(2^{numBits} - 1)$, inclusive. The uniformity of the distribution assumes that a fair source of random bits is provided in `rnd`. Note that this constructor always constructs a non-negative BigInteger.

Parameters:

`numBits` - maximum bitLength of the new BigInteger.
`rnd` - source of randomness to be used in computing the new BigInteger.

Throws:

`IllegalArgumentException` - `numBits` is negative.

See Also:

`bitLength()`

BigInteger

```
public BigInteger(int bitLength,
                  int certainty,
                  Random rnd)
```

Constructs a randomly generated positive BigInteger that is probably prime, with the specified bitLength.

It is recommended that the `probablePrime` method be used in preference to this constructor unless there is a compelling need to specify a certainty.

Parameters:

`bitLength` - bitLength of the returned BigInteger.
`certainty` - a measure of the uncertainty that the caller is willing to tolerate. The probability that the new BigInteger represents a prime number will exceed $(1 - 1/2^{certainty})$. The execution time of this constructor is proportional to the value of this parameter.
`rnd` - source of random bits used to select candidates to be tested for primality.

Throws:

`ArithmeticException` - `bitLength < 2`.

See Also:

`bitLength()`

Method Detail

probablePrime

```
public static BigInteger probablePrime(int bitLength,
                                       Random rnd)
```

Returns a positive BigInteger that is probably prime, with the specified bitLength. The probability that a BigInteger returned by this method is composite does not exceed 2^{-100}.

Parameters:

`bitLength` - bitLength of the returned BigInteger.

`rnd` - source of random bits used to select candidates to be tested for primality.

Returns:

a BigInteger of `bitLength` bits that is probably prime

Throws:

`ArithmeticException` - `bitLength < 2`.

See Also:

`bitLength()`

---

valueOf

```
public static BigInteger valueOf(long val)
```

Returns a BigInteger whose value is equal to that of the specified `long`. This "static factory method" is provided in preference to a (`long`) constructor because it allows for reuse of frequently used BigIntegers.

Parameters:

`val` - value of the BigInteger to return.

Returns:

a BigInteger with the specified value.

---

add

```
public BigInteger add(BigInteger val)
```

Returns a BigInteger whose value is `(this + val)`.

Parameters:

`val` - value to be added to this BigInteger.

Returns:

`this + val`

---

subtract

```
public BigInteger subtract(BigInteger val)
```

Returns a BigInteger whose value is `(this - val)`.

Parameters:
`val` - value to be subtracted from this BigInteger.
Returns:
`this - val`

---

multiply

```
public BigInteger multiply(BigInteger val)
```

Returns a BigInteger whose value is `(this * val)`.

Parameters:
`val` - value to be multiplied by this BigInteger.
Returns:
`this * val`

---

divide

```
public BigInteger divide(BigInteger val)
```

Returns a BigInteger whose value is `(this / val)`.

Parameters:
`val` - value by which this BigInteger is to be divided.
Returns:
`this / val`
Throws:
`ArithmeticException` - `val==0`

---

divideAndRemainder

```
public BigInteger[] divideAndRemainder(BigInteger val)
```

Returns an array of two BigIntegers containing `(this / val)` followed by `(this % val)`.

Parameters:
`val` - value by which this BigInteger is to be divided, and the remainder computed.
Returns:
an array of two BigIntegers: the quotient `(this / val)` is the initial element, and the remainder `(this % val)` is the final element.
Throws:
`ArithmeticException` - `val==0`

---

remainder

`public BigInteger remainder(BigInteger val)`

Returns a BigInteger whose value is `(this % val)`.

Parameters:
`val` - value by which this BigInteger is to be divided, and the remainder computed.
Returns:
`this % val`
Throws:
`ArithmeticException` - `val==0`

---

pow

`public BigInteger pow(int exponent)`

Returns a BigInteger whose value is $(this^{exponent})$. Note that `exponent` is an integer rather than a BigInteger.

Parameters:
`exponent` - exponent to which this BigInteger is to be raised.
Returns:
$this^{exponent}$
Throws:
`ArithmeticException` - `exponent` is negative. (This would cause the operation to yield a non-integer value.)

---

gcd

`public BigInteger gcd(BigInteger val)`

Returns a BigInteger whose value is the greatest common divisor of `abs(this)` and `abs(val)`. Returns 0 if `this==0 && val==0`.

Parameters:
`val` - value with with the GCD is to be computed.
Returns:
`GCD(abs(this), abs(val))`

---

abs

`public BigInteger abs()`

Returns a BigInteger whose value is the absolute value of this BigInteger.

Returns:
`abs(this)`

---

negate

`public BigInteger negate()`

Returns a BigInteger whose value is `(-this)`.

Returns:
`-this`

---

signum

`public int signum()`

Returns the signum function of this BigInteger.

Returns:

-1, 0 or 1 as the value of this BigInteger is negative, zero or positive.

---

mod

```
public BigInteger mod(BigInteger m)
```

Returns a BigInteger whose value is `(this mod m)`. This method differs from `remainder` in that it always returns a *non-negative* BigInteger.

Parameters:

`m` - the modulus.

Returns:

`this mod m`

Throws:

`ArithmeticException` - `m <= 0`

See Also:

`remainder(java.math.BigInteger)`

---

modPow

```
public BigInteger modPow(BigInteger exponent,
                         BigInteger m)
```

Returns a BigInteger whose value is $(this^{exponent} \bmod m)$. (Unlike `pow`, this method permits negative exponents.)

Parameters:

`exponent` - the exponent.

`m` - the modulus.

Returns:

$this^{exponent} \bmod m$

Throws:

`ArithmeticException` - `m <= 0`

See Also:

`modInverse(java.math.BigInteger)`

---

modInverse

`public BigInteger modInverse(BigInteger m)`

Returns a BigInteger whose value is $(this^{-1} \bmod m)$.

Parameters:
`m` - the modulus.
Returns:
$this^{-1} \bmod m$.
Throws:
`ArithmeticException` - `m <= 0`, or this BigInteger has no multiplicative inverse mod m (that is, this BigInteger is not *relatively prime* to m).

---

shiftLeft

`public BigInteger shiftLeft(int n)`

Returns a BigInteger whose value is $(this \ll n)$. The shift distance, `n`, may be negative, in which case this method performs a right shift. (Computes $floor(this * 2^n)$.)

Parameters:
`n` - shift distance, in bits.
Returns:
`this << n`
See Also:
`shiftRight(int)`

---

shiftRight

`public BigInteger shiftRight(int n)`

Returns a BigInteger whose value is $(this \gg n)$. Sign extension is performed. The shift distance, `n`, may be negative, in which case this method performs a left shift. (Computes $floor(this / 2^n)$.)

Parameters:
`n` - shift distance, in bits.
Returns:

`this >> n`

See Also:

`shiftLeft(int)`

---

and

`public BigInteger and(BigInteger val)`

Returns a BigInteger whose value is `(this & val)`. (This method returns a negative BigInteger if and only if this and val are both negative.)

Parameters:

`val` - value to be AND'ed with this BigInteger.

Returns:

`this & val`

---

or

`public BigInteger or(BigInteger val)`

Returns a BigInteger whose value is `(this | val)`. (This method returns a negative BigInteger if and only if either this or val is negative.)

Parameters:

`val` - value to be OR'ed with this BigInteger.

Returns:

`this | val`

---

xor

`public BigInteger xor(BigInteger val)`

Returns a BigInteger whose value is `(this ^ val)`. (This method returns a negative BigInteger if and only if exactly one of this and val are negative.)

Parameters:

`val` - value to be XOR'ed with this BigInteger.

Returns:
`this ^ val`

---

not

`public BigInteger **not**()`

Returns a BigInteger whose value is `(~this)`. (This method returns a negative value if and only if this BigInteger is non-negative.)

Returns:
`~this`

---

andNot

`public BigInteger **andNot**(BigInteger val)`

Returns a BigInteger whose value is `(this & ~val)`. This method, which is equivalent to `and(val.not())`, is provided as a convenience for masking operations. (This method returns a negative BigInteger if and only if `this` is negative and `val` is positive.)

Parameters:
`val` - value to be complemented and AND'ed with this BigInteger.
Returns:
`this & ~val`

---

testBit

`public boolean **testBit**(int n)`

Returns `true` if and only if the designated bit is set. (Computes `((this & (1<<n)) != 0)`.)

Parameters:
`n` - index of bit to test.
Returns:
`true` if and only if the designated bit is set.
Throws:

ArithmeticException - n is negative.

---

setBit

```
public BigInteger setBit(int n)
```

Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set. (Computes `(this | (1<<n))`.)

Parameters:
n - index of bit to set.
Returns:
`this | (1<<n)`
Throws:
ArithmeticException - n is negative.

---

clearBit

```
public BigInteger clearBit(int n)
```

Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared. (Computes `(this & ~(1<<n))`.)

Parameters:
n - index of bit to clear.
Returns:
`this & ~(1<<n)`
Throws:
ArithmeticException - n is negative.

---

flipBit

```
public BigInteger flipBit(int n)
```

Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped. (Computes `(this ^ (1<<n))`.)

Parameters:

n - index of bit to flip.

Returns:

`this ^ (1<<n)`

Throws:

`ArithmeticException` - n is negative.

---

getLowestSetBit

`public int **getLowestSetBit**()`

Returns the index of the rightmost (lowest-order) one bit in this BigInteger (the number of zero bits to the right of the rightmost one bit). Returns -1 if this BigInteger contains no one bits. (Computes $(\texttt{this==0? -1 : } \log_2(\texttt{this \& -this}))$.)

Returns:

index of the rightmost one bit in this BigInteger.

---

bitLength

`public int **bitLength**()`

Returns the number of bits in the minimal two's-complement representation of this BigInteger, *excluding* a sign bit. For positive BigIntegers, this is equivalent to the number of bits in the ordinary binary representation. (Computes $(\texttt{ceil}(\log_2(\texttt{this < 0 ? -this : this +1})))$.)

Returns:

number of bits in the minimal two's-complement representation of this BigInteger, *excluding* a sign bit.

---

bitCount

`public int **bitCount**()`

Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit. This method is useful when implementing bit-vector style sets atop

BigIntegers.

Returns:

number of bits in the two's complement representation of this BigInteger that differ from its sign bit.

---

isProbablePrime

```
public boolean isProbablePrime(int certainty)
```

Returns `true` if this BigInteger is probably prime, `false` if it's definitely composite. If `certainty` is `<= 0`, `true` is returned.

Parameters:

`certainty` - a measure of the uncertainty that the caller is willing to tolerate: if the call returns `true` the probability that this BigInteger is prime exceeds `(1 - 1/2^certainty)`. The execution time of this method is proportional to the value of this parameter.

Returns:

`true` if this BigInteger is probably prime, `false` if it's definitely composite.

---

compareTo

```
public int compareTo(BigInteger val)
```

Compares this BigInteger with the specified BigInteger. This method is provided in preference to individual methods for each of the six boolean comparison operators (<, ==, >, >=, !=, <=). The suggested idiom for performing these comparisons is: `(x.compareTo(y)` <*op*> `0)`, where <*op*> is one of the six comparison operators.

Parameters:

`val` - BigInteger to which this BigInteger is to be compared.

Returns:

-1, 0 or 1 as this BigInteger is numerically less than, equal to, or greater than `val`.

---

compareTo

```
public int compareTo(Object o)
```

Compares this BigInteger with the specified Object. If the Object is a BigInteger, this method behaves like `compareTo(BigInteger)`. Otherwise, it throws a `ClassCastException` (as BigIntegers are comparable only to other BigIntegers).

Specified by:

`compareTo` in interface `Comparable`

Parameters:

`o` - Object to which this BigInteger is to be compared.

Returns:

a negative number, zero, or a positive number as this BigInteger is numerically less than, equal to, or greater than `o`, which must be a BigInteger.

Throws:

`ClassCastException` - `o` is not a BigInteger.

Since:

1.2

See Also:

`compareTo(java.math.BigInteger)`, `Comparable`

---

equals

```
public boolean equals(Object x)
```

Compares this BigInteger with the specified Object for equality.

Overrides:

`equals` in class `Object`

Parameters:

`x` - Object to which this BigInteger is to be compared.

Returns:

`true` if and only if the specified Object is a BigInteger whose value is numerically equal to this BigInteger.

See Also:

`Object.hashCode()`, `Hashtable`

---

min

```
public BigInteger min(BigInteger val)
```

Returns the minimum of this BigInteger and `val`.

Parameters:
`val` - value with with the minimum is to be computed.
Returns:
the BigInteger whose value is the lesser of this BigInteger and `val`. If they are equal, either may be returned.

---

max

```
public BigInteger max(BigInteger val)
```

Returns the maximum of this BigInteger and `val`.

Parameters:
`val` - value with with the maximum is to be computed.
Returns:
the BigInteger whose value is the greater of this and `val`. If they are equal, either may be returned.

---

hashCode

```
public int hashCode()
```

Returns the hash code for this BigInteger.

Overrides:
`hashCode` in class `Object`
Returns:
hash code for this BigInteger.
See Also:
`Object.equals(java.lang.Object)`, `Hashtable`

---

toString

```
public String toString(int radix)
```

Returns the String representation of this BigInteger in the given radix. If the radix is outside the range from `Character.MIN_RADIX` to `Character.MAX_RADIX` inclusive, it will default to 10 (as is the case for `Integer.toString`). The digit-to-character mapping provided by `Character.forDigit` is used, and a minus sign is prepended if appropriate. (This representation is compatible with the `(String, int)` constructor.)

Parameters:

`radix` - radix of the String representation.

Returns:

String representation of this BigInteger in the given radix.

See Also:

`Integer.toString(int, int)`, `Character.forDigit(int, int)`, `BigInteger(java.lang.String, int)`

---

toString

```
public String toString()
```

Returns the decimal String representation of this BigInteger. The digit-to-character mapping provided by `Character.forDigit` is used, and a minus sign is prepended if appropriate. (This representation is compatible with the `(String)` constructor, and allows for String concatenation with Java's + operator.)

Overrides:

`toString` in class `Object`

Returns:

decimal String representation of this BigInteger.

See Also:

`Character.forDigit(int, int)`, `BigInteger(java.lang.String)`

---

toByteArray

```
public byte[] toByteArray()
```

Returns a byte array containing the two's-complement representation of this BigInteger. The byte array will be in *big-endian* byte-order: the most significant byte is in the zeroth element. The array will contain the minimum number of bytes required to represent this BigInteger, including at least one sign bit, which is `(ceil((this.bitLength() + 1)/8))`. (This representation is compatible with the `(byte[])` constructor.)

Returns:
a byte array containing the two's-complement representation of this BigInteger.
See Also:
`BigInteger(byte[])`

---

intValue

```
public int intValue()
```

Converts this BigInteger to an `int`. This conversion is analogous to a *narrowing primitive conversion* from `long` to `int` as defined in the Java Language Specification: if this BigInteger is too big to fit in an `int`, only the low-order 32 bits are returned. Note that this conversion can lose information about the overall magnitude of the BigInteger value as well as return a result with the opposite sign.

Specified by:
`intValue` in class `Number`
Returns:
this BigInteger converted to an `int`.

---

longValue

```
public long longValue()
```

Converts this BigInteger to a `long`. This conversion is analogous to a *narrowing primitive conversion* from `long` to `int` as defined in the Java Language Specification: if this BigInteger is too big to fit in a `long`, only the low-order 64 bits are returned. Note that this conversion can lose information about the overall magnitude of the BigInteger value as well as return a result with the opposite sign.

Specified by:
`longValue` in class `Number`
Returns:
this BigInteger converted to a `long`.

---

floatValue

`public float floatValue()`

Converts this BigInteger to a `float`. This conversion is similar to the *narrowing primitive conversion* from `double` to `float` defined in the Java Language Specification: if this BigInteger has too great a magnitude to represent as a `float`, it will be converted to `Float.NEGATIVE_INFINITY` or `Float.POSITIVE_INFINITY` as appropriate. Note that even when the return value is finite, this conversion can lose information about the precision of the BigInteger value.

Specified by:
`floatValue` in class `Number`

Returns:
this BigInteger converted to a `float`.

---

doubleValue

`public double doubleValue()`

Converts this BigInteger to a `double`. This conversion is similar to the *narrowing primitive conversion* from `double` to `float` defined in the Java Language Specification: if this BigInteger has too great a magnitude to represent as a `double`, it will be converted to `Double.NEGATIVE_INFINITY` or `Double.POSITIVE_INFINITY` as appropriate. Note that even when the return value is finite, this conversion can lose information about the precision of the BigInteger value.

Specified by:
`doubleValue` in class `Number`

Returns:
this BigInteger converted to a `double`.

---

documentation redistribution policy.

---

java.lang

Class String

java.lang.Object
  └ **java.lang.String**

All Implemented Interfaces:

CharSequence, Comparable, Serializable

---

public final class **String**
extends Object
implements Serializable, Comparable, CharSequence

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

The class `String` includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. Case mapping relies heavily on the information provided by the Unicode Consortium's Unicode 3.0 specification. The specification's UnicodeData.txt and SpecialCasing.txt files are used extensively to provide case mapping.

The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings. String concatenation is implemented through the `StringBuffer` class and its `append` method. String conversions are implemented through the method `toString`, defined by `Object` and inherited by all classes in Java. For additional information on string concatenation and conversion, see Gosling, Joy, and Steele, *The Java Language Specification*.

Unless otherwise noted, passing a `null` argument to a constructor or method in this class will cause a `NullPointerException` to be thrown.

Since:

JDK1.0

See Also:

`Object.toString()`, `StringBuffer`, `StringBuffer.append(boolean)`, `StringBuffer.append(char)`, `StringBuffer.append(char[])`, `StringBuffer.append(char[], int, int)`, `StringBuffer.append(double)`, `StringBuffer.append(float)`, `StringBuffer.append(int)`, `StringBuffer.append(long)`, `StringBuffer.append(java.lang.Object)`, `StringBuffer.append(java.lang.String)`, `Charset`, Serialized Form

---

Field Summary

`static Comparator`	**`CASE_INSENSITIVE_ORDER`** A Comparator that orders `String` objects as by `compareToIgnoreCase`.

Constructor Summary

`String``()` Initializes a newly created `String` object so that it represents an empty character sequence.
`String``(byte[] bytes)` Constructs a new `String` by decoding the specified array of bytes using the platform's default charset.

String(byte[] ascii, int hibyte) **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the* `String` *constructors that take a charset name or that use the platform's default charset.*
String(byte[] bytes, int offset, int length) Constructs a new `String` by decoding the specified subarray of bytes using the platform's default charset.
String(byte[] ascii, int hibyte, int offset, int count) **Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the* `String` *constructors that take a charset name or that use the platform's default charset.*
String(byte[] bytes, int offset, int length, String charsetName) Constructs a new `String` by decoding the specified subarray of bytes using the specified charset.
String(byte[] bytes, String charsetName) Constructs a new `String` by decoding the specified array of bytes using the specified charset.
String(char[] value) Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument.
String(char[] value, int offset, int count) Allocates a new `String` that contains characters from a subarray of the character array argument.
String(String original) Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.
String(StringBuffer buffer) Allocates a new string that contains the sequence of characters currently contained in the string buffer argument.

Method Summary

char	**charAt**(int index) Returns the character at the specified index.
int	**compareTo**(Object o) Compares this String to another Object.
int	**compareTo**(String anotherString) Compares two strings lexicographically.

int	**compareToIgnoreCase**(String str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(String str) Concatenates the specified string to the end of this string.
boolean	**contentEquals**(StringBuffer sb) Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer.
static String	**copyValueOf**(char[] data) Returns a String that represents the character sequence in the array specified.
static String	**copyValueOf**(char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
boolean	**endsWith**(String suffix) Tests if this string ends with the specified suffix.
boolean	**equals**(Object anObject) Compares this string to the specified object.
boolean	**equalsIgnoreCase**(String anotherString) Compares this String to another String, ignoring case considerations.
byte[]	**getBytes**() Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
void	**getBytes**(int srcBegin, int srcEnd, byte[] dst, int dstBegin) **Deprecated.** *This method does not properly convert characters into bytes. As of JDK 1.1, the preferred way to do this is via the the getBytes() method, which uses the platform's default charset.*
byte[]	**getBytes**(String charsetName) Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
void	**getChars**(int srcBegin, int srcEnd, char[] dst, int dstBegin) Copies characters from this string into the destination character array.
int	**hashCode**() Returns a hash code for this string.
int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.

int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
String	**intern**() Returns a canonical representation for the string object.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(String str) Returns the index within this string of the rightmost occurrence of the specified substring.
int	**lastIndexOf**(String str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
boolean	**matches**(String regex) Tells whether or not this string matches the given regular expression.
boolean	**regionMatches**(boolean ignoreCase, int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
boolean	**regionMatches**(int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
String	**replace**(char oldChar, char newChar) Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

String	**replaceAll**(String regex, String replacement) Replaces each substring of this string that matches the given regular expression with the given replacement.
String	**replaceFirst**(String regex, String replacement) Replaces the first substring of this string that matches the given regular expression with the given replacement.
String[]	**split**(String regex) Splits this string around matches of the given regular expression.
String[]	**split**(String regex, int limit) Splits this string around matches of the given regular expression.
boolean	**startsWith**(String prefix) Tests if this string starts with the specified prefix.
boolean	**startsWith**(String prefix, int toffset) Tests if this string starts with the specified prefix beginning a specified index.
CharSequence	**subSequence**(int beginIndex, int endIndex) Returns a new character sequence that is a subsequence of this sequence.
String	**substring**(int beginIndex) Returns a new string that is a substring of this string.
String	**substring**(int beginIndex, int endIndex) Returns a new string that is a substring of this string.
char[]	**toCharArray**() Converts this string to a new character array.
String	**toLowerCase**() Converts all of the characters in this String to lower case using the rules of the default locale.
String	**toLowerCase**(Locale locale) Converts all of the characters in this String to lower case using the rules of the given Locale.
String	**toString**() This object (which is already a string!) is itself returned.
String	**toUpperCase**() Converts all of the characters in this String to upper case using the rules of the default locale.

String	**toUpperCase**(Locale locale) Converts all of the characters in this String to upper case using the rules of the given Locale.
String	**trim**() Returns a copy of the string, with leading and trailing whitespace omitted.
static String	**valueOf**(boolean b) Returns the string representation of the boolean argument.
static String	**valueOf**(char c) Returns the string representation of the char argument.
static String	**valueOf**(char[] data) Returns the string representation of the char array argument.
static String	**valueOf**(char[] data, int offset, int count) Returns the string representation of a specific subarray of the char array argument.
static String	**valueOf**(double d) Returns the string representation of the double argument.
static String	**valueOf**(float f) Returns the string representation of the float argument.
static String	**valueOf**(int i) Returns the string representation of the int argument.
static String	**valueOf**(long l) Returns the string representation of the long argument.
static String	**valueOf**(Object obj) Returns the string representation of the Object argument.

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

CASE_INSENSITIVE_ORDER

```
public static final Comparator CASE_INSENSITIVE_ORDER
```

A Comparator that orders String objects as by compareToIgnoreCase. This

comparator is serializable.

Note that this Comparator does *not* take locale into account, and will result in an unsatisfactory ordering for certain locales. The java.text package provides *Collators* to allow locale-sensitive ordering.

Since:

1.2

See Also:

`Collator.compare(String, String)`

Constructor Detail

String

```
public String()
```

Initializes a newly created `String` object so that it represents an empty character sequence. Note that use of this constructor is unnecessary since Strings are immutable.

String

```
public String(String original)
```

Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string. Unless an explicit copy of `original` is needed, use of this constructor is unnecessary since Strings are immutable.

Parameters:

`original` - a `String`.

String

```
public String(char[] value)
```

Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument. The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.

Parameters:

`value` - the initial value of the string.

String

```
public String(char[] value,
              int offset,
              int count)
```

Allocates a new `String` that contains characters from a subarray of the character array argument. The `offset` argument is the index of the first character of the subarray and the `count` argument specifies the length of the subarray. The contents of the subarray are copied; subsequent modification of the character array does not affect the newly created string.

Parameters:

`value` - array that is the source of characters.
`offset` - the initial offset.
`count` - the length.

Throws:

`IndexOutOfBoundsException` - if the `offset` and `count` arguments index characters outside the bounds of the `value` array.

String

```
public String(byte[] ascii,
              int hibyte,
              int offset,
              int count)
```

Deprecated. *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the `String` constructors that take a charset name or that use the platform's default charset.*

Allocates a new `String` constructed from a subarray of an array of 8-bit integer values.

The `offset` argument is the index of the first byte of the subarray, and the `count` argument specifies the length of the subarray.

Each `byte` in the subarray is converted to a `char` as specified in the method above.

Parameters:

`ascii` - the bytes to be converted to characters.
`hibyte` - the top 8 bits of each 16-bit Unicode character.
`offset` - the initial offset.
`count` - the length.

Throws:

`IndexOutOfBoundsException` - if the `offset` or `count` argument is invalid.

See Also:

`String(byte[], int)`, `String(byte[], int, int, java.lang.String)`, `String(byte[], int, int)`, `String(byte[], java.lang.String)`, `String(byte[])`

String

```
public String(byte[] ascii,
              int hibyte)
```

Deprecated. *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the `String` constructors that take a charset name or that use the platform's default charset.*

Allocates a new `String` containing characters constructed from an array of 8-bit integer values. Each character *c*in the resulting string is constructed from the corresponding component *b* in the byte array such that:

```
c == (char)(((hibyte & 0xff) << 8)
                         | (b & 0xff))
```

Parameters:

`ascii` - the bytes to be converted to characters.
`hibyte` - the top 8 bits of each 16-bit Unicode character.

See Also:

`String(byte[], int, int, java.lang.String)`, `String(byte[], int, int)`, `String(byte[], java.lang.String)`, `String(byte[])`

String

```
public String(byte[] bytes,
```

```
            int offset,
            int length,
            String charsetName)
      throws UnsupportedEncodingException
```

Constructs a new `String` by decoding the specified subarray of bytes using the specified charset. The length of the new `String` is a function of the charset, and hence may not be equal to the length of the subarray.

The behavior of this constructor when the given bytes are not valid in the given charset is unspecified. The `CharsetDecoder` class should be used when more control over the decoding process is required.

Parameters:

`bytes` - the bytes to be decoded into characters
`offset` - the index of the first byte to decode
`length` - the number of bytes to decode
`charsetName` - the name of a supported charset

Throws:

`UnsupportedEncodingException` - if the named charset is not supported
`IndexOutOfBoundsException` - if the `offset` and `length` arguments index characters outside the bounds of the `bytes` array

Since:

JDK1.1

String

```
public String(byte[] bytes,
            String charsetName)
      throws UnsupportedEncodingException
```

Constructs a new `String` by decoding the specified array of bytes using the specified charset. The length of the new `String` is a function of the charset, and hence may not be equal to the length of the byte array.

The behavior of this constructor when the given bytes are not valid in the given charset is unspecified. The `CharsetDecoder` class should be used when more control over the decoding process is required.

Parameters:

`bytes` - the bytes to be decoded into characters
`charsetName` - the name of a supported charset

Throws:

`UnsupportedEncodingException` - If the named charset is not supported

Since:

JDK1.1

String

```
public String(byte[] bytes,
              int offset,
              int length)
```

Constructs a new `String` by decoding the specified subarray of bytes using the platform's default charset. The length of the new `String` is a function of the charset, and hence may not be equal to the length of the subarray.

The behavior of this constructor when the given bytes are not valid in the default charset is unspecified. The `CharsetDecoder` class should be used when more control over the decoding process is required.

Parameters:

`bytes` - the bytes to be decoded into characters
`offset` - the index of the first byte to decode
`length` - the number of bytes to decode

Throws:

`IndexOutOfBoundsException` - if the `offset` and the `length` arguments index characters outside the bounds of the `bytes` array

Since:

JDK1.1

String

```
public String(byte[] bytes)
```

Constructs a new `String` by decoding the specified array of bytes using the platform's default charset. The length of the new `String` is a function of the charset, and hence may not be equal to the length of the byte array.

The behavior of this constructor when the given bytes are not valid in the default charset is unspecified. The `CharsetDecoder` class should be used when more control over the decoding process is required.

Parameters:
`bytes` - the bytes to be decoded into characters
Since:
JDK1.1

String

`public String(StringBuffer buffer)`

Allocates a new string that contains the sequence of characters currently contained in the string buffer argument. The contents of the string buffer are copied; subsequent modification of the string buffer does not affect the newly created string.

Parameters:
`buffer` - a `StringBuffer`.

Method Detail

length

`public int length()`

Returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string.

Specified by:
`length` in interface `CharSequence`
Returns:
the length of the sequence of characters represented by this object.

---

charAt

`public char charAt(int index)`

Returns the character at the specified index. An index ranges from `0` to `length() - 1`. The first character of the sequence is at index `0`, the next at index `1`, and so on, as for array indexing.

Specified by:

`charAt` in interface `CharSequence`

Parameters:

`index` - the index of the character.

Returns:

the character at the specified index of this string. The first character is at index `0`.

Throws:

`IndexOutOfBoundsException` - if the `index` argument is negative or not less than the length of this string.

---

getChars

```
public void getChars(int srcBegin,
                     int srcEnd,
                     char[] dst,
                     int dstBegin)
```

Copies characters from this string into the destination character array.

The first character to be copied is at index `srcBegin`; the last character to be copied is at index `srcEnd-1` (thus the total number of characters to be copied is `srcEnd-srcBegin`). The characters are copied into the subarray of `dst` starting at index `dstBegin` and ending at index:

```
dstbegin + (srcEnd-srcBegin) - 1
```

Parameters:

`srcBegin` - index of the first character in the string to copy.
`srcEnd` - index after the last character in the string to copy.
`dst` - the destination array.
`dstBegin` - the start offset in the destination array.

Throws:

`IndexOutOfBoundsException` - If any of the following is true:

- `srcBegin` is negative.
- `srcBegin` is greater than `srcEnd`
- `srcEnd` is greater than the length of this string
- `dstBegin` is negative
- `dstBegin+(srcEnd-srcBegin)` is larger than `dst.length`

---

getBytes

```
public void getBytes(int srcBegin,
                     int srcEnd,
                     byte[] dst,
                     int dstBegin)
```

Deprecated. *This method does not properly convert characters into bytes. As of JDK 1.1, the preferred way to do this is via the the* `getBytes()` *method, which uses the platform's default charset.*

Copies characters from this string into the destination byte array. Each byte receives the 8 low-order bits of the corresponding character. The eight high-order bits of each character are not copied and do not participate in the transfer in any way.

The first character to be copied is at index `srcBegin`; the last character to be copied is at index `srcEnd-1`. The total number of characters to be copied is `srcEnd-srcBegin`. The characters, converted to bytes, are copied into the subarray of `dst` starting at index `dstBegin` and ending at index:

```
dstbegin + (srcEnd-srcBegin) - 1
```

Parameters:

`srcBegin` - index of the first character in the string to copy.
`srcEnd` - index after the last character in the string to copy.
`dst` - the destination array.
`dstBegin` - the start offset in the destination array.

Throws:

`IndexOutOfBoundsException` - if any of the following is true:

- `srcBegin` is negative
- `srcBegin` is greater than `srcEnd`
- `srcEnd` is greater than the length of this String
- `dstBegin` is negative
- `dstBegin+(srcEnd-srcBegin)` is larger than `dst.length`

---

getBytes

```
public byte[] getBytes(String charsetName)
                throws UnsupportedEncodingException
```

Encodes this `String` into a sequence of bytes using the named charset, storing the result into

a new byte array.

The behavior of this method when this string cannot be encoded in the given charset is unspecified. The `CharsetEncoder` class should be used when more control over the encoding process is required.

Parameters:
`charsetName` - the name of a supported charset
Returns:
The resultant byte array
Throws:
`UnsupportedEncodingException` - If the named charset is not supported
Since:
JDK1.1

---

getBytes

```
public byte[] getBytes()
```

Encodes this `String` into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

The behavior of this method when this string cannot be encoded in the default charset is unspecified. The `CharsetEncoder` class should be used when more control over the encoding process is required.

Returns:
The resultant byte array
Since:
JDK1.1

---

equals

```
public boolean equals(Object anObject)
```

Compares this string to the specified object. The result is `true` if and only if the argument is not `null` and is a `String` object that represents the same sequence of characters as this object.

Overrides:

`equals` in class `Object`

Parameters:

`anObject` - the object to compare this `String` against.

Returns:

`true` if the `String` are equal; `false` otherwise.

See Also:

`compareTo(java.lang.String)`, `equalsIgnoreCase(java.lang.String)`

---

contentEquals

`public boolean contentEquals(StringBuffer sb)`

Returns `true` if and only if this `String` represents the same sequence of characters as the specified `StringBuffer`.

Parameters:

`sb` - the `StringBuffer` to compare to.

Returns:

`true` if and only if this `String` represents the same sequence of characters as the specified `StringBuffer`, otherwise `false`.

Since:

1.4

---

equalsIgnoreCase

`public boolean equalsIgnoreCase(String anotherString)`

Compares this `String` to another `String`, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

Two characters `c1` and `c2` are considered the same, ignoring case if at least one of the following is true:

- The two characters are the same (as compared by the == operator).
- Applying the method `Character.toUpperCase(char)` to each character produces the same result.
- Applying the method `Character.toLowerCase(char)` to each character

produces the same result.

Parameters:

`anotherString` - the `String` to compare this `String` against.

Returns:

`true` if the argument is not `null` and the `String`s are equal, ignoring case; `false` otherwise.

See Also:

`equals(Object)`, `Character.toLowerCase(char)`, `Character.toUpperCase(char)`

---

compareTo

```
public int compareTo(String anotherString)
```

Compares two strings lexicographically. The comparison is based on the Unicode value of each character in the strings. The character sequence represented by this `String` object is compared lexicographically to the character sequence represented by the argument string. The result is a negative integer if this `String` object lexicographically precedes the argument string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns `0` exactly when the `equals(Object)` method would return `true`.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller value, as determined by using the < operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position `k` in the two string -- that is, the value:

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings -- that is, the value:

```
this.length()-anotherString.length()
```

Parameters:

`anotherString` - the `String` to be compared.

Returns:

the value `0` if the argument string is equal to this string; a value less than `0` if this string is lexicographically less than the string argument; and a value greater than `0` if this string is lexicographically greater than the string argument.

---

compareTo

```
public int compareTo(Object o)
```

Compares this String to another Object. If the Object is a String, this function behaves like `compareTo(String)`. Otherwise, it throws a `ClassCastException` (as Strings are comparable only to other Strings).

Specified by:

`compareTo` in interface `Comparable`

Parameters:

`o` - the `Object` to be compared.

Returns:

the value `0` if the argument is a string lexicographically equal to this string; a value less than `0` if the argument is a string lexicographically greater than this string; and a value greater than `0` if the argument is a string lexicographically less than this string.

Throws:

`ClassCastException` - if the argument is not a `String`.

Since:

1.2

See Also:

`Comparable`

---

compareToIgnoreCase

```
public int compareToIgnoreCase(String str)
```

Compares two strings lexicographically, ignoring case differences. This method returns an integer whose sign is that of calling `compareTo` with normalized versions of the strings where case differences have been eliminated by calling `Character.toLowerCase(Character.toUpperCase(character))` on each character.

Note that this method does *not* take locale into account, and will result in an unsatisfactory ordering for certain locales. The java.text package provides *collators* to allow locale-sensitive ordering.

Parameters:

`str` - the `String` to be compared.

Returns:

a negative integer, zero, or a positive integer as the the specified String is greater than, equal to, or less than this String, ignoring case considerations.

Since:

1.2

See Also:

`Collator.compare(String, String)`

---

regionMatches

```
public boolean regionMatches(int toffset,
                             String other,
                             int ooffset,
                             int len)
```

Tests if two string regions are equal.

A substring of this `String` object is compared to a substring of the argument other. The result is true if these substrings represent identical character sequences. The substring of this `String` object to be compared begins at index `toffset` and has length `len`. The substring of other to be compared begins at index `ooffset` and has length `len`. The result is `false` if and only if at least one of the following is true:

- `toffset` is negative.
- `ooffset` is negative.
- `toffset+len` is greater than the length of this `String` object.
- `ooffset+len` is greater than the length of the other argument.
- There is some nonnegative integer *k* less than `len` such that: `this.charAt(toffset+`*k*`) != other.charAt(ooffset+`*k*`)`

Parameters:

`toffset` - the starting offset of the subregion in this string.
`other` - the string argument.
`ooffset` - the starting offset of the subregion in the string argument.
`len` - the number of characters to compare.

Returns:

`true` if the specified subregion of this string exactly matches the specified subregion of the string argument; `false` otherwise.

---

regionMatches

```
public boolean regionMatches(boolean ignoreCase,
                             int toffset,
                             String other,
                             int ooffset,
                             int len)
```

Tests if two string regions are equal.

A substring of this `String` object is compared to a substring of the argument `other`. The result is `true` if these substrings represent character sequences that are the same, ignoring case if and only if `ignoreCase` is true. The substring of this `String` object to be compared begins at index `toffset` and has length `len`. The substring of `other` to be compared begins at index `ooffset` and has length `len`. The result is `false` if and only if at least one of the following is true:

- `toffset` is negative.
- `ooffset` is negative.
- `toffset+len` is greater than the length of this `String` object.
- `ooffset+len` is greater than the length of the other argument.
- `ignoreCase` is `false` and there is some nonnegative integer *k* less than `len` such that:

```
this.charAt(toffset+k) != other.charAt(ooffset
+k)
```

- `ignoreCase` is `true` and there is some nonnegative integer *k* less than `len` such that:

```
Character.toLowerCase(this.charAt(toffset+k)) !=
                  Character.toLowerCase(other.charAt
(ooffset+k))
```

and:

```
Character.toUpperCase(this.charAt(toffset+k)) !=
          Character.toUpperCase(other.charAt
(ooffset+k))
```

Parameters:

`ignoreCase` - if `true`, ignore case when comparing characters.
`toffset` - the starting offset of the subregion in this string.
`other` - the string argument.
`ooffset` - the starting offset of the subregion in the string argument.
`len` - the number of characters to compare.

Returns:

`true` if the specified subregion of this string matches the specified subregion of the string argument; `false` otherwise. Whether the matching is exact or case insensitive depends on the `ignoreCase` argument.

---

startsWith

```
public boolean startsWith(String prefix,
                          int toffset)
```

Tests if this string starts with the specified prefix beginning a specified index.

Parameters:

`prefix` - the prefix.
`toffset` - where to begin looking in the string.

Returns:

`true` if the character sequence represented by the argument is a prefix of the substring of this object starting at index `toffset`; `false` otherwise. The result is `false` if `toffset` is negative or greater than the length of this `String` object; otherwise the result is the same as the result of the expression

```
this.subString(toffset).startsWith(prefix)
```

---

startsWith

```
public boolean startsWith(String prefix)
```

Tests if this string starts with the specified prefix.

Parameters:

`prefix` - the prefix.

Returns:

`true` if the character sequence represented by the argument is a prefix of the character sequence represented by this string; `false` otherwise. Note also that `true` will be

returned if the argument is an empty string or is equal to this `String` object as determined by the `equals(Object)` method.

Since:

1. 0

---

endsWith

`public boolean **endsWith**(String suffix)`

Tests if this string ends with the specified suffix.

Parameters:

`suffix` - the suffix.

Returns:

`true` if the character sequence represented by the argument is a suffix of the character sequence represented by this object; `false` otherwise. Note that the result will be `true` if the argument is the empty string or is equal to this `String` object as determined by the `equals(Object)` method.

---

hashCode

`public int **hashCode**()`

Returns a hash code for this string. The hash code for a `String` object is computed as

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

using `int` arithmetic, where `s[i]` is the *i*th character of the string, `n` is the length of the string, and `^` indicates exponentiation. (The hash value of the empty string is zero.)

Overrides:

`hashCode` in class `Object`

Returns:

a hash code value for this object.

See Also:

`Object.equals(java.lang.Object)`, `Hashtable`

---

indexOf

```
public int indexOf(int ch)
```

Returns the index within this string of the first occurrence of the specified character. If a character with value `ch` occurs in the character sequence represented by this `String` object, then the index of the first such occurrence is returned -- that is, the smallest value *k* such that:

```
this.charAt(k) == ch
```

is `true`. If no such character occurs in this string, then `-1`is returned.

Parameters:

`ch` - a character.

Returns:

the index of the first occurrence of the character in the character sequence represented by this object, or `-1` if the character does not occur.

---

indexOf

```
public int indexOf(int ch,
                   int fromIndex)
```

Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.

If a character with value `ch` occurs in the character sequence represented by this `String` object at an index no smaller than `fromIndex`, then the index of the first such occurrence is returned--that is, the smallest value *k* such that:

```
(this.charAt(k) == ch) && (k >= fromIndex)
```

is true. If no such character occurs in this string at or after position `fromIndex`, then `-1` is returned.

There is no restriction on the value of `fromIndex`. If it is negative, it has the same effect as if it were zero: this entire string may be searched. If it is greater than the length of this string, it has the same effect as if it were equal to the length of this string: `-1` is returned.

Parameters:

`ch` - a character.
`fromIndex` - the index to start the search from.

Returns:

the index of the first occurrence of the character in the character sequence represented by this object that is greater than or equal to `fromIndex`, or `-1` if the character does not occur.

---

lastIndexOf

```
public int lastIndexOf(int ch)
```

Returns the index within this string of the last occurrence of the specified character. That is, the index returned is the largest value *k* such that:

```
this.charAt(k) == ch
```

is true. The String is searched backwards starting at the last character.

Parameters:

`ch` - a character.

Returns:

the index of the last occurrence of the character in the character sequence represented by this object, or `-1` if the character does not occur.

---

lastIndexOf

```
public int lastIndexOf(int ch,
                       int fromIndex)
```

Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index. That is, the index returned is the largest value *k* such that:

```
this.charAt(k) == ch) && (k <= fromIndex)
```

is true.

Parameters:

`ch` - a character.
`fromIndex` - the index to start the search from. There is no restriction on the value of `fromIndex`. If it is greater than or equal to the length of this string, it has the same effect as if it were equal to one less than the length of this string: this entire string may be searched. If it is negative, it has the same effect as if it were -1: -1 is returned.

Returns:

the index of the last occurrence of the character in the character sequence represented by this object that is less than or equal to `fromIndex`, or `-1` if the character does not occur before that point.

---

indexOf

```
public int indexOf(String str)
```

Returns the index within this string of the first occurrence of the specified substring. The integer returned is the smallest value *k* such that:

```
this.startsWith(str, k)
```

is `true`.

Parameters:

`str` - any string.

Returns:

if the string argument occurs as a substring within this object, then the index of the first character of the first such substring is returned; if it does not occur as a substring, `-1` is returned.

---

indexOf

```
public int indexOf(String str,
                   int fromIndex)
```

Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. The integer returned is the smallest value `k` for which:

```
k >= Math.min(fromIndex, str.length()) && this.
startsWith(str, k)
```

If no such value of *k*exists, then -1 is returned.

Parameters:

`str` - the substring for which to search.

`fromIndex` - the index from which to start the search.

Returns:

the index within this string of the first occurrence of the specified substring, starting at the specified index.

---

lastIndexOf

```
public int lastIndexOf(String str)
```

Returns the index within this string of the rightmost occurrence of the specified substring. The rightmost empty string "" is considered to occur at the index value `this.length()`. The returned index is the largest value *k* such that

```
this.startsWith(str, k)
```

is true.

Parameters:

`str` - the substring to search for.

Returns:

if the string argument occurs one or more times as a substring within this object, then the index of the first character of the last such substring is returned. If it does not occur as a substring, `-1` is returned.

---

lastIndexOf

```
public int lastIndexOf(String str,
                       int fromIndex)
```

Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index. The integer returned is the largest value *k* such that:

```
k <= Math.min(fromIndex, str.length()) && this.
startsWith(str, k)
```

If no such value of *k*exists, then -1 is returned.

Parameters:
`str` - the substring to search for.
`fromIndex` - the index to start the search from.
Returns:
the index within this string of the last occurrence of the specified substring.

---

substring

```
public String substring(int beginIndex)
```

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Examples:

```
"unhappy".substring(2) returns "happy"
"Harbison".substring(3) returns "bison"
"emptiness".substring(9) returns "" (an empty string)
```

Parameters:
`beginIndex` - the beginning index, inclusive.
Returns:
the specified substring.
Throws:
`IndexOutOfBoundsException` - if `beginIndex` is negative or larger than the length of this `String` object.

---

substring

```
public String substring(int beginIndex,
                        int endIndex)
```

Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex-beginIndex`.

Examples:

```
"hamburger".substring(4, 8) returns "urge"
"smiles".substring(1, 5) returns "mile"
```

Parameters:

`beginIndex` - the beginning index, inclusive.
`endIndex` - the ending index, exclusive.

Returns:

the specified substring.

Throws:

`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.

---

subSequence

```
public CharSequence subSequence(int beginIndex,
                                int endIndex)
```

Returns a new character sequence that is a subsequence of this sequence.

An invocation of this method of the form

```
str.subSequence(begin, end)
```

behaves in exactly the same way as the invocation

```
str.substring(begin, end)
```

This method is defined so that the `String`class can implement the `CharSequence`interface.

Specified by:

`subSequence` in interface `CharSequence`

Parameters:

`beginIndex` - the begin index, inclusive.
`endIndex` - the end index, exclusive.

Returns:

the specified subsequence.

Throws:

`IndexOutOfBoundsException` - if `beginIndex` or `endIndex` are negative, if

`endIndex` is greater than `length()`, or if `beginIndex` is greater than `startIndex`

Since:

1.4

---

concat

```
public String concat(String str)
```

Concatenates the specified string to the end of this string.

If the length of the argument string is `0`, then this `String` object is returned. Otherwise, a new `String` object is created, representing a character sequence that is the concatenation of the character sequence represented by this `String` object and the character sequence represented by the argument string.

Examples:

```
"cares".concat("s") returns "caress"
"to".concat("get").concat("her") returns "together"
```

Parameters:

`str` - the `String` that is concatenated to the end of this `String`.

Returns:

a string that represents the concatenation of this object's characters followed by the string argument's characters.

---

replace

```
public String replace(char oldChar,
                      char newChar)
```

Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

If the character `oldChar` does not occur in the character sequence represented by this `String` object, then a reference to this `String` object is returned. Otherwise, a new `String` object is created that represents a character sequence identical to the character sequence represented by this `String` object, except that every occurrence of `oldChar` is

replaced by an occurrence of `newChar`.

Examples:

```
"mesquite in your cellar".replace('e', 'o')
        returns "mosquito in your collar"
"the war of baronets".replace('r', 'y')
        returns "the way of bayonets"
"sparring with a purple porpoise".replace('p', 't')
        returns "starring with a turtle tortoise"
"JonL".replace('q', 'x') returns "JonL" (no change)
```

Parameters:

`oldChar` - the old character.
`newChar` - the new character.

Returns:

a string derived from this string by replacing every occurrence of `oldChar` with `newChar`.

---

matches

`public boolean` **`matches`**`(String regex)`

Tells whether or not this string matches the given regular expression.

An invocation of this method of the form *str*`.matches(`*regex*`)` yields exactly the same result as the expression

`Pattern.matches(`*regex*`, `*str*`)`

Parameters:

`regex` - the regular expression to which this string is to be matched

Returns:

`true` if, and only if, this string matches the given regular expression

Throws:

`PatternSyntaxException` - if the regular expression's syntax is invalid

Since:

1.4

See Also:

`Pattern`

---

replaceFirst

```
public String replaceFirst(String regex,
                           String replacement)
```

Replaces the first substring of this string that matches the given regular expression with the given replacement.

An invocation of this method of the form *str*`.replaceFirst(`*regex*`, `*repl*`)` yields exactly the same result as the expression

`Pattern.compile(`*regex*`).matcher(`*str*`).replaceFirst(`*repl*`)`

Parameters:
`regex` - the regular expression to which this string is to be matched
Returns:
The resulting `String`
Throws:
`PatternSyntaxException` - if the regular expression's syntax is invalid
Since:
1.4
See Also:
`Pattern`

---

replaceAll

```
public String replaceAll(String regex,
                         String replacement)
```

Replaces each substring of this string that matches the given regular expression with the given replacement.

An invocation of this method of the form *str*`.replaceAll(`*regex*`, `*repl*`)` yields exactly the same result as the expression

`Pattern.compile(`*regex*`).matcher(`*str*`).replaceAll(`*repl*`)`

Parameters:
`regex` - the regular expression to which this string is to be matched

Returns:

The resulting `String`

Throws:

`PatternSyntaxException` - if the regular expression's syntax is invalid

Since:

1.4

See Also:

`Pattern`

---

split

```
public String[] split(String regex,
                      int limit)
```

Splits this string around matches of the given regular expression.

The array returned by this method contains each substring of this string that is terminated by another substring that matches the given expression or is terminated by the end of the string. The substrings in the array are in the order in which they occur in this string. If the expression does not match any part of the input then the resulting array has just one element, namely this string.

The `limit` parameter controls the number of times the pattern is applied and therefore affects the length of the resulting array. If the limit *n* is greater than zero then the pattern will be applied at most *n* - 1 times, the array's length will be no greater than *n*, and the array's last entry will contain all input beyond the last matched delimiter. If *n* is non-positive then the pattern will be applied as many times as possible and the array can have any length. If *n* is zero then the pattern will be applied as many times as possible, the array can have any length, and trailing empty strings will be discarded.

The string `"boo:and:foo"`, for example, yields the following results with these parameters:

Regex	Limit	Result
:	2	`{ "boo", "and:foo" }`
:	5	`{ "boo", "and", "foo" }`
:	-2	`{ "boo", "and", "foo" }`
o	5	`{ "b", "", ":and:f", "", "" }`
o	-2	`{ "b", "", ":and:f", "", "" }`
o	0	`{ "b", "", ":and:f" }`

An invocation of this method of the form *str*.`split(`*regex*`, `*n*`)` yields the same result as the expression

`Pattern.compile(`*regex*`).split(`*str*`, `*n*`)`

Parameters:

`regex` - the delimiting regular expression
`limit` - the result threshold, as described above

Returns:

the array of strings computed by splitting this string around matches of the given regular expression

Throws:

`PatternSyntaxException` - if the regular expression's syntax is invalid

Since:

1.4

See Also:

`Pattern`

---

split

```
public String[] split(String regex)
```

Splits this string around matches of the given regular expression.

This method works as if by invoking the two-argument `split` method with the given expression and a limit argument of zero. Trailing empty strings are therefore not included in the resulting array.

The string `"boo:and:foo"`, for example, yields the following results with these expressions:

Regex	Result
:	`{ "boo", "and", "foo" }`
o	`{ "b", "", ":and:f" }`

Parameters:

`regex` - the delimiting regular expression

Returns:

the array of strings computed by splitting this string around matches of the given regular expression

Throws:

`PatternSyntaxException` - if the regular expression's syntax is invalid

Since:

1.4

See Also:

`Pattern`

---

toLowerCase

```
public String toLowerCase(Locale locale)
```

Converts all of the characters in this `String` to lower case using the rules of the given `Locale`. Case mappings rely heavily on the Unicode specification's character data. Since case mappings are not always 1:1 char mappings, the resulting `String` may be a different length than the original `String`.

Examples of lowercase mappings are in the following table:

Language Code of Locale	Upper Case	Lower Case	Description
tr (Turkish)	\u0130	\u0069	capital letter I with dot above -> small letter i
tr (Turkish)	\u0049	\u0131	capital letter I -> small letter dotless i
(all)	French Fries	french fries	lowercased all chars in String
(all)	ΙΧΘΥΣ	ιχθυς	lowercased all chars in String

Parameters:

`locale` - use the case transformation rules for this locale

Returns:

the `String`, converted to lowercase.

Since:

1.1

See Also:

`toLowerCase()`, `toUpperCase()`, `toUpperCase(Locale)`

---

toLowerCase

```
public String toLowerCase()
```

Converts all of the characters in this `String` to lower case using the rules of the default

locale. This is equivalent to calling `toLowerCase(Locale.getDefault())`.

Returns:

the `String`, converted to lowercase.

See Also:

`toLowerCase(Locale)`

---

toUpperCase

`public String` **`toUpperCase`**`(Locale locale)`

Converts all of the characters in this `String` to upper case using the rules of the given `Locale`. Case mappings rely heavily on the Unicode specification's character data. Since case mappings are not always 1:1 char mappings, the resulting `String` may be a different length than the original `String`.

Examples of locale-sensitive and 1:M case mappings are in the following table.

Language Code of Locale	Lower Case	Upper Case	Description
tr (Turkish)	\u0069	\u0130	small letter i -> capital letter I with dot above
tr (Turkish)	\u0131	\u0049	small letter dotless i -> capital letter I
(all)	\u00df	\u0053 \u0053	small letter sharp s -> two letters: SS
(all)	Fahrvergnügen	FAHRVERGNÜGEN	

Parameters:

`locale` - use the case transformation rules for this locale

Returns:

the `String`, converted to uppercase.

Since:

1.1

See Also:

`toUpperCase()`, `toLowerCase()`, `toLowerCase(Locale)`

---

toUpperCase

`public String toUpperCase()`

Converts all of the characters in this `String` to upper case using the rules of the default locale. This method is equivalent to `toUpperCase(Locale.getDefault())`.

Returns:
the `String`, converted to uppercase.
See Also:
`toUpperCase(Locale)`

---

trim

`public String trim()`

Returns a copy of the string, with leading and trailing whitespace omitted.

If this `String` object represents an empty character sequence, or the first and last characters of character sequence represented by this `String` object both have codes greater than `'\u0020'` (the space character), then a reference to this `String` object is returned.

Otherwise, if there is no character with a code greater than `'\u0020'` in the string, then a new `String` object representing an empty string is created and returned.

Otherwise, let *k* be the index of the first character in the string whose code is greater than `'\u0020'`, and let *m* be the index of the last character in the string whose code is greater than `'\u0020'`. A new `String` object is created, representing the substring of this string that begins with the character at index *k* and ends with the character at index *m*-that is, the result of `this.substring(k, m+1)`.

This method may be used to trim `whitespace` from the beginning and end of a string; in fact, it trims all ASCII control characters as well.

Returns:
A copy of this string with leading and trailing white space removed, or this string if it has no leading or trailing white space.

---

toString

`public String toString()`

This object (which is already a string!) is itself returned.

Specified by:
`toString` in interface `CharSequence`
Overrides:
`toString` in class `Object`
Returns:
the string itself.

---

toCharArray

```
public char[] toCharArray()
```

Converts this string to a new character array.

Returns:
a newly allocated character array whose length is the length of this string and whose contents are initialized to contain the character sequence represented by this string.

---

valueOf

```
public static String valueOf(Object obj)
```

Returns the string representation of the `Object` argument.

Parameters:
`obj` - an `Object`.
Returns:
if the argument is `null`, then a string equal to `"null"`; otherwise, the value of `obj.toString()` is returned.
See Also:
`Object.toString()`

---

valueOf

```
public static String valueOf(char[] data)
```

Returns the string representation of the `char` array argument. The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.

Parameters:

`data` - a `char` array.

Returns:

a newly allocated string representing the same sequence of characters contained in the character array argument.

---

valueOf

```
public static String valueOf(char[] data,
                             int offset,
                             int count)
```

Returns the string representation of a specific subarray of the `char` array argument.

The `offset` argument is the index of the first character of the subarray. The `count` argument specifies the length of the subarray. The contents of the subarray are copied; subsequent modification of the character array does not affect the newly created string.

Parameters:

`data` - the character array.

`offset` - the initial offset into the value of the `String`.

`count` - the length of the value of the `String`.

Returns:

a string representing the sequence of characters contained in the subarray of the character array argument.

Throws:

`IndexOutOfBoundsException` - if `offset` is negative, or `count` is negative, or `offset+count` is larger than `data.length`.

---

copyValueOf

```
public static String copyValueOf(char[] data,
                                 int offset,
                                 int count)
```

Returns a String that represents the character sequence in the array specified.

Parameters:
`data` - the character array.
`offset` - initial offset of the subarray.
`count` - length of the subarray.
Returns:
a `String` that contains the characters of the specified subarray of the character array.

---

copyValueOf

```
public static String copyValueOf(char[] data)
```

Returns a String that represents the character sequence in the array specified.

Parameters:
`data` - the character array.
Returns:
a `String` that contains the characters of the character array.

---

valueOf

```
public static String valueOf(boolean b)
```

Returns the string representation of the `boolean` argument.

Parameters:
`b` - a `boolean`.
Returns:
if the argument is `true`, a string equal to `"true"` is returned; otherwise, a string equal to `"false"` is returned.

---

valueOf

```
public static String valueOf(char c)
```

Returns the string representation of the `char` argument.

Parameters:
`c` - a `char`.
Returns:
a string of length `1` containing as its single character the argument `c`.

---

valueOf

```
public static String valueOf(int i)
```

Returns the string representation of the `int` argument.

The representation is exactly the one returned by the `Integer.toString` method of one argument.

Parameters:
`i` - an `int`.
Returns:
a string representation of the `int` argument.
See Also:
`Integer.toString(int, int)`

---

valueOf

```
public static String valueOf(long l)
```

Returns the string representation of the `long` argument.

The representation is exactly the one returned by the `Long.toString` method of one argument.

Parameters:
`l` - a `long`.
Returns:
a string representation of the `long` argument.
See Also:
`Long.toString(long)`

---

valueOf

```
public static String valueOf(float f)
```

Returns the string representation of the `float` argument.

The representation is exactly the one returned by the `Float.toString` method of one argument.

Parameters:
`f` - a `float`.
Returns:
a string representation of the `float` argument.
See Also:
`Float.toString(float)`

---

valueOf

```
public static String valueOf(double d)
```

Returns the string representation of the `double` argument.

The representation is exactly the one returned by the `Double.toString` method of one argument.

Parameters:
`d` - a `double`.
Returns:
a string representation of the `double` argument.
See Also:
`Double.toString(double)`

---

intern

```
public String intern()
```

Returns a canonical representation for the string object.

A pool of strings, initially empty, is maintained privately by the class `String`.

When the intern method is invoked, if the pool already contains a string equal to this `String` object as determined by the `equals(Object)` method, then the string from the pool is returned. Otherwise, this `String` object is added to the pool and a reference to this `String` object is returned.

It follows that for any two strings `s` and `t`, `s.intern() == t.intern()` is `true` if and only if `s.equals(t)` is `true`.

All literal strings and string-valued constant expressions are interned. String literals are defined in §3.10.5 of the Java Language Specification

Returns:

a string that has the same contents as this string, but is guaranteed to be from a pool of unique strings.

SOAP Version 1.2 Part 0: Primer

W3C Recommendation 24 June 2003

This version:
http://www.w3.org/TR/2003/REC-soap12-part0-20030624/
Latest version:
http://www.w3.org/TR/soap12-part0/
Previous version:
http://www.w3.org/TR/2003/PR-soap12-part0-20030507/
Editor:
Nilo Mitra (Ericsson)

Please refer to the **errata** for this document, which may include some normative corrections.

The English version of this specification is the only normative version. Non-normative translations may also be available.

Abstract

SOAP Version 1.2 Part 0: Primer is a non-normative document intended to provide an easily understandable tutorial on the features of the SOAP Version 1.2 specifications. In particular, it describes the features through various usage scenarios, and is intended to complement the normative text contained in Part 1 and Part 2 of the SOAP 1.2 specifications.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document is a Recommendation of the W3C. This document has been produced by the XML Protocol Working Group, which is part of the Web Services Activity. It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

Comments on this document are welcome. Please send them to the public mailing-list xmlp-comments@w3.org (archive). It is inappropriate to send discussion email to this address.

Information about implementations relevant to this specification can be found in the Implementation Report at http://www.w3.org/2000/xp/Group/2/03/soap1.2implementation.html.

Patent disclosures relevant to this specification may be found on the Working Group's patent disclosure page, in conformance with W3C policy.

A list of current W3C Recommendations and other technical reports can be found at http://www.w3.org/TR.

Table of Contents

1. Introduction

SOAP Version 1.2 Part 0: Primer is a non-normative document intended to provide an easily understandable tutorial on the features of the SOAP Version 1.2 specifications. Its purpose is to help a technically competent person understand how SOAP may be used, by describing representative SOAP message structures and message exchange patterns.

In particular, this primer describes the features of SOAP through various usage scenarios, and is intended to complement the normative text contained in SOAP Version 1.2 Part 1: Messaging Framework (hereafter [SOAP Part1]) and SOAP Version 1.2 Part 2: Adjuncts (hereafter [SOAP Part2]) of the SOAP Version 1.2 specifications.

It is expected that the reader has some familiarity with the basic syntax of XML, including the use of XML namespaces and infosets, and Web concepts such as URIs and HTTP. It is intended primarily for users of SOAP, such as application designers, rather than implementors of the SOAP specifications, although the latter may derive some benefit. This primer aims at highlighting the essential features of SOAP Version 1.2, not at completeness in describing every nuance or edge case. Therefore, there is no substitute for the main specifications to obtain a fuller understanding of SOAP. To that end, this primer provides extensive links to the main specifications wherever new concepts are introduced or used.

[SOAP Part1] defines the SOAP envelope, which is a construct that defines an overall framework for representing the contents of a SOAP message, identifying who should deal with all or part of it, and whether handling such

parts are optional or mandatory. It also defines a protocol binding framework, which describes how the specification for a binding of SOAP onto another underlying protocol may be written.

[SOAP Part2] defines a data model for SOAP, a particular encoding scheme for data types which may be used for conveying remote procedure calls (RPC), as well as one concrete realization of the underlying protocol binding framework defined in [SOAP Part1]. This binding allows the exchange of SOAP messages either as payload of a HTTP POST request and response, or as a SOAP message in the response to a HTTP GET.

This document (the primer) is not normative, which means that it does not provide the definitive specification of SOAP Version 1.2. The examples provided here are intended to complement the formal specifications, and in any question of interpretation the formal specifications naturally take precedence. The examples shown here provide a subset of the uses expected for SOAP. In actual usage scenarios, SOAP will most likely be a part of an overall solution, and there will no doubt be other application-specific requirements which are not captured in these examples.

1.1 Overview

SOAP Version 1.2 provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment. [SOAP Part1] explains that a SOAP message is formally specified as an XML Infoset [XML Infoset], which provides an abstract description of its contents. Infosets can have different on-the-wire representations, one common example of which is as an XML 1.0 [XML 1.0] document.

SOAP is fundamentally a stateless, one-way message exchange paradigm, but applications can create more complex interaction patterns (e.g., request/response, request/multiple responses, etc.) by combining such one-way exchanges with features provided by an underlying protocol and/or application-specific information. SOAP is silent on the semantics of any application-specific data it conveys, as it is on issues such as the routing of SOAP messages, reliable data transfer, firewall traversal, etc. However, SOAP provides the framework by which application-specific information may be conveyed in an extensible manner. Also, SOAP provides a full description of the required actions taken by a SOAP node on receiving a SOAP message.

Section 2 of this document provides an introduction to the basic features of SOAP starting with the simplest usage scenarios, namely a one-way SOAP message, followed by various request-response type exchanges, including

RPCs. Fault situations are also described.

Section 3 provides an overview of the SOAP processing model, which describes the rules for initial construction of a message, rules by which messages are processed when received at an intermediary or ultimate destination, and rules by which portions of the message can be inserted, deleted or modified by the actions of an intermediary.

Section 4 of this document describes the ways in which SOAP messages may be transported to realize various usage scenarios. It describes the SOAP HTTP binding specified in [SOAP Part2], as well as an example of how SOAP messages may be conveyed in email messages. As a part of the HTTP binding, it introduces two message exchange patterns which are available to an application, one of which uses the HTTP POST method, while the other uses HTTP GET. Examples are also provided on how RPCs, in particular those that represent "safe" information retrieval, may be represented in SOAP message exchanges in a manner that is compatible with the architectural principles of the World Wide Web .

Section 5 of this document provides a treatment of various aspects of SOAP that can be used in more complex usage scenarios. These include the extensibility mechanism offered through the use of header elements, which may be targeted at specific intermediate SOAP nodes to provide value-added services to communicating applications, and using various encoding schemes to serialize application-specific data in SOAP messages.

Section 6 of this document describes the changes from SOAP Version 1.1 [SOAP 1.1].

Section 7 of this document provides references.

For ease of reference, terms and concepts used in this primer are hyper-linked to their definition in the main specifications.

1.2 Notational Conventions

Throughout this primer, sample SOAP envelopes and messages are shown as [XML 1.0] documents. [SOAP Part1] explains that a SOAP message is formally specified as an [XML InfoSet], which is an abstract description of its contents. The distinction between the SOAP XML Infosets and the documents is unlikely to be of interest to those using this primer as an introduction to SOAP; those who do care (typically those who port SOAP to new protocol bindings where the messages may have alternative representations) should understand these examples as referring to the corresponding XML infosets.

Further elaboration of this point is provided in Section 4 of this document.

The namespace prefixes "env", "enc", and "rpc" used in the prose sections of this document are associated with the SOAP namespace names "http://www.w3.org/2003/05/soap-envelope" , "http://www.w3.org/2003/05/soap-encoding", and "http://www.w3.org/2003/05/soap-rpc" respectively.

The namespace prefixes "xs" and "xsi" used in the prose sections of this document are associated with the namespace names "http://www.w3.org/2001/XMLSchema" and "http://www.w3.org/2001/XMLSchema-instance" respectively, both of which are defined in the XML Schema specifications [XML Schema Part1], [XML Schema Part2].

Note that the choice of any other namespace prefix is arbitrary and not semantically significant.

Namespace URIs of the general form "http://example.org/..." and "http://example.com/..." represent an application-dependent or context-dependent URI [RFC 2396].

2. Basic Usage Scenarios

A SOAP message is fundamentally a one-way transmission between SOAP nodes, from a SOAP sender to a SOAP receiver, but SOAP messages are expected to be combined by applications to implement more complex interaction patterns ranging from request/response to multiple, back-and-forth "conversational" exchanges.

The primer starts by exposing the structure of a SOAP message and its exchange in some simple usage scenarios based on a travel reservation application. Various aspects of this application scenario will be used throughout the primer. In this scenario, the travel reservation application for an employee of a company negotiates a travel reservation with a travel booking service for a planned trip. The information exchanged between the travel reservation application and the travel service application is in the form of SOAP messages.

The ultimate recipient of a SOAP message sent from the travel reservation application is the travel service application, but it is possible that the SOAP message may be "routed" through one or more SOAP intermediaries which act in some way on the message. Some simple examples of such SOAP intermediaries might be ones that log, audit or, possibly, amend each travel request. Examples, and a more detailed discussion of the behavior and role of

SOAP intermediaries, is postponed to section 5.1.

Section 2.1 describes a travel reservation request expressed as a SOAP message, which offers the opportunity to describe the various "parts" of a SOAP message.

Section 2.2.1 continues the same scenario to show a response from the travel service in the form of another SOAP message, which forms a part of a conversational message exchange as the various choices meeting the constraints of the travel request are negotiated.

Section 2.2.2 assumes that the various parameters of the travel reservation have been accepted by the traveller, and an exchange - modelled as a remote procedure call (RPC) - between the travel reservation and the travel service applications confirms the payment for the reservation.

Section 2.3 shows examples of fault handling.

2.1 SOAP Messages

Example 1 shows data for a travel reservation expressed in a SOAP message.

Example 1

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.
example.org/reservation"
          env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-
pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:
dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/
employees"
          env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Åke Jógvan Øyvind</n:name>
```

```
    </n:passenger>
   </env:Header>
   <env:Body>
    <p:itinerary
      xmlns:p="http://travelcompany.example.org/
  reservation/travel">
     <p:departure>
       <p:departing>New York</p:departing>
       <p:arriving>Los Angeles</p:arriving>
       <p:departureDate>2001-12-14</p:departureDate>
       <p:departureTime>late afternoon</p:departureTime>
       <p:seatPreference>aisle</p:seatPreference>
     </p:departure>
     <p:return>
       <p:departing>Los Angeles</p:departing>
       <p:arriving>New York</p:arriving>
       <p:departureDate>2001-12-20</p:departureDate>
       <p:departureTime>mid-morning</p:departureTime>
       <p:seatPreference/>
     </p:return>
    </p:itinerary>
    <q:lodging
     xmlns:q="http://travelcompany.example.org/
  reservation/hotels">
     <q:preference>none</q:preference>
    </q:lodging>
   </env:Body>
  </env:Envelope>
```

Sample SOAP message for a travel reservation containing header blocks and a body

The SOAP message in Example 1 contains two SOAP-specific sub-elements within the overall `env:Envelope`, namely an `env:Header` and an `env:Body`. The contents of these elements are application defined and not a part of the SOAP specifications, although the latter do have something to say about how such elements must be handled.

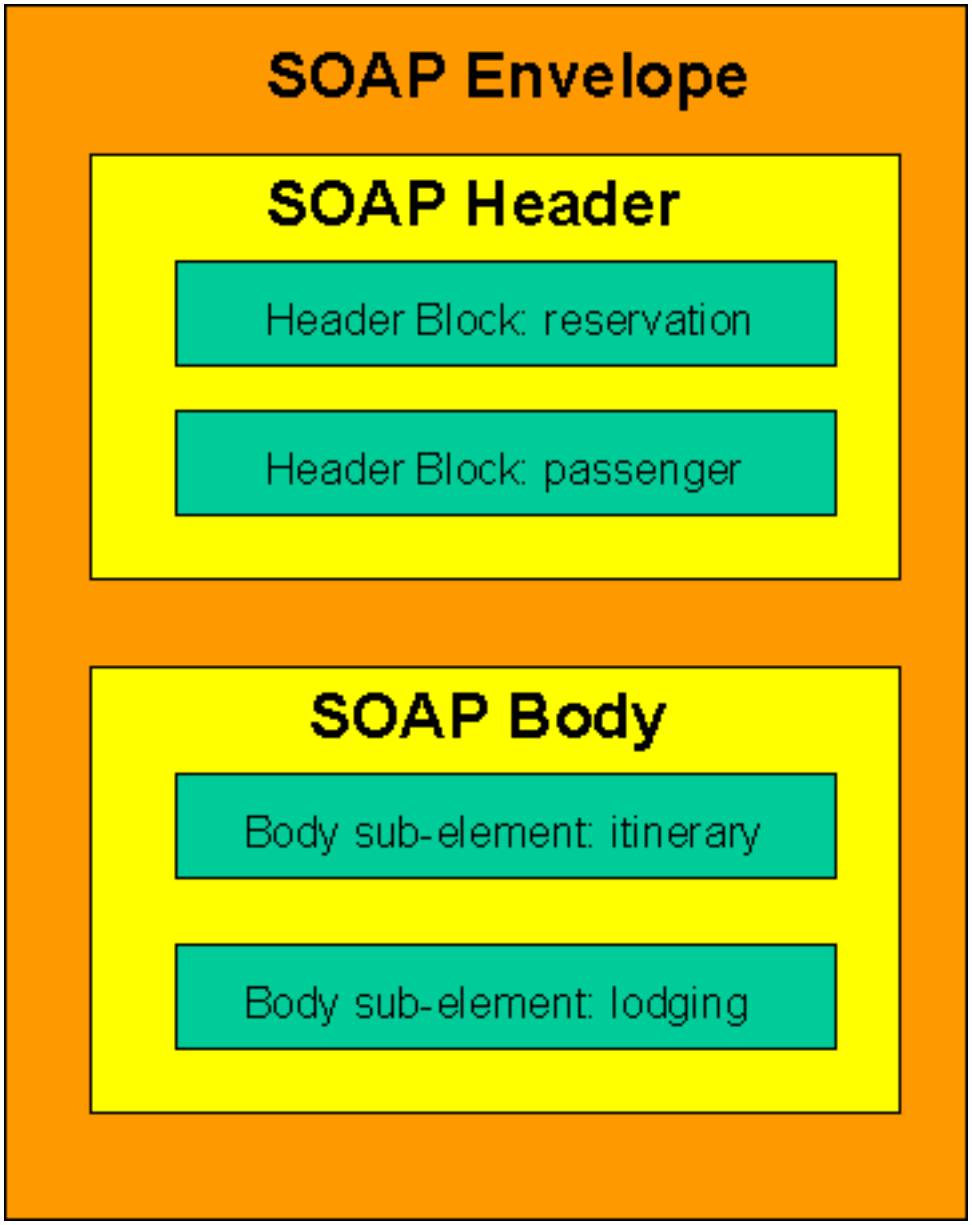
A SOAP header element is optional, but it has been included in the example to explain certain features of SOAP. A SOAP header is an extension mechanism that provides a way to pass information in SOAP messages that is not application payload. Such "control" information includes, for example, passing directives or contextual information related to the processing of the message. This allows a SOAP message to be extended in an application-specific manner. The immediate child elements of the `env:Header` element

are called header blocks, and represent a logical grouping of data which, as shown later, can individually be targeted at SOAP nodes that might be encountered in the path of a message from a sender to an ultimate receiver.

SOAP headers have been designed in anticipation of various uses for SOAP, many of which will involve the participation of other SOAP processing nodes - called SOAP intermediaries - along a message's path from an initial SOAP sender to an ultimate SOAP receiver. This allows SOAP intermediaries to provide value-added services. Headers, as shown later, may be inspected, inserted, deleted or forwarded by SOAP nodes encountered along a SOAP message path. (It should be kept in mind, though, that the SOAP specifications do not deal with what the contents of header elements are, or how SOAP messages are routed between nodes, or the manner by which the route is determined and so forth. These are a part of the overall application, and could be the subject of other specifications.)

The SOAP body is the mandatory element within the SOAP `env:Envelope`, which implies that this is where the main end-to-end information conveyed in a SOAP message must be carried.

A pictorial representation of the SOAP message in Example 1 is as follows.



In Example 1, the header contains two header blocks, each of which is defined in its own XML namespace and which represent some aspect pertaining to the overall processing of the body of the SOAP message. For this travel reservation application, such "meta" information pertaining to the overall request is a `reservation` header block which provides a reference and time stamp for this instance of a reservation, and the traveller's identity in the `passenger` block.

The header blocks `reservation` and `passenger` must be processed by the next SOAP intermediary encountered in the message path or, if there is no intermediary, by the ultimate recipient of the message. The fact that it is targeted at the next SOAP node encountered *en route* is indicated by the presence of the attribute `env:role` with the value "http://www.w3.org/2003/05/soap-envelope/role/next" (hereafter simply "next"), which is a role that all SOAP nodes must be willing to play. The presence of an `env:mustUnderstand` attribute with value "true" indicates that the node(s) processing the header must absolutely process these header blocks in a

manner consistent with their specifications, or else not process the message at all and throw a fault. Note that whenever a header block is processed, either because it is marked `env:mustUnderstand="true"` or for another reason, the block must be processed in accordance with the specifications for that block. Such header block specifications are application defined and not a part of SOAP. Section 3 will elaborate further on SOAP message processing based on the values of these attributes.

The choice of what data is placed in a header block and what goes in the SOAP body are decisions taken at the time of application design. The main point to keep in mind is that header blocks may be targeted at various nodes that might be encountered along a message's path from a sender to the ultimate recipient. Such intermediate SOAP nodes may provide value-added services based on data in such headers. In Example 1, the passenger data is placed in a header block to illustrate the use of this data at a SOAP intermediary to do some additional processing. For example, as shown later in section 5.1, the outbound message is altered by the SOAP intermediary by having the travel policies pertaining to this passenger appended to the message as another header block.

The `env:Body` element and its associated child elements, `itinerary` and `lodging`, are intended for exchange of information between the initial SOAP sender and the SOAP node which assumes the role of the ultimate SOAP receiver in the message path, which is the travel service application. Therefore, the `env:Body` and its contents are implicitly targeted and are expected to be understood by the ultimate receiver. The means by which a SOAP node assumes such a role is not defined by the SOAP specification, and is determined as a part of the overall application semantics and associated message flow.

Note that a SOAP intermediary may decide to play the role of the ultimate SOAP receiver for a given message transfer, and thus process the `env:Body`. However, even though this sort of a behavior cannot be prevented, it is not something that should be done lightly as it may pervert the intentions of the message's sender, and have undesirable side effects (such as not processing header blocks that might be targeted at intermediaries further along the message path).

A SOAP message such as that in Example 1 may be transferred by different underlying protocols and used in a variety of message exchange patterns. For example, for a Web-based access to a travel service application, it could be placed in the body of a HTTP POST request. In another protocol binding, it might be sent in an email message (see section 4.2). Section 4 will describe how SOAP messages may be conveyed by a variety of underlying protocols.

For the time being, it is assumed that a mechanism exists for message transfer and the remainder of this section concentrates on the details of the SOAP messages and their processing.

2.2 SOAP Message Exchange

SOAP Version 1.2 is a simple messaging framework for transferring information specified in the form of an XML infoset between an initial SOAP sender and an ultimate SOAP receiver. The more interesting scenarios typically involve multiple message exchanges between these two nodes. The simplest such exchange is a request-response pattern. Some early uses of [SOAP 1.1] emphasized the use of this pattern as means for conveying remote procedure calls (RPC), but it is important to note that not all SOAP request-response exchanges can or need to be modelled as RPCs. The latter is used when there is a need to model a certain programmatic behavior, with the exchanged messages conforming to a pre-defined description of the remote call and its return.

A much larger set of usage scenarios than that covered by the request-response pattern can be modeled simply as XML-based content exchanged in SOAP messages to form a back-and-forth "conversation", where the semantics are at the level of the sending and receiving applications. Section 2.2.1 covers the case of XML-based content exchanged in SOAP messages between the travel reservation application and the travel service application in a conversational pattern, while section 2.2.2 provides an example of an exchange modeled as an RPC.

2.2.1 Conversational Message Exchanges

Continuing with the travel request scenario, Example 2 shows a SOAP message returned from the travel service in response to the reservation request message in Example 1. This response seeks to refine some information in the request, namely the choice of airports in the departing city.

Example 2

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.
example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
```

```
            env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-
pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:
dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/
employees"
      env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
            env:mustUnderstand="true">
   <n:name>Åke Jógvan Øyvind</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itineraryClarification
    xmlns:p="http://travelcompany.example.org/
reservation/travel">
   <p:departure>
     <p:departing>
       <p:airportChoices>
          JFK LGA EWR
       </p:airportChoices>
     </p:departing>
   </p:departure>
   <p:return>
     <p:arriving>
       <p:airportChoices>
         JFK LGA EWR
       </p:airportChoices>
     </p:arriving>
   </p:return>
  </p:itineraryClarification>
 </env:Body>
</env:Envelope>
```

SOAP message sent in response to the message in Example 1

As described earlier, the `env:Body` contains the primary content of the message, which in this example includes a list of the various alternatives for the airport, conforming to a schema definition in the XML namespace http://travelcompany.example.org/reservation/travel. In this example, the header blocks from Example 1 are returned (with some sub-element values altered) in the response. This could allow message correlation at the SOAP level, but such headers are very likely to also have other application-specific uses.

The message exchange in Examples 1 and 2 are cases where XML-based content conforming to some application-defined schema are exchanged via SOAP messages. Once again, a discussion of the means by which such messages are transferred is deferred to section 4.

It is easy enough to see how such exchanges can build up to a multiple back-and-forth "conversational" message exchange pattern. Example 3 shows a SOAP message sent by the travel reservation application in response to that in Example 2 choosing one from the list of available airports. The header block `reservation` with the same value of the `reference` sub-element accompanies each message in this conversation, thereby offering a way, should it be needed, to correlate the messages exchanged between them at the application level.

Example 3

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
 <env:Header>
  <m:reservation
     xmlns:m="http://travelcompany.example.org/
reservation"
      env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
           env:mustUnderstand="true">
    <m:reference>uuid:093a2da1-q345-739r-ba5d-
pqff98fe8j7d</m:reference>
    <m:dateAndTime>2001-11-29T13:36:50.000-05:00</m:
dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/
employees"
      env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Åke Jógvan Øyvind</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary
   xmlns:p="http://travelcompany.example.org/
reservation/travel">
   <p:departure>
     <p:departing>LGA</p:departing>
```

```
    </p:departure>
    <p:return>
      <p:arriving>EWR</p:arriving>
    </p:return>
   </p:itinerary>
  </env:Body>
 </env:Envelope>
```

Response to the message in Example 2 continuing a conversational message exchange

2.2.2 Remote Procedure Calls

One of the design goals of SOAP Version 1.2 is to encapsulate remote procedure call functionality using the extensibility and flexibility of XML. SOAP Part 2 section 4 has defined a uniform representation for RPC invocations and responses carried in SOAP messages. This section continues with the travel reservation scenario to illustrate the use of SOAP messages to convey remote procedure calls and their return.

To that end, the next example shows the payment for the trip using a credit card. (It is assumed that the conversational exchanges described in section 2.2.1 have resulted in a confirmed itinerary.) Here, it is further assumed that the payment happens in the context of an overall transaction where the credit card is charged only when the travel and the lodging (not shown in any example, but presumably reserved in a similar manner) are both confirmed. The travel reservation application provides credit card information and the successful completion of the different activities results in the card being charged and a reservation code returned. This reserve-and-charge interaction between the travel reservation application and the travel service application is modeled as a SOAP RPC.

To invoke a SOAP RPC, the following information is needed:

1. The address of the target SOAP node.
2. The procedure or method name.
3. The identities and values of any arguments to be passed to the procedure or method together with any output parameters and return value.
4. A clear separation of the arguments used to identify the Web resource which is the actual target for the RPC, as contrasted with those that convey data or control information used for processing the call by the target resource.
5. The message exchange pattern which will be employed to convey the RPC, together with an identification of the so-called "Web Method" (on

which more later) to be used.
6. Optionally, data which may be carried as a part of SOAP header blocks.

Such information may be expressed by a variety of means, including formal Interface Definition Languages (IDL). Note that SOAP does not provide any IDL, formal or informal. Note also that the above information differs in subtle ways from information generally needed to invoke other, non-SOAP RPCs.

Regarding Item 1 above, there is, from a SOAP perspective, a SOAP node which "contains" or "supports" the target of the RPC. It is the SOAP node which (appropriately) adopts the role of the ultimate SOAP receiver. As required by Item 1, the ultimate recipient can identify the target of the named procedure or method by looking for its URI. The manner in which the target URI is made available depends on the underlying protocol binding. One possibility is that the URI identifying the target is carried in a SOAP header block. Some protocol bindings, such as the SOAP HTTP binding defined in [SOAP Part2], offer a mechanism for carrying the URI outside the SOAP message. In general, one of the properties of a protocol binding specification must be a description of how the target URI is carried as a part of the binding. Section 4.1 provides some concrete examples of how the URI is carried in the case of the standardized SOAP protocol binding to HTTP.

Item 4 and Item 5 above are required to ensure that RPC applications that employ SOAP can do so in a manner which is compatible with the architectural principles of the World Wide Web. Section 4.1.3 discusses how the information provided by items 4 and 5 are utilized.

For the remainder of this section, it is assumed that the RPC conveyed in a SOAP message as shown in Example 4 is appropriately targeted and dispatched. The purpose of this section is to highlight the syntactical aspects of RPC requests and returns carried within a SOAP message.

Example 4

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope" >
 <env:Header>
   <t:transaction
           xmlns:t="http://thirdparty.example.org/
transaction"
           env:encodingStyle="http://example.com/
encoding"
           env:mustUnderstand="true" >5</t:
transaction>
```

```
 </env:Header>
 <env:Body>
  <m:chargeReservation
      env:encodingStyle="http://www.w3.org/2003/05/
soap-encoding"
         xmlns:m="http://travelcompany.example.org/">
   <m:reservation xmlns:m="http://travelcompany.
example.org/reservation">
    <m:code>FT35ZBQ</m:code>
   </m:reservation>
   <o:creditCard xmlns:o="http://mycompany.example.
com/financial">
    <n:name xmlns:n="http://mycompany.example.com/
employees">
           Åke Jógvan Øyvind
    </n:name>
    <o:number>123456789099999</o:number>
    <o:expiration>2005-02</o:expiration>
   </o:creditCard>
  </m:chargeReservation>
 </env:Body>
</env:Envelope>
```

SOAP RPC request with a mandatory header and two input (or "in") parameters

The RPC itself is carried as a child of the `env:Body` element, and is modelled as a `struct` which takes the name of the procedure or method, in this case `chargeReservation`. (A `struct` is a concept from the SOAP Data Model defined in [SOAP Part2] that models a structure or record type that occurs in some common programming languages.) The design of the RPC in the example (whose formal description has not been explicitly provided) takes two input (or "in") parameters, the `reservation` corresponding to the planned trip identified by the reservation `code`, and the `creditCard` information. The latter is also a `struct`, which takes three elements, the card holder's `name`, the card `number` and an `expiration` date.

In this example, the `env:encodingStyle` attribute with the value http://www.w3.org/2003/05/soap-encoding shows that the contents of the `chargeReservation` structure have been serialized according to the SOAP encoding rules, i.e., the particular rules defined in SOAP Part 2 section 3. Even though SOAP specifies this particular encoding scheme, its use is optional and the specification makes clear that other encoding schemes may be used for application-specific data within a SOAP message. It is for this purpose that it provides the `env:encodingStyle` attribute to qualify header

blocks and body sub-elements. The choice of the value for this attribute is an application-specific decision and the ability of a caller and callee to interoperate is assumed to have been settled "out-of-band". Section 5.2 shows an example of using another encoding scheme.

As noted in Item 6 above, RPCs may also require additional information to be carried, which can be important for the processing of the call in a distributed environment, but which are not a part of the formal procedure or method description. (Note, however, that providing such additional contextual information is not specific to RPCs, but may be required in general for the processing of any distributed application.) In the example, the RPC is carried out in the context of an overall transaction which involves several activities which must all complete successfully before the RPC returns successfully. Example 4 shows how a header block `transaction` directed at the ultimate recipient (implied by the absence of the `env:role` attribute) is used to carry such information. (The value "5" is some transaction identifier set by and meaningful to the application. No further elaboration of the application-specific semantics of this header are provided here, as it is not germane to the discussion of the syntactical aspects of SOAP RPC messages.)

Let us assume that the RPC in the charging example has been designed to have the procedure description which indicates that there are two output (or "out") parameters, one providing the reference code for the reservation and the other a URL where the details of the reservation may be viewed. The RPC response is returned in the `env:Body` element of a SOAP message, which is modeled as a `struct` taking the procedure name `chargeReservation` and, as a convention, the word "Response" appended. The two output (or "out") parameters accompanying the response are the alphanumeric `code` identifying the reservation in question, and a URI for the location, `viewAt`, from where the reservation may be retrieved.

This is shown in Example 5a, where the header again identifies the transaction within which this RPC is performed.

Example 5a

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope" >
 <env:Header>
     <t:transaction
        xmlns:t="http://thirdparty.example.org/
transaction"
          env:encodingStyle="http://example.com/
encoding"
```

```
            env:mustUnderstand="true">5</t:transaction>
 </env:Header>
 <env:Body>
     <m:chargeReservationResponse
         env:encodingStyle="http://www.w3.org/2003/05/
soap-encoding"
             xmlns:m="http://travelcompany.example.
org/">
       <m:code>FT35ZBQ</m:code>
       <m:viewAt>
         http://travelcompany.example.org/
reservations?code=FT35ZBQ
       </m:viewAt>
     </m:chargeReservationResponse>
 </env:Body>
</env:Envelope>
```

RPC response with two output (or "out") parameters for the call shown in Example 4

RPCs often have descriptions where a particular output parameter is distinguished, the so-called "return" value. The SOAP RPC convention offers a way to distinguish this "return" value from the other output parameters in the procedure description. To show this, the charging example is modified to have an RPC description that is almost the same as that for Example 5a, i.e, with the same two "out" parameters, but in addition it also has a "return" value, which is an enumeration with potential values of "confirmed" and "pending". The RPC response conforming to this description is shown in Example 5b, where the SOAP header, as before, identifies the transaction within which this RPC is performed.

Example 5b

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope" >
 <env:Header>
    <t:transaction
       xmlns:t="http://thirdparty.example.org/
transaction"
         env:encodingStyle="http://example.com/
encoding"
          env:mustUnderstand="true">5</t:transaction>
 </env:Header>
 <env:Body>
```

```
    <m:chargeReservationResponse
       env:encodingStyle="http://www.w3.org/2003/05/
soap-encoding"
           xmlns:rpc="http://www.w3.org/2003/05/soap-
rpc"
             xmlns:m="http://travelcompany.example.
org/">
       <rpc:result>m:status</rpc:result>
       <m:status>confirmed</m:status>
       <m:code>FT35ZBQ</m:code>
       <m:viewAt>
        http://travelcompany.example.org/reservations?
code=FT35ZBQ
       </m:viewAt>
    </m:chargeReservationResponse>
 </env:Body>
</env:Envelope>
```

RPC response with a "return" value and two "out" parameters for the call shown in Example 4

In Example 5b, the return value is identified by the element `rpc:result`, and contains the XML Qualified Name (of type `xs:QName`) of another element within the `struct` which is `m:status`. This, in turn, contains the actual return value, "confirmed". This technique allows the actual return value to be strongly typed according to some schema. If the `rpc:result` element is absent, as is the case in Example 5a, the return value is not present or is of the type `void`.

While, in principle, using SOAP for RPC is independent of the decision to use a particular means for transferring the RPC call and its return, certain protocol bindings that support the SOAP Request-Response message exchange pattern may be more naturally suited for such purposes. A protocol binding supporting this message exchange pattern can provide the correlation between a request and a response. Of course, the designer of an RPC-based application could choose to put a correlation ID relating a call and its return in a SOAP header, thereby making the RPC independent of any underlying transfer mechanism. In any case, application designers have to be aware of all the characteristics of the particular protocols chosen for transferring SOAP RPCs, such as latency, synchrony, etc.

In the commonly used case, standardized in SOAP Part 2 section 7, of using HTTP as the underlying transfer protocol, an RPC invocation maps naturally to the HTTP request and an RPC response maps to the HTTP response. Section 4.1 provides examples of carrying RPCs using the HTTP binding.

However, it is worth keeping in mind that even though most examples of SOAP for RPC use the HTTP protocol binding, it is not limited to that means alone.

2.3 Fault Scenarios

SOAP provides a model for handling situations when faults arise in the processing of a message. SOAP distinguishes between the conditions that result in a fault, and the ability to signal that fault to the originator of the faulty message or another node. The ability to signal the fault depends on the message transfer mechanism used, and one aspect of the binding specification of SOAP onto an underlying protocol is to specify how faults are signalled, if at all. The remainder of this section assumes that a transfer mechanism is available for signalling faults generated while processing received messages, and concentrates on the structure of the SOAP fault message.

The SOAP `env:Body` element has another distinguished role in that it is the place where such fault information is placed. The SOAP fault model (see SOAP Part 1, section 2.6) requires that all SOAP-specific and application-specific faults be reported using a *single* distinguished element, `env:Fault`, carried within the `env:Body` element. The `env:Fault` element contains two mandatory sub-elements, `env:Code` and `env:Reason`, and (optionally) application-specific information in the `env:Detail` sub-element. Another optional sub-element, `env:Node`, identifies via a URI the SOAP node which generated the fault, its absence implying that it was the ultimate recipient of the message which did so. There is yet another optional sub-element, `env:Role`, which identifies the role being played by the node which generated the fault.

The `env:Code` sub-element of `env:Fault` is itself made up of a mandatory `env:Value` sub-element, whose content is specified in the SOAP specification (see SOAP Part 1 section 5.4.6) as well as an optional `env:Subcode` sub-element.

Example 6a shows a SOAP message returned in response to the RPC request in Example 4, and indicating a failure to process the RPC.

Example 6a

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
```

```
soap-envelope"
            xmlns:rpc='http://www.w3.org/2003/05/soap-
rpc'>
  <env:Body>
   <env:Fault>
     <env:Code>
       <env:Value>env:Sender</env:Value>
       <env:Subcode>
        <env:Value>rpc:BadArguments</env:Value>
       </env:Subcode>
     </env:Code>
     <env:Reason>
      <env:Text xml:lang="en-US">Processing error</
env:Text>
       <env:Text xml:lang="cs">Chyba zpracování</env:
Text>
     </env:Reason>
     <env:Detail>
      <e:myFaultDetails
        xmlns:e="http://travelcompany.example.org/
faults">
        <e:message>Name does not match card number</e:
message>
        <e:errorcode>999</e:errorcode>
      </e:myFaultDetails>
     </env:Detail>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

Sample SOAP message indicating failure to process the RPC in Example 4

In Example 6a, the top-level `env:Value` uses a standardized XML Qualified Name (of type `xs:QName`) to identify that it is an `env:Sender` fault, which indicates that it is related to some syntactical error or inappropriate information in the message. (When a `env:Sender` fault is received by the sender, it is expected that some corrective action is taken before a similar message is sent again.) The `env:Subcode` element is optional, and, if present, as it is in this example, qualifies the parent value further. In Example 6a, the `env:Subcode` denotes that an RPC specific fault, `rpc:BadArguments`, defined in SOAP Part 2 section 4.4, is the cause of the failure to process the request.

The structure of the `env:Subcode` element has been chosen to be hierarchical - each child `env:Subcode` element has a mandatory `env:`

Value and an optional env:Subcode sub-element - to allow application-specific codes to be carried. This hierarchical structure of the env:Code element allows for an uniform mechanism for conveying multiple level of fault codes. The top-level env:Value is a base fault that is specified in the SOAP Version 1.2 specifications (see SOAP Part 1 section 5.4.6) and must be understood by all SOAP nodes. Nested env:Values are application-specific, and represent further elaboration or refinement of the base fault from an application perspective. Some of these values may well be standardized, such as the RPC codes standardized in SOAP 1.2 (see SOAP Part 2 section 4.4), or in some other standards that use SOAP as an encapsulation protocol. The only requirement for defining such application-specific subcode values is that they be namespace qualified using any namespace other than the SOAP env namespace which defines the main classifications for SOAP faults. There is no requirement from a SOAP perspective that applications need to understand, or even look at all levels of the subcode values.

The env:Reason sub-element is not meant for algorithmic processing, but rather for human understanding; so, even though this is a mandatory item, the chosen value need not be standardized. Therefore all that is required is that it reasonably accurately describe the fault situation. It must have one or more env:Text sub-elements, each with a unique xml:lang attribute, which allows applications to make the fault reason available in multiple languages. (Applications could negotiate the language of the fault text using a mechanism built using SOAP headers; however this is outside the scope of the SOAP specifications.)

The absence of a env:Node sub-element within env:Fault in Example 6a implies that it is generated by the ultimate receiver of the call. The contents of env:Detail, as shown in the example, are application-specific.

During the processing of a SOAP message, a fault may also be generated if a mandatory header element is not understood or the information contained in it cannot be processed. Errors in processing a header block are also signalled using a env:Fault element within the env:Body, but with a particular distinguished header block, env:NotUnderstood, that identifies the offending header block.

Example 6b shows an example of a response to the RPC in Example 4 indicating a failure to process the t:transaction header block. Note the presence of the env:MustUnderstand fault code in the env:Body, and the identification of the header not understood using an (unqualified) attribute, qname, in the special (empty) header block env:NotUnderstood.

Example 6b

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope>
 <env:Header>
    <env:NotUnderstood qname="t:transaction"
                xmlns:t="http://thirdparty.example.org/
transaction"/>
 </env:Header>
 <env:Body>
  <env:Fault>
   <env:Code>
    <env:Value>env:MustUnderstand</env:Value>
   </env:Code>
   <env:Reason>
      <env:Text xml:lang="en-US">Header not
understood</env:Text>
      <env:Text xml:lang="fr">En-tête non compris</
env:Text>
   </env:Reason>
  </env:Fault>
 </env:Body>
</env:Envelope>
```

Sample SOAP message indicating failure to process the header block in Example 4

If there were several mandatory header blocks that were not understood, then each could be identified by its `qname` attribute in a series of such `env:NotUnderstood` header blocks.

3. SOAP Processing Model

Having established the various syntactical aspects of a SOAP message as well as some basic message exchange patterns, this section provides a general overview of the SOAP processing model (specified in SOAP Part 1, section 2). The SOAP processing model describes the actions taken by a SOAP node on receiving a SOAP message.

Example 7a shows a SOAP message with several header blocks (with their contents omitted for brevity). Variations of this will be used in the remainder of this section to illustrate various aspects of the processing model.

Example 7a

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
   <env:Header>
     <p:oneBlock xmlns:p="http://example.com"
            env:role="http://example.com/Log">
     ...
     ...
     </p:oneBlock>
     <q:anotherBlock xmlns:q="http://example.com"
       env:role="http://www.w3.org/2003/05/soap-
envelope/role/next">
     ...
     ...
     </q:anotherBlock>
     <r:aThirdBlock xmlns:r="http://example.com">
     ...
     ...
     </r:aThirdBlock>
   </env:Header>
   <env:Body >
     ...
     ...
   </env:Body>
 </env:Envelope>
```

SOAP message showing a variety of header blocks

The SOAP processing model describes the (logical) actions taken by a SOAP node on receiving a SOAP message. There is a requirement for the node to analyze those parts of a message that are SOAP-specific, namely those elements in the SOAP "env" namespace. Such elements are the envelope itself, the header element and the body element. A first step is, of course, the overall check that the SOAP message is syntactically correct. That is, it conforms to the SOAP XML infoset subject to the restrictions on the use of certain XML constructs - Processing Instructions and Document Type Definitions - as defined in SOAP Part 1, section 5.

3.1 The "role" Attribute

Further processing of header blocks and the body depend on the role(s) assumed by the SOAP node for the processing of a given message. SOAP defines the (optional) `env:role` attribute - syntactically, `xs:anyURI` - that may be present in a header block, which identifies the role played by the intended target of that header block. A SOAP node is required to process a

header block if it assumes the role identified by the value of the URI. How a SOAP node assumes a particular role is not a part of the SOAP specifications.

Three standardized roles have been defined (see SOAP Part 1, section 2.2), which are

- "http://www.w3.org/2003/05/soap-envelope/role/none" (hereafter simply "none")
- "http://www.w3.org/2003/05/soap-envelope/role/next" (hereafter simply "next"), and
- "http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver" (hereafter simply "ultimateReceiver").

In Example 7a, the header block `oneBlock` is targeted at any SOAP node that plays the application-defined role defined by the URI http://example.com/Log. For purposes of illustration, it is assumed that the specification for such a header block requires that any SOAP node adopting this role log the entire message.

Every SOAP node receiving a message with a header block that has a `env:role` attribute of "next" must be capable of processing the contents of the element, as this is a standardized role that every SOAP node must be willing to assume. A header block thus attributed is one which is expected to be examined and (possibly) processed by the next SOAP node along the path of a message, assuming that such a header has not been removed as a result of processing at some node earlier in the message path.

In Example 7a, the header block `anotherBlock` is targeted at the next node in the message path. In this case, the SOAP message received by the node playing the application-defined role of "http://example.com/Log", must also be willing to play the SOAP-defined role of "next". This is also true for the node which is the ultimate recipient of the message, as it obviously (and implicitly) also plays the "next" role by virtue of being next in the message path.

The third header block, `aThirdBlock`, in Example 7a does not have the `env:role` attribute. It is targeted at a SOAP node which assumes the "ultimateReceiver" role. The "ultimateReceiver" role (which can be explicitly declared or is implicit if the `env:role` attribute is absent in a header block) is played by a SOAP node that assumes the role of the ultimate recipient of a particular SOAP message. The absence of a `env:role` attribute in the `aThirdBlock` header block means that this header element is targeted at the SOAP node that assumes the "ultimateReceiver" role.

Note that the `env:Body` element does not have a `env:role` attribute. The body element is *always* targeted at the SOAP node that assumes the

"ultimateReceiver" role. In that sense, the body element is just like a header block targeted at the ultimate receiver, but it has been distinguished to allow for SOAP nodes (typically SOAP intermediaries) to skip over it if they assume roles other than that of the ultimate receiver. SOAP does not prescribe any structure for the `env:Body` element, except that it recommends that any sub-elements be XML namespace qualified. Some applications, such as that in Example 1, may choose to organize the sub-elements of `env:Body` in blocks, but this is not of concern to the SOAP processing model.

The other distinguished role for the `env:Body` element, as the container where information on SOAP-specific faults, i.e., failure to process elements of a SOAP message, is placed has been described previously in section 2.3.

If a header element has the standardized `env:role` attribute with value "none", it means that no SOAP node should process the contents, although a node may need to examine it if the content are data referenced by another header element that is targeted at the particular SOAP node.

If the `env:role` attribute has an empty value, i.e., `env:role=""`, it means that the relative URI identifying the role is resolved to the base URI for the SOAP message in question. SOAP Version 1.2 does not define a base URI for a SOAP message, but defers to the mechanisms defined in [XMLBase] for deriving the base URI, which can be used to make any relative URIs absolute. One such mechanism is for the protocol binding to establish a base URI, possibly by reference to the encapsulating protocol in which the SOAP message is embedded for transport. (In fact, when SOAP messages are transported using HTTP, SOAP Part 2 section 7.1.2 defines the base URI as the Request-URI of the HTTP request, or the value of the HTTP Content-Location header.)

The following table summarizes the applicable standardized roles that may be assumed at various SOAP nodes. ("Yes" and "No" means that the corresponding node does or does not, respectively, play the named role.)

Role	absent	"none"	"next"	"ultimateReceiver"
Node				
initial sender	not applicable	not applicable	not applicable	not applicable
intermediary	no	no	yes	no
ultimate receiver	yes	no	yes	yes

3.2 The "mustUnderstand" Attribute

Example 7b augments the previous example by introducing another (optional) attribute for header blocks, the `env:mustUnderstand` attribute.

Example 7b

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
   <env:Header>
     <p:oneBlock xmlns:p="http://example.com"
           env:role="http://example.com/Log"
               env:mustUnderstand="true">
     ...
     ...
     </p:oneBlock>
     <q:anotherBlock xmlns:q="http://example.com"
       env:role="http://www.w3.org/2003/05/soap-
envelope/role/next">
     ...
     ...
     </q:anotherBlock>
     <r:aThirdBlock xmlns:r="http://example.com">
     ...
     ...
     </r:aThirdBlock>
   </env:Header>
   <env:Body >
     ...
     ...
   </env:Body>
 </env:Envelope>
```

SOAP message showing a variety of header blocks, one of which is mandatory for processing

After a SOAP node has correctly identified the header blocks (and possibly the body) targeted at itself using the `env:role` attribute, the additional attribute, `env:mustUnderstand`, in the header elements determines further processing actions that have to be taken. In order to ensure that SOAP nodes do not ignore header blocks which are important to the overall purpose of the application, SOAP header blocks also provide for the additional optional attribute, `env:mustUnderstand`, which, if "true", means that the targeted SOAP node **must** process the block according to the specification of that

block. Such a block is colloquially referred to as a mandatory header block. In fact, processing of the SOAP message must not even start until the node has identified all the mandatory header blocks targeted at itself, and "understood" them. Understanding a header means that the node must be prepared to do whatever is described in the specification of that block. (Keep in mind that the specifications of header blocks are not a part of the SOAP specifications.)

In Example 7b, the header block `oneBlock` is marked with a `env:mustUnderstand` value set to "true", which means that it is mandatory to process this block if the SOAP node plays the role identified by "http://example.com/Log". The other two header blocks are not so marked, which means that SOAP node at which these blocks are targeted need not process them. (Presumably the specifications for these blocks allow for this.)

A `env:mustUnderstand` value of "true" means that the SOAP node must process the header with the semantics described in that header's specification, or else generate a SOAP fault. Processing the header appropriately may include removing the header from any generated SOAP message, reinserting the header with the same or altered value, or inserting a new header. The inability to process a mandatory header requires that all further processing of the SOAP message cease, and a SOAP fault be generated. The message is not forwarded any further.

The `env:Body` element has no `env:mustUnderstand` attribute but it *must* be processed by the ultimate recipient. In Example 7b, the ultimate recipient of the message - the SOAP node which plays the "ultimateReceiver" role - must process the `env:Body` and may process the header block `aThirdBlock`. It may also process the header block `anotherBlock`, as it is targeted at it (in the role of "next") but it is not mandatory to do so if the specifications for processing the blocks do not demand it. (If the specification for `anotherBlock` demanded that it must be processed at the next recipient, it would have required that it be marked with a `env:mustUnderstand="true"`.)

The role(s) a SOAP node plays when processing a SOAP message can be determined by many factors. The role could be known *a priori*, or set by some out-of-band means, or a node can inspect all parts of a received message to determine which roles it will assume before processing the message. An interesting case arises when a SOAP node, during the course of processing a message, decides that there are additional roles that it needs to adopt. No matter when this determination is made, externally it must appear as though the processing model has been adhered to. That is, it must appear as though the role had been known from the start of the processing of the message. In particular, the external appearance must be that the `env:mustUnderstand` checking of any headers with those additional roles assumed was performed

before any processing began. Also, if a SOAP node assumes such additional roles, it must ensure that it is prepared to do everything that the specifications for those roles require.

The following table summarizes how the processing actions for a header block are qualified by the `env:mustUnderstand` attribute with respect to a node that has been appropriately targeted (via the `env:role` attribute).

Node	intermediary	ultimate receiver
mustUnderstand		
"true"	must process	must process
"false"	may process	may process
absent	may process	may process

As a result of processing a SOAP message, a SOAP node may generate a single SOAP fault if it fails to process a message, or, depending on the application, generate additional SOAP messages for consumption at other SOAP nodes. SOAP Part 1 section 5.4 describes the structure of the fault message while the SOAP processing model defines the conditions under which it is generated. As illustrated previously in section 2.3, a SOAP fault is a SOAP message with a standardized `env:Body` sub-element named `env:Fault`.

SOAP makes a distinction between generating a fault and ensuring that the fault is returned to the originator of the message or to another appropriate node which can benefit from this information. However, whether a generated fault can be propagated appropriately depends on the underlying protocol binding chosen for the SOAP message message exchange. The specification does not define what happens if faults are generated during the propagation of one-way messages. The only normative underlying protocol binding, which is the SOAP HTTP binding, offers the HTTP response as a means for reporting a fault in the incoming SOAP message. (See Section 4 for more details on SOAP protocol bindings.)

3.3 The "relay" Attribute

SOAP Version 1.2 defines another optional attribute for header blocks, `env:relay` of type `xs:boolean`, which indicates if a header block targeted at a SOAP intermediary must be relayed if it is *not* processed.

Note that if a header block *is* processed, the SOAP processing rules (see SOAP Part 1 section 2.7.2) requires that it be removed from the outbound

message. (It may, however, be reinserted, either unchanged or with its contents altered, if the processing of other header blocks determines that the header block be retained in the forwarded message.) The default behavior for *an un*processed header block targeted at a role played by a SOAP intermediary is that it must be removed before the message is relayed.

The reason for this choice of default is to lean on the side of safety by ensuring that a SOAP intermediary make no assumptions about the survivability past itself of a header block targeted at a role it assumes, and representing some value-added feature, particularly if it chooses not to process the header block, very likely because it does not "understand" it. That is because certain header blocks represent hop-by-hop features, and it may not make sense to unknowingly propagate it end-to-end. As an intermediary may not be in a position to make this determination, it was thought that it would be safer if unprocessed header blocks were removed before the message was relayed.

However, there are instances when an application designer would like to introduce a new feature, manifested through a SOAP header block, targeted at *any* capable intermediary which might be encountered in the SOAP message path. Such a header block would be available to those intermediaries that "understood" it, but ignored and relayed onwards by those that did not. Being a new feature, the processing software for this header block may be implemented, at least initially, in some but not all SOAP nodes. Marking such a header block with `env:mustUnderstand` = "false" is obviously needed, so that intermediaries that have not implemented the feature do not generate a fault. To circumvent the default rule of the processing model, marking a header block with the additional attribute `env:relay` with the value "true" allows the intermediary to forward the header block targeted at itself in the event that it chooses not to process it.

Targeting the header block at the role "next" together with the `env:relay` attribute set to "true" can always serve to ensure that each intermediary has a chance to examine the header, because one of the anticipated uses of the "next" role is with header blocks that carry information that are expected to persist along a SOAP message path. Of course, the application designer can always define a custom role that allows targetting at specific intermediaries that assume this role. Therefore, there is no restriction on the use of the `env:relay` attribute with any role except of course the roles of "none" and "ultimateReceiver", for which it is meaningless.

Example 7c shows the use of the `env:relay` attribute.

Example 7c

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
   <env:Header>
     <p:oneBlock xmlns:p="http://example.com"
           env:role="http://example.com/Log"
               env:mustUnderstand="true">
     ...
     ...
     </p:oneBlock>
     <q:anotherBlock xmlns:q="http://example.com"
       env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
       env:relay="true">
     ...
     ...
     </q:anotherBlock>
     <r:aThirdBlock xmlns:r="http://example.com">
     ...
     ...
     </r:aThirdBlock>
   </env:Header>
   <env:Body >
     ...
     ...
   </env:Body>
 </env:Envelope>
```

SOAP message showing a variety of header blocks, one of which must be relayed if unprocessed.

The header block `q:anotherBlock`, targeted at the "next" node in the message path, has the additional attribute `env:relay`="true". A SOAP node receiving this message may process this header block if it "understands" it, but if it does so the processing rules require that this header block be removed before forwarding. However, if the SOAP node chooses to ignore this header block, which it can because it is not mandatory to process it, as indicated by the absence of the `env:mustUnderstand` attribute, then it must forward it.

Processing the header block `p:oneBlock` is mandatory and the SOAP processing rules require that it not be relayed, unless the processing of some other header block requires that it be present in the outbound message. The header block `r:aThirdBlock` does not have an `env:relay` attribute, which is equivalent to having it with the value `env:relay`="false". Hence, this header is not forwarded if it is not processed.

SOAP 1.2 Part 1 Table 3 summarizes the conditions which determine when a SOAP intermediary assuming a given role is allowed to forward unprocessed header blocks.

4. Using Various Protocol Bindings

SOAP messages may be exchanged using a variety of "underlying" protocols, including other application layer protocols. The specification of how SOAP messages may be passed from one SOAP node to another using an underlying protocol is called a SOAP binding. [SOAP Part1] defines a SOAP message in the form of an [XML Infoset], i.e., in terms of element and attribute information items of an abstract "document" called the `env:Envelope` (see SOAP Part 1, section 5). Any SOAP `env:Envelope` infoset representation will be made concrete through a protocol binding, whose task, among other things, it is to provide a serialized representation of the infoset that can be conveyed to the next SOAP node in the message path in a manner such that the infoset can be reconstructed without loss of information. In typical examples of SOAP messages, and certainly in all the examples in this primer, the serialization shown is that of a well-formed [XML 1.0] document. However, there may be other protocol bindings - for example a protocol binding between two SOAP nodes over a limited bandwidth interface - where an alternative, compressed serialization of the same infoset may be chosen. Another binding, chosen for a different purpose, may provide a serialization which is an encrypted structure representing the same infoset.

In addition to providing a concrete realization of a SOAP infoset between adjacent SOAP nodes along a SOAP message path, a protocol binding provides the mechanisms to support features that are needed by a SOAP application. A feature is a specification of a certain functionality provided by a binding. A feature description is identified by a URI, so that all applications referencing it are assured of the same semantics. For example, a typical usage scenario might require many concurrent request-response exchanges between adjacent SOAP nodes, in which case the feature that is required is the ability to correlate a request with a response. Other examples includes "an encrypted channel" feature, or a "reliable delivery channel" feature, or a particular SOAP message exchange pattern feature.

A SOAP binding specification (see SOAP Part 1 section 4) describes, among other things, which (if any) features it provides. Some features may be provided natively by the underlying protocol. If the feature is not available through the binding, it may be implemented within the SOAP envelope, using SOAP header blocks. The specification of a feature implemented using SOAP header blocks is called a SOAP module.

For example, if SOAP message exchanges were being transported directly over a datagram protocol like UDP, obviously the message correlation feature would have to be provided elsewhere, either directly by the application or more likely as a part of the SOAP infosets being exchanged. In the latter case, the message correlation feature has a binding-specific expression within the SOAP envelope, i.e., as a SOAP header block, defined in a "Request-Response Correlation" module identified by a URI. However, if the SOAP infosets were being exchanged using an underlying protocol that was itself request/response, the application could implicitly "inherit" this feature provided by the binding, and no further support need be provided at the application or the SOAP level. (In fact, the HTTP binding for SOAP takes advantage of just this feature of HTTP.)

However, a SOAP message may travel over several hops between a sender and the ultimate receiver, where each hop may be a different protocol binding. In other words, a feature (e.g., message correlation, reliability etc.) that is supported by the protocol binding in one hop may not be supported by another along the message path. SOAP itself does not provide any mechanism for hiding the differences in features provided by different underlying protocols. However, any feature that is required by a particular application, but which may not be available in the underlying infrastructure along the *anticipated* message path, can be compensated for by being carried as a part of the SOAP message infoset, i.e., as a SOAP header block specified in some module.

Thus it is apparent that there are a number of issues that have to be tackled by an application designer to accomplish particular application semantics, including how to take advantage of the native features of underlying protocols that are available for use in the chosen environment. SOAP Part 1 section 4.2 provides a general framework for describing how SOAP-based applications may choose to use the features provided by an underlying protocol binding to accomplish particular application semantics. It is intended to provide guidelines for writing interoperable protocol binding specifications for exchanging SOAP messages.

Among other things, a binding specification must define one particular feature, namely the message exchange pattern(s) that it supports. [SOAP Part2] defines two such message exchange patterns, namely a SOAP Request-Response message exchange pattern where one SOAP message is exchanged in each direction between two adjacent SOAP nodes, and a SOAP Response message exchange pattern which consists of a non-SOAP message acting as a request followed by a SOAP message included as a part of the response.

[SOAP Part2] also offers the application designer a general feature called the SOAP Web Method feature that allows applications full control over the choice of the so-called "Web method" - one of GET, POST, PUT, DELETE whose semantics are as defined in the [HTTP 1.1] specifications - that may be used over the binding. This feature is defined to ensure that applications using SOAP can do so in a manner which is compatible with the architectural principles of the World Wide Web. (Very briefly, the simplicity and scalability of the Web is largely due to the fact that there are a few "generic" methods (GET, POST, PUT, DELETE) which can be used to interact with any resource made available on the Web via a URI.) The SOAP Web Method feature is supported by the SOAP HTTP binding, although, in principle, it is available to all SOAP underlying protocol bindings.

SOAP Part 2 section 7 specifies one standardized protocol binding using the binding framework of [SOAP Part1], namely how SOAP is used in conjunction with HTTP as the underlying protocol. SOAP Version 1.2 restricts itself to the definition of a HTTP binding allowing only the use of the POST method in conjunction with the Request-Response message exchange pattern and the GET method with the SOAP Response message exchange pattern. Other specifications in future could define SOAP bindings to HTTP or other transports that make use of the other Web methods (i.e., PUT, DELETE).

The next sections show examples of two underlying protocol bindings for SOAP, namely those to [HTTP 1.1] and email. It should be emphasized again that the only normative binding for SOAP 1.2 messages is to [HTTP 1.1]. The examples in section 4.2 showing email as a transport mechanism for SOAP is simply meant to suggest that other choices for the transfer of SOAP messages are possible, although not standardized at this time. A W3C Note [SOAP Email Binding] offers an application of the SOAP protocol binding framework of [SOAP Part1] by describing an experimental binding of SOAP to email transport, specifically [RFC 2822]-based message transport.

4.1 The SOAP HTTP Binding

HTTP has a well-known connection model and a message exchange pattern. The client identifies the server via a URI, connects to it using the underlying TCP/IP network, issues a HTTP request message and receives a HTTP response message over the same TCP connection. HTTP implicitly correlates its request message with its response message; therefore, an application using this binding can chose to infer a correlation between a SOAP message sent in the body of a HTTP request message and a SOAP message returned in the HTTP response. Similarly, HTTP identifies the server endpoint via a URI, the Request-URI, which can also serve as the identification of a SOAP node at the server.

HTTP allows for multiple intermediaries between the initial client and the origin server identified by the Request-URI, in which case the request/response model is a series of such pairs. Note, however, that HTTP intermediaries are distinct from SOAP intermediaries.

The HTTP binding in [SOAP Part2] makes use of the SOAP Web Method feature to allow applications to choose the so-called Web method - restricting it to one of GET or POST - to use over the HTTP message exchange. In addition, it makes use of two message exchange patterns that offer applications two ways of exchanging SOAP messages via HTTP: 1) the use of the HTTP POST method for conveying SOAP messages in the bodies of HTTP request and response messages, and 2) the use of the HTTP GET method in a HTTP request to return a SOAP message in the body of a HTTP response. The first usage pattern is the HTTP-specific instantiation of a binding feature called the SOAP request-response message exchange pattern, while the second uses a feature called the SOAP response message exchange pattern.

The purpose of providing these two types of usages is to accommodate the two interaction paradigms which are well established on the World Wide Web. The first type of interaction allows for the use of data within the body of a HTTP POST to create or modify the state of a resource identified by the URI to which the HTTP request is destined. The second type of interaction pattern offers the ability to use a HTTP GET request to obtain a representation of a resource without altering its state in any way. In the first case, the SOAP-specific aspect of concern is that the body of the HTTP POST request is a SOAP message which has to be processed (per the SOAP processing model) as a part of the application-specific processing required to conform to the POST semantics. In the second case, the typical usage that is forseen is the case where the representation of the resource that is being requested is returned not as a HTML, or indeed a generic XML document, but as a SOAP message. That is, the HTTP content type header of the response message identifies it as being of media type "application/soap+xml". Presumably, there will be publishers of resources on the Web who determine that such resources are best retrieved and made available in the form of SOAP messages. Note, however, that resources can, in general, be made available in multiple representations, and the desired or preferred representation is indicated by the requesting application using the HTTP Accept header.

One further aspect of the SOAP HTTP binding is the question of how an application determines which of these two types of message exchange patterns to use. [SOAP Part2] offers guidance on circumstances when applications may use one of the two specified message exchange patterns. (It

is guidance - albeit a strong one - as it is phrased in the form of a "SHOULD" in the specifications rather than an absolute requirement identified by the word "MUST", where these words are interpreted as defined in the IETF [RFC 2119].) The SOAP response message exchange pattern with the HTTP GET method is used when an application is assured that the message exchange is for the purposes of information retrieval, where the information resource is "untouched" as a result of the interaction. Such interactions are referred to as safe and idempotent in the HTTP specification. As the HTTP SOAP GET usage does not allow for a SOAP message in the request, applications that need features in the outbound interaction that can only be supported by a binding-specific expression within the SOAP infoset (i.e., as SOAP header blocks) obviously cannot make use of this message exchange pattern. Note that the HTTP POST binding is available for use in all cases.

The following subsections provide examples of the use of these two message exchange patterns defined for the HTTP binding.

4.1.1. SOAP HTTP GET Usage

Using the HTTP binding with the SOAP Response message exchange pattern is restricted to the HTTP GET method. This means that the response to a HTTP GET request from a requesting SOAP node is a SOAP message in the HTTP response.

Example 8a shows a HTTP GET directed by the traveller's application (in the continuing travel reservation scenario) at the URI `http://travelcompany.example.org/reservations?code=FT35ZBQ` where the traveler's itinerary may be viewed. (How this URL was made available can be seen in Example 5a.)

Example 8a

```
GET /travelcompany.example.org/reservations?
code=FT35ZBQ  HTTP/1.1
Host: travelcompany.example.org
Accept: text/html;q=0.5, application/soap+xml
```

HTTP GET Request

The HTTP Accept header is used to indicate the preferred representation of the resource being requested, which in this example is an "application/soap +xml" media type for consumption by a machine client, rather than the "text/ html" media type for rendition by a browser client for consumption by a human.

Example 8b shows the HTTP response to the GET in Example 8a. The body of the HTTP response contains a SOAP message showing the travel details. A discussion of the contents of the SOAP message is postponed until section 5.2 , as it is not relevant, at this point, to understanding the HTTP GET binding usage.

Example 8b

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.
example.org/reservation"
       env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-
pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-30T16:25:00.000-05:00</m:
dateAndTime>
  </m:reservation>
 </env:Header>
 <env:Body>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"
            xmlns:x="http://travelcompany.example.org/
vocab#"
     env:encodingStyle="http://www.w3.org/1999/02/22-
rdf-syntax-ns#">
   <x:ReservationRequest
 rdf:about="http://travelcompany.example.org/
reservations?code=FT35ZBQ">
      <x:passenger>Åke Jógvan Øyvind</x:passenger>
      <x:outbound>
       <x:TravelRequest>
        <x:to>LAX</x:to>
        <x:from>LGA</x:from>
        <x:date>2001-12-14</x:date>
       </x:TravelRequest>
      </x:outbound>
      <x:return>
```

```
        <x:TravelRequest>
         <x:to>JFK</x:to>
         <x:from>LAX</x:from>
         <x:date>2001-12-20</x:date>
        </x:TravelRequest>
       </x:return>
     </x:ReservationRequest>
   </rdf:RDF>
  </env:Body>
 </env:Envelope>
```

SOAP message returned as a response to the HTTP GET in Example 8a

Note that the reservation details could well have been returned as an (X) HTML document, but this example wanted to show a case where the reservation application is returning the state of the resource (the reservation) in a data-centric media form (a SOAP message) which can be machine processed, instead of (X)HTML which would be processed by a browser. Indeed, in the most likely anticipated uses of SOAP, the consuming application will not be a browser.

Also, as shown in the example, the use of SOAP in the HTTP response body offers the possibility of expressing some application-specific feature through the use of SOAP headers. By using SOAP, the application is provided with a useful and consistent framework and processing model for expressing such features.

4.1.2 SOAP HTTP POST Usage

Using the HTTP binding with the SOAP Request-Response message exchange pattern is restricted to the HTTP POST method. Note that the use of this message exchange pattern in the SOAP HTTP binding is available to all applications, whether they involve the exchange of general XML data or RPCs (as in the following examples) encapsulated in SOAP messages.

Examples 9 and 10 show an example of a HTTP binding using the SOAP Request-Response message exchange pattern, using the same scenario as that for Example 4 and Example 5a, respectively, namely conveying an RPC and its return in the body of a SOAP message. The examples and discussion in this section only concentrate on the HTTP headers and their role.

Example 9

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
```

```
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope" >
 <env:Header>
   <t:transaction
           xmlns:t="http://thirdparty.example.org/
transaction"
           env:encodingStyle="http://example.com/
encoding"
           env:mustUnderstand="true" >5</t:
transaction>
 </env:Header>
 <env:Body>
  <m:chargeReservation
     env:encodingStyle="http://www.w3.org/2003/05/
soap-encoding"
          xmlns:m="http://travelcompany.example.org/">
   <m:reservation xmlns:m="http://travelcompany.
example.org/reservation">
    <m:code>FT35ZBQ</m:code>
   </m:reservation>
    <o:creditCard xmlns:o="http://mycompany.example.
com/financial">
     <n:name xmlns:n="http://mycompany.example.com/
employees">
           Åke Jógvan Øyvind
     </n:name>
     <o:number>123456789099999</o:number>
     <o:expiration>2005-02</o:expiration>
    </o:creditCard>
   </m:chargeReservation>
  </env:Body>
</env:Envelope>
```

RPC in Example 4 carried in an HTTP POST Request

Example 9 shows an RPC request directed at the travel service application. The SOAP message is sent in the body of a HTTP POST method directed at the URI identifying the "Reservations" resource on the server travelcompany.example.org. When using HTTP, the Request-URI indicates the resource to which the invocation is "posted". Other than requiring that it be a valid URI, SOAP places no *formal* restriction on the form of the request URI (see [RFC

2396] for more information on URIs). However, one of the principles of the Web architecture is that all important resources be identified by URIs. This implies that most well-architected SOAP services will be embodied as a large number of resources, each with its own URI. Indeed, many such resources are likely to be created dynamically during the operation of a service, such as, for instance, the specific travel reservation shown in the example. So, a well-architected travel service application should have different URIs for each reservation, and SOAP requests to retrieve or manipulate those reservations will be directed at their URIs, and not at a single monolithic "Reservations" URI, as shown in Example 9. Example 13 in section 4.1.3 shows the preferred way to address resources such as a particular travel reservation. Therefore, we defer until section 4.1.3 further discussion of Web architecture compatible SOAP/HTTP usage.

When placing SOAP messages in HTTP bodies, the HTTP Content-type header must be chosen as "application/soap+xml". (The optional charset parameter, which can take the value of "utf-8" or "utf-16", is shown in this example, but if it is absent the character set rules for freestanding [XML 1.0] apply to the body of the HTTP request.)

Example 10 shows the RPC return (with details omitted) sent by the travel service application in the corresponding HTTP response to the request from Example 5a. SOAP, using HTTP transport, follows the semantics of the HTTP status codes for communicating status information in HTTP. For example, the 2xx series of HTTP status codes indicate that the client's request (including the SOAP component) was successfully received, understood, and accepted etc.

Example 10

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope" >
 <env:Header>
       ...
       ...
 </env:Header>
 <env:Body>
       ...
       ...
 </env:Body>
```

```
  </env:Envelope>
```

RPC return in Example 5a embedded in an HTTP Response indicating a successful completion

If an error occurs processing the request, the HTTP binding specification requires that a HTTP 500 "Internal Server Error" be used with an embedded SOAP message containing a SOAP fault indicating the server-side processing error.

Example 11 is the same SOAP fault message as Example 6a, but this time with the HTTP headers added.

Example 11

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
  <env:Body>
    <env:Fault>
     <env:Code>
       <env:Value>env:Sender</env:Value>
       <env:Subcode>
        <env:Value>rpc:BadArguments</env:Value>
       </env:Subcode>
     </env:Code>
     <env:Reason>
      <env:Text xml:lang="en-US">Processing error</
env:Text>
       <env:Text xml:lang="cs">Chyba zpracování</env:
Text>
     </env:Reason>
     <env:Detail>
      <e:myFaultDetails
        xmlns:e="http://travelcompany.example.org/
faults" >
        <e:message>Name does not match card number</e:
message>
        <e:errorcode>999</e:errorcode>
      </e:myFaultDetails>
     </env:Detail>
   </env:Fault>
```

```
  </env:Body>
</env:Envelope>
```

Sample SOAP message in a HTTP Response indicating failure to handle the SOAP Body in Example 4

SOAP Part 2 Table 16 provides detailed behavior for handling the various possible HTTP response codes, i.e., the 2xx (successful), 3xx (redirection), 4xx (client error) and 5xx (server error).

4.1.3 Web Architecture Compatible SOAP Usage

One of the most central concepts of the World Wide Web is that of a URI as a resource identifier. SOAP services that use the HTTP binding and wish to interoperate with other Web software should use URIs to address all important resources in their service. For example, a very important - indeed predominant - use of the World Wide Web is pure information retrieval, where the representation of an available resource, identified by a URI, is fetched using a HTTP GET request without affecting the resource in any way. (This is called a safe and idempotent method in HTTP terminology.) The key point is that the publisher of a resource makes available its URI, which consumers may "GET".

There are many instances when SOAP messages are designed for uses which are purely for information retrieval, such as when the state of some resource (or object, in programming terms) is requested, as opposed to uses that perform resource manipulation. In such instances, the use of a SOAP body to carry the request for the state, with an element of the body representing the object in question, is seen as counter to the spirit of the Web because the resource is not identified by the Request-URI of the HTTP GET. (In some SOAP/RPC implementations, the HTTP Request-URI is often not the identifier of the resource itself but some intermediate entity which has to evaluate the SOAP message to identify the resource.)

To highlight the changes needed, Example 12a shows the way that is **not** recommended for doing safe information retrieval on the Web. This is an example of an RPC carried in a SOAP message, again using the travel reservation theme, where the request is to retrieve the itinerary for a particular reservation identified by one of the parameters, `reservationCode`, of the RPC. (For purposes of this discussion, it is assumed that the application using this RPC request does not need features which require the use of SOAP headers.)

Example 12a

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope" >
  <env:Body>
    <m:retrieveItinerary
        env:encodingStyle="http://www.w3.org/2003/05/
soap-encoding"
            xmlns:m="http://travelcompany.example.
org/">
      <m:reservationCode>FT35ZBQ</m:reservationCode>
    </m:retrieveItinerary>
  </env:Body>
</env:Envelope>
```

This representation is discouraged in cases where the operation is a "safe" retrieval (i.e., it has no side effects)

Note that the resource to be retrieved is not identified by the target URI in the HTTP request but has to be obtained by looking within the SOAP envelope. Thus, it is not possible, as would be the case with other "gettable" URIs on the Web, to make this available via HTTP alone to consumers on the World Wide Web.

SOAP Part 2 section 4.1 offers recommendations on how RPCs that constitute safe and idempotent information retrievals may be defined in a Web-friendly manner. It does so by distinguishing aspects of the method and specific parameters in an RPC definition that serve to identify resources from those that serve other purposes. In Example 12a, the resource to be retrieved is identified by two things: the first is that it is an itinerary (part of the method name), and the second is the reference to a specific instance (a parameter to the method). In such a case, the recommendation is that these resource-identifying parts be made available in the HTTP Request-URI identifying the resource, as for example, as follows: `http://travelcompany.example.org/reservations/itinerary?reservationCode=FT35ZBQ`.

Furthermore, when an RPC definition is such that all parts of its method description can be described as resource-identifying, the entire target of the RPC may be identified by a URI. In this case, if the supplier of the resource can also assure that a retrieval request is safe, then SOAP Version 1.2 recommends that the choice of the Web method property of GET and the use

of the SOAP Response message exchange pattern be used as described in section 4.1.1. This will ensure that the SOAP RPC is performed in a Web architecture compatible manner. Example 12b shows the preferred way for a SOAP node to request the safe retrieval of a resource.

Example 12b

```
GET /Reservations/itinerary?reservationCode=FT35ZBQ
HTTP/1.1
Host: travelcompany.example.org
Accept: application/soap+xml
```

The Web architecture compatible alternative to representing the RPC in Example 12a

It should be noted that SOAP Version 1.2 does not specify any algorithm on how to compute a URI from the definition of an RPC which has been determined to represent pure information retrieval.

Note, however, that if the application requires the use of features that can only have a binding-specific expression within the SOAP infoset, i.e., using SOAP header blocks, then the application must choose HTTP POST method with a SOAP message in the request body.

It also requires the use of the SOAP Request-Response message exchange pattern implemented via a HTTP POST if the RPC description includes data (parameters) which are not resource-identifying. Even in this case, the HTTP POST with a SOAP message can be represented in a Web-friendly manner. As with the use of the GET, [SOAP Part2] recommends for the general case that any part of the SOAP message that serves to identify the resource to which the request is POSTed be identified in the HTTP Request-URI. The same parameters may, of course, be retained in the SOAP `env:Body` element. (The parameters must be retained in the Body in the case of a SOAP-based RPC as these are related to the procedure/method description expected by the receiving application.)

Example 13 is the same as that in Example 9, except that the HTTP Request-URI has been modified to include the reservation `code`, which serves to identify the resource (the reservation in question, which is being confirmed and paid for).

Example 13

```
POST /Reservations?code=FT35ZBQ HTTP/1.1
```

```
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope" >
 <env:Header>
   <t:transaction
           xmlns:t="http://thirdparty.example.org/
transaction"
           env:encodingStyle="http://example.com/
encoding"
           env:mustUnderstand="true" >5</t:
transaction>
 </env:Header>
 <env:Body>
  <m:chargeReservation
      env:encodingStyle="http://www.w3.org/2003/05/
soap-encoding"
         xmlns:m="http://travelcompany.example.org/">
   <m:reservation xmlns:m="http://travelcompany.
example.org/reservation">
    <m:code>FT35ZBQ</m:code>
   </m:reservation>
   <o:creditCard xmlns:o="http://mycompany.example.
com/financial">
    <n:name xmlns:n="http://mycompany.example.com/
employees">
           Åke Jógvan Øyvind
    </n:name>
    <o:number>123456789099999</o:number>
    <o:expiration>2005-02</o:expiration>
   </o:creditCard>
  </m:chargeReservation>
 </env:Body>
</env:Envelope>
```

RPC from Example 4 carried in an HTTP POST Request in a Web-friendly manner

In Example 13, the resource to be manipulated is identified by two things: the first is that it is a reservation (part of the method name), and the second is the specific instance of a reservation (which is the value of the parameter `code` to the method). The remainder of the parameters in the RPC such as the `creditCard` number are not resource-identifying, but are ancillary data to be

processed by the resource. It is the recommendation of [SOAP Part2] that resources that may be accessed by SOAP-based RPCs should, where practical, place any such resource-identifying information as a part of the URI identifying the target of the RPC. It should be noted, however, that [SOAP Part2] does not offer any algorithm to do so. Such algorithms may be developed in future. Note, however, that all the resource-identifying elements have been retained as in Example 9 in their encoded form in the SOAP `env:Body` element.

In other words, as seen from the above examples, the recommendation in the SOAP specifications is to use URIs in a Web-architecture compatible way - that is, as resource identifiers - whether or not it is GET or POST that is used.

4.2 SOAP Over Email

Application developers can use the Internet email infrastructure to move SOAP messages as either email text or attachments. The examples shown below offer one way to carry SOAP messages, and should not be construed as being the standard way of doing so. The SOAP Version 1.2 specifications do not specify such a binding. However, there is a *non-normative* W3C Note [SOAP Email Binding] describing an email binding for SOAP, its main purpose being to demonstrate the application of the general SOAP Protocol Binding Framework described in [SOAP Part 1].

Example 14 shows the travel reservation request message from Example 1 carried as an email message between a sending and receiving mail user agent. It is implied that the receiver node has SOAP capabilities, to which the body of the email is delivered for processing. (It is assumed that the sending node also has SOAP capabilities so as to be able to process any SOAP faults received in response, or to correlate any SOAP messages received in response to this one.)

Example 14

```
From: a.oyvind@mycompany.example.com
To: reservations@travelcompany.example.org
Subject: Travel to LA
Date: Thu, 29 Nov 2001 13:20:00 EST
Message-Id:
<EE492E16A090090276D208424960C0C@mycompany.example.
com>
Content-Type: application/soap+xml

<?xml version='1.0' ?>
```

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.
example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
         env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-
pqff98fe8j7d</reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:
dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/
employees"
      env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
         env:mustUnderstand="true">
   <n:name>Åke Jógvan Øyvind</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary
     xmlns:p="http://travelcompany.example.org/
reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2001-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2001-12-20</p:departureDate>
     <p:departureTime>mid morning</p:departureTime>
     <p:seatPreference/>
   </p:return>
  </p:itinerary>
  <q:lodging
     xmlns:q="http://travelcompany.example.org/
reservation/hotels">
   <q:preference>none</q:preference>
  </q:lodging>
 </env:Body>
```

```
</env:Envelope>
```

SOAP message from Example 1 carried in a SMTP message

The header in Example 14 is in the standard form [RFC 2822] for email messages.

Although an email is a one-way message exchange, and no guarantee of delivery is provided, email infrastructures like the Simple Mail Transport Protocol (SMTP) specification [SMTP] offer a delivery notification mechanism which, in the case of SMTP, are called Delivery Status Notification (DSN) and Message Disposition Notification (MDN). These notifications take the form of email messages sent to the email address specified in the mail header. Applications, as well as email end users, can use these mechanisms to provide the status of an email transmission, but these, if delivered, are notifications at the SMTP level. The application developer must fully understand the capabilities and limitations of these delivery notifications or risk assuming a successful data delivery when none occurred.

SMTP delivery status messages are separate from message processing at the SOAP layer. Resulting SOAP responses to the contained SOAP data will be returned through a new email message which may or may not have a link to the original requesting email at the SMTP level. The use of the [RFC 2822] `In-reply-to:` header can achieve a correlation at the SMTP level, but does not necessarily offer a correlation at the SOAP level.

Example 15 is exactly the same scenario as described for Example 2, which shows the SOAP message (body details omitted for brevity) sent from the travel service application to the travel reservation application seeking clarification on some reservation details, except that it is carried as an email message. In this example, the original email's `Message-Id` is carried in the additional email header `In-reply-to:`, which correlates email messages at the SMTP level, but cannot provide a SOAP-specific correlation. In this example, the application relies on the `reservation` header block to correlate SOAP messages. Again, how such correlation is achieved is application-specific, and is not within the scope of SOAP.

Example 15

```
From: reservations@travelcompany.example.org
To: a.oyvind@mycompany.example.com
Subject: Which NY airport?
Date: Thu, 29 Nov 2001 13:35:11 EST
Message-Id: <200109251753.NAA10655@travelcompany.
example.org>
```

```
In-reply-to:
<EE492E16A090090276D208424960C0C@mycompany.example.
com>
Content-Type: application/soap+xml

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.
example.org/reservation"
     env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
       env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-
pqff98fe8j7d</reference>
   <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:
dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/
employees"
      env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
        env:mustUnderstand="true">
   <n:name>Åke Jógvan Øyvind</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary
     xmlns:p="http://travelcompany.example.org/
reservation/travel">
   <p:itineraryClarifications>
      ...
      ...
   </p:itineraryClarifications>
  </p:itinerary>
 </env:Body>
</env:Envelope>
```

SOAP message from Example 2 carried in an email message with a header correlating it to a previous message.

5. Advanced Usage Scenarios

5.1 Using SOAP Intermediaries

The travel reservation scenario used throughout the primer offers an opportunity to expose some uses of SOAP intermediaries. Recall that the basic exchange was the exchange of a travel reservation request between a travel reservation application and a travel service application. SOAP does not specify how such a message path is determined and followed. That is outside the scope of the SOAP specification. It does describe, though, how a SOAP node should behave if it receives a SOAP message for which it is not the ultimate receiver. SOAP Version 1.2 describes two types of intermediaries: forwarding intermediaries and active intermediaries.

A forwarding intermediary is a SOAP node which, based on the semantics of a header block in a received SOAP message or based on the message exchange pattern in use, forwards the SOAP message to another SOAP node. For example, processing a "routing" header block describing a message path feature in an incoming SOAP message may dictate that the SOAP message be forwarded to another SOAP node identified by data in that header block. The format of the SOAP header of the outbound SOAP message, i.e., the placement of inserted or reinserted header blocks, is determined by the overall processing at this *forwarding* intermediary based on the semantics of the processed header blocks.

An active intermediary is one that does additional processing on an incoming SOAP message before forwarding the message using criteria that are not described by incoming SOAP header blocks, or by the message exchange pattern in use. Some examples of such active intervention at a SOAP node could be, for instance, encrypting some parts of a SOAP message and providing the information on the cipher key in a header block, or including some additional information in a new header block in the outbound message providing a timestamp or an annotation, for example, for interpretation by appropriately targeted nodes downstream.

One mechanism by which an active intermediary can describe the modifications performed on a message is by inserting header blocks into the outbound SOAP message. These header blocks can inform downstream SOAP nodes acting in roles whose correct operation depends on receiving such notification. In this case, the semantics of such inserted header blocks should also call for either the same or other header blocks to be (re)inserted at subsequent intermediaries as necessary to ensure that the message can be safely processed by nodes yet further downstream. For example, if a message with header blocks removed for encryption passes through a second intermediary (without the original header blocks being decrypted and reconstructed), then indication that the encryption has occurred must be retained in the second relayed message.

In the following example, a SOAP node is introduced in the message path between the travel reservation and travel service applications, which intercepts the message shown in Example 1. An example of such a SOAP node is one which logs all travel requests for off-line review by a corporate travel office. Note that the header blocks `reservation` and `passenger` in that example are intended for the node(s) that assume the role "next", which means that it is targeted at the next SOAP node in the message path that receives the message. The header blocks are mandatory (the `mustUnderstand` attribute is set to "true"), which means that the node must have knowledge (through an external specification of the header blocks' semantics) of what to do. A logging specification for such header blocks might simply require that various details of the message be recorded at every node that receives such a message, and that the message be relayed along the message path unchanged. (Note that the specifications of the header blocks must require that the same header blocks be reinserted in the outbound message, because otherwise, the SOAP processing model would require that they be removed.) In this case, the SOAP node acts as a forwarding intermediary.

A more complex scenario is one where the received SOAP message is amended in some way not anticipated by the initial sender. In the following example, it is assumed that a corporate travel application at the SOAP intermediary attaches a header block to the SOAP message from Example 1 before relaying it along its message path towards the travel service application - the ultimate recipient. The header block contains the constraints imposed by a travel policy for this requested trip. The specification of such a header block might require that the ultimate recipient (and only the ultimate recipient, as implied by the absence of the `role` attribute) make use of the information conveyed by it when processing the body of the message.

Example 16 shows an active intermediary inserting an additional header block, `travelPolicy`, intended for the ultimate recipient which includes information that qualifies the application-level processing of this travel request.

Example 16

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.
example.org/reservation"
     env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
        env:mustUnderstand="true">
```

```
    <m:reference>uuid:093a2da1-q345-739r-ba5d-
pqff98fe8j7d</reference>
    <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:
dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/
employees"
     env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
       env:mustUnderstand="true">
    <n:name>Åke Jógvan Øyvind</n:name>
  </n:passenger>
  <z:travelPolicy
    xmlns:z="http://mycompany.example.com/policies"
      env:mustUnderstand="true">
    <z:class>economy</z:class>
    <z:fareBasis>non-refundable<z:fareBasis>
    <z:exceptions>none</z:exceptions>
  </z:travelPolicy>
 </env:Header>
 <env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/
reservation/travel">
    <p:departure>
      <p:departing>New York</p:departing>
      <p:arriving>Los Angeles</p:arriving>
      <p:departureDate>2001-12-14</p:departureDate>
      <p:departureTime>late afternoon</p:departureTime>
      <p:seatPreference>aisle</p:seatPreference>
    </p:departure>
    <p:return>
      <p:departing>Los Angeles</p:departing>
      <p:arriving>New York</p:arriving>
      <p:departureDate>2001-12-20</p:departureDate>
      <p:departureTime>mid morning</p:departureTime>
      <p:seatPreference/>
    </p:return>
  </p:itinerary>
  <q:lodging
    xmlns:q="http://travelcompany.example.org/
reservation/hotels">
    <q:preference>none</q:preference>
  </q:lodging>
 </env:Body>
</env:Envelope>
```

SOAP message from Example 1 for a travel reservation after an active intermediary has inserted a mandatory header intended for the ultimate recipient of the message

5.2 Using Other Encoding Schemes

Even though SOAP Version 1.2 defines a particular encoding scheme (see SOAP Part 2 section 3), its use is optional and the specification makes clear that other encoding schemes may be used for application-specific data within a SOAP message. For this purpose it provides the attribute `env:encodingStyle`, of type `xs:anyURI`, to qualify header blocks, any child elements of the SOAP `env:Body`, and any child elements of the `env:Detail` element and their descendants. It signals a serialization scheme for the nested contents, or at least the one in place until another element is encountered which indicates another encoding style for its nested contents. The choice of the value for the `env:encodingStyle` attribute is an application-specific decision and the ability to interoperate is assumed to have been settled "out-of-band". If this attribute is not present, then no claims are being made about the encoding being used.

The use of an alternative encoding scheme is illustrated in Example 17. Continuing with the travel reservation theme, this example shows a SOAP message which is sent to the passenger from the travel service after the reservation is confirmed, showing the travel details. (The same message was used in Example 8b in another context.)

Example 17

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.
example.org/reservation"
     env:role="http://www.w3.org/2003/05/soap-
envelope/role/next"
        env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-
pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-30T16:25:00.000-05:00</m:
dateAndTime>
  </m:reservation>
 </env:Header>
 <env:Body>
```

```
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-
rdf-syntax-ns#"
           xmlns:x="http://travelcompany.example.org/
vocab#"
    env:encodingStyle="http://www.w3.org/1999/02/22-
rdf-syntax-ns#">
   <x:ReservationRequest
 rdf:about="http://travelcompany.example.org/
reservations?code=FT35ZBQ">
      <x:passenger>Åke Jógvan Øyvind</x:passenger>
      <x:outbound>
       <x:TravelRequest>
        <x:to>LAX</x:to>
        <x:from>LGA</x:from>
        <x:date>2001-12-14</x:date>
       </x:TravelRequest>
      </x:outbound>
      <x:return>
       <x:TravelRequest>
        <x:to>JFK</x:to>
        <x:from>LAX</x:from>
        <x:date>2001-12-20</x:date>
       </x:TravelRequest>
      </x:return>
    </x:ReservationRequest>
  </rdf:RDF>
 </env:Body>
</env:Envelope>
```

SOAP message showing the use of an alternative encoding for the `Body` element

In Example 17, the body of the SOAP message contains a description of the itinerary using the encoding of a graph of resources and their properties using the syntax of the Resource Description Framework (RDF) [RDF]. (Very briefly, as RDF syntax or usage is not the subject of this primer, an RDF graph relates resources - such as the travel reservation resource available at `http://travelcompany.example.org/reservations?code=FT35ZBQ` - to other resources (or values) via properties, such as the `passenger`, the `outbound` and `return` dates of travel. The RDF encoding for the itinerary might have been chosen, for example, to allow the passenger's travel application to store it in an RDF-capable calendar application, which could then be queried in complex ways.)

6. Changes Between SOAP 1.1 and SOAP

1.2

SOAP Version 1.2 has a number of changes in syntax and provides additional (or clarified) semantics from those described in [SOAP 1.1]. The following is a list of features where the two specifications differ. The purpose of this list is to provide the reader with a quick and easily accessible summary of the differences between the two specifications. The features have been put in categories purely for ease of reference, and in some cases, an item might equally well have been placed in another category.

Document structure

- The SOAP 1.2 specifications have been provided in two parts. [SOAP Part1] provides an abstract Infoset-based definition of the SOAP message structure, a processing model and an underlying protocol binding framework, while [SOAP Part2] provides serialization rules for conveying that infoset as well as a particular HTTP binding.
- SOAP 1.2 will not spell out the acronym.
- SOAP 1.2 has been rewritten in terms of XML infosets, and not as serializations of the form <?xml....?> required by SOAP 1.1.

Additional or changed syntax

- SOAP 1.2 does not permit any element after the body. The SOAP 1.1 schema definition allowed for such a possibility, but the textual description is silent about it.
- SOAP 1.2 does not allow the `env:encodingStyle` attribute to appear on the SOAP `env:Envelope`, whereas SOAP 1.1 allows it to appear on any element. SOAP 1.2 specifies specific elements where this attribute may be used.
- SOAP 1.2 defines the new `env:NotUnderstood` header element for conveying information on a mandatory header block which could not be processed, as indicated by the presence of an `env:MustUnderstand` fault code. SOAP 1.1 provided the fault code, but no details on its use.
- In the SOAP 1.2 infoset-based description, the `env:mustUnderstand` attribute in header elements takes the (logical) value "true" or "false", whereas in SOAP 1.1 they are the literal value "1" or "0" respectively.
- SOAP 1.2 provides a new fault code `DataEncodingUnknown`.
- The various namespaces defined by the two protocols are of course different.
- SOAP 1.2 replaces the attribute `env:actor` with `env:role` but with essentially the same semantics.
- SOAP 1.2 defines a new attribute, `env:relay`, for header blocks to indicate if unprocessed header blocks should be forwarded.

- SOAP 1.2 defines two new roles, "none" and "ultimateReceiver", together with a more detailed processing model on how these behave.
- SOAP 1.2 has removed the "dot" notation for fault codes, which are now simply an XML Qualified Name, where the namespace prefix is the SOAP envelope namespace.
- SOAP 1.2 replaces "client" and "server" fault codes with "Sender" and "Receiver".
- SOAP 1.2 uses the element names `env:Code` and `env:Reason`, respectively, for what used to be called `faultcode` and `faultstring` in SOAP 1.1. SOAP 1.2 also allows multiple `env:Text` child elements of `env:Reason` qualified by `xml:lang` to allow multiple language versions of the fault reason.
- SOAP 1.2 provides a hierarchical structure for the mandatory SOAP `env:Code` sub-element in the `env:Fault` element, and introduces two new optional subelements, `env:Node` and `env:Role`.
- SOAP 1.2 removes the distinction that was present in SOAP 1.1 between header and body faults as indicated by the presence of the `env:Details` element in `env:Fault`. In SOAP 1.2, the presence of the `env:Details` element has no significance as to which part of the fault SOAP message was processed.
- SOAP 1.2 uses XML Base [XML Base] for determining a base URI for relative URI references whereas SOAP 1.1 is silent about the matter.

SOAP HTTP binding

- In the SOAP 1.2 HTTP binding, the `SOAPAction` HTTP header defined in SOAP 1.1 has been removed, and a new HTTP status code 427 has been sought from IANA for indicating (at the discretion of the HTTP origin server) that its presence is required by the server application. The contents of the former `SOAPAction` HTTP header are now expressed as a value of an (optional) "action" parameter of the "application/soap+xml" media type that is signaled in the HTTP binding.
- In the SOAP 1.2 HTTP binding, the Content-type header should be "application/soap+xml" instead of "text/xml" as in SOAP 1.1. The IETF registration for this new media type is pending [SOAP MediaType].
- SOAP 1.2 provides a finer grained description of use of the various 2xx, 3xx, 4xx HTTP status codes.
- Support of the HTTP extensions framework has been removed from SOAP 1.2.
- SOAP 1.2 provides an additional message exchange pattern which may be used as a part of the HTTP binding that allows the use of HTTP GET for safe and idempotent information retrievals.

RPC

- SOAP 1.2 provides a `rpc:result` element accessor for RPCs.
- SOAP 1.2 provides several additional fault codes in the RPC namespace.
- SOAP 1.2 offers guidance on a Web-friendly approach to defining RPCs where the procedure's purpose is purely "safe" informational retrieval.

SOAP encodings

- An abstract data model based on a directed edge labeled graph has been formulated for SOAP 1.2. The SOAP 1.2 encodings are dependent on this data model. The SOAP RPC conventions are dependent on this data model, but have no dependencies on the SOAP encoding. Support of the SOAP 1.2 encodings and SOAP 1.2 RPC conventions are optional.
- The syntax for the serialization of an array has been changed in SOAP 1.2 from that in SOAP 1.1.
- The support provided in SOAP 1.1 for partially transmitted and sparse arrays is not available in SOAP 1.2.
- SOAP 1.2 allows the inline (embedded) serialization of multiref values.
- The `href` attribute in SOAP 1.1 (of type `xs:anyURI`) is called `enc:ref` in SOAP 1.2 and is of type `IDREF`.
- In SOAP 1.2, omitted accessors of compound types are made equal to NILs.
- SOAP 1.2 provides several fault sub-codes for indicating encoding errors.
- Types on nodes are made optional in SOAP 1.2.
- SOAP 1.2 has removed generic compound values from the SOAP Data Model.
- SOAP 1.2 has added an optional attribute `enc:nodeType` to elements encoded using SOAP encoding that identifies its structure (i.e., a simple value, a struct or an array).

SOAP Part 1 Appendix A provides version management rules for a SOAP node that can support the version transition from [SOAP 1.1] to SOAP Version 1.2. In particular, in defines an `env:Upgrade` header block which can be used by a SOAP 1.2 node on receipt of a [SOAP 1.1] message to send a SOAP fault message to the originator to signal which version of SOAP it supports.

7. References

[SOAP Part1] W3C Recommendation "SOAP 1.2 Part 1: Messaging Framework", Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik

Frystyk Nielsen, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.)

[SOAP Part2] W3C Recommendation "SOAP 1.2 Part 2: Adjuncts", Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik Frystyk Nielsen, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part2-20030624.)

[RFC 2396] IETF "RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, August 1998. (See http://www.ietf.org/rfc/rfc2396.txt.)

[HTTP 1.1] IETF "RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, T. Berners-Lee, January 1997. (See http://www.ietf.org/rfc/rfc2616.txt.)

[XML 1.0] W3C Recommendation "Extensible Markup Language (XML) 1.0 (Second Edition)", Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, 6 October 2000. (See http://www.w3.org/TR/2000/REC-xml-20001006.

[Namespaces in XML] W3C Recommendation "Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman, 14 January 1999. (See http://www.w3.org/TR/1999/REC-xml-names-19990114/.)

[XML Schema Part1] W3C Recommendation "XML Schema Part 1: Structures", Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, 2 May 2001. (See http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/)

[XML Schema Part2] W3C Recommendation "XML Schema Part 2: Datatypes", Paul V. Biron, Ashok Malhotra, 2 May 2001. (See http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/)

[SMTP] SMTP is defined in a series of RFCs:

- RFC 2822 Internet Message Format. (See http://www.ietf.org/rfc/rfc2822.txt.)
- RFC 2045 Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. (See http://www.ietf.org/rfc/rfc2045.txt.)
- RFC 2046 Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. (See http://www.ietf.org/rfc/rfc2046.txt.)
- RFC 1894 An Extensible Message Format for Delivery Status

Notifications. (See http://www.ietf.org/rfc/rfc1894.txt.)
- RFC 2852 Deliver By SMTP Service Extension. (See http://www.ietf.org/rfc/rfc2852.txt.)
- RFC 2298 An Extensible Message Format for Message Disposition Notifications. (See http://www.ietf.org/rfc/rfc2298.txt.)

[RDF] W3C Recommendation "Resource Description Framework (RDF) Model and Syntax Specification", O. Lassila and R. Swick, Editors. World Wide Web Consortium. 22 February 1999. (See http://www.w3.org/TR/REC-rdf-syntax.)

[SOAP 1.1] W3C Note "Simple Object Access Protocol (SOAP) 1.1", Don Box *et al*., 8 May, 2000 (See http://www.w3.org/TR/SOAP/)

[XML Infoset] W3C Recommendation "XML Information Set", John Cowan, Richard Tobin, 24 October 2001. (See http://www.w3.org/TR/xml-infoset/)

[SOAP MediaType] IETF Internet Draft "The 'application/soap+xml' media type", M. Baker, M. Nottingham, "draft-baker-soap-media-reg-03.txt", May 29, 2003. (See http://www.ietf.org/internet-drafts/draft-baker-soap-media-reg-03.txt, note that this Internet Draft expires in November 2003)

[SOAP Email Binding] W3C Note "SOAP Version 1.2 Email Binding", Highland Mary Mountain et al, draft June 13, 2002. (See http://www.w3.org/TR/2002/NOTE-soap12-email-20020626.)

[XML Base] W3C Recommendation "XML Base", Jonathan Marsh, 27 June 2001. (See http://www.w3.org/TR/2001/REC-xmlbase-20010627/)

[RFC 2119] IETF "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997. (See http://www.ietf.org/rfc/rfc2119.txt.)

A. Acknowledgements

Highland Mary Mountain (Intel) provided the initial material for the section on the SMTP binding. Paul Denning provided material for a usage scenario, which has since been moved to the SOAP Version 1.2 Usage Scenarios Working Draft. Stuart Williams, Oisin Hurley, Chris Ferris, Lynne Thompson, John Ibbotson, Marc Hadley, Yin-Leng Husband and Jean-Jacques Moreau provided detailed comments on earlier versions of this document, as did many others during the Last Call Working Draft review. Jacek Kopecky provided a

list of RPC and SOAP encoding changes.

This document is the work of the W3C XML Protocol Working Group.

Participants in the Working Group are (at the time of writing, and by alphabetical order): Carine Bournez (W3C), Michael Champion (Software AG), Glen Daniels (Macromedia, formerly of Allaire), David Fallside (IBM, Chair), Dietmar Gaertner (Software AG), Tony Graham (Sun Microsystems), Martin Gudgin (Microsoft Corporation, formerly of DevelopMentor), Marc Hadley (Sun Microsystems), Gerd Hoelzing (SAP AG), Oisin Hurley (IONA Technologies), John Ibbotson (IBM), Ryuji Inoue (Matsushita Electric), Kazunori Iwasa (Fujitsu Limited), Mario Jeckle (DaimlerChrysler R. & Tech), Mark Jones (AT&T), Anish Karmarkar (Oracle), Jacek Kopecky (Systinet/ Idoox), Yves Lafon (W3C), Michah Lerner (AT&T), Noah Mendelsohn (IBM, formerly of Lotus Development), Jeff Mischkinsky (Oracle), Nilo Mitra (Ericsson), Jean-Jacques Moreau (Canon), Masahiko Narita (Fujitsu Limited), Eric Newcomer (IONA Technologies), Mark Nottingham (BEA Systems, formerly of Akamai Technologies), David Orchard (BEA Systems, formerly of Jamcracker), Andreas Riegg (DaimlerChrysler R. & Tech), Hervé Ruellan (Canon), Jeff Schlimmer (Microsoft Corporation), Miroslav Simek (Systinet/ Idoox), Pete Wenzel (SeeBeyond), Volker Wiechers (SAP AG).

Previous participants were: Yasser alSafadi (Philips Research), Bill Anderson (Xerox), Vidur Apparao (Netscape), Camilo Arbelaez (WebMethods), Mark Baker (Idokorro Mobile (Planetfred), formerly of Sun Microsystems), Philippe Bedu (EDF (Electricité de France)), Olivier Boudeville (EDF (Electricité de France)), Don Box (Microsoft Corporation, formerly of DevelopMentor), Tom Breuel (Xerox), Dick Brooks (Group 8760), Winston Bumpus (Novell), David Burdett (Commerce One), Charles Campbell (Informix Software), Alex Ceponkus (Bowstreet), David Chappell (Sonic Software), Miles Chaston (Epicentric), David Clay (Oracle), David Cleary (Progress Software), Conleth O'Connell (Vignette), Ugo Corda (Xerox), Paul Cotton (Microsoft Corporation), Fransisco Cubera (IBM), Jim d'Augustine (eXcelon), Ron Daniel (Interwoven), Dug Davis (IBM), Ray Denenberg (Library of Congress), Paul Denning (MITRE), Frank DeRose (Tibco), Mike Dierken (DataChannel), Andrew Eisenberg (Progress Software), Brian Eisenberg (DataChannel), Colleen Evans (Sonic Software), John Evdemon (XMLSolutions), David Ezell (Hewlett-Packard), Eric Fedok (Active Data Exchange), Chris Ferris (Sun Microsystems), Daniela Florescu (Propel), Dan Frantz (BEA Systems), Michael Freeman (Engenia Software), Scott Golubock (Epicentric), Rich Greenfield (Library of Congress), Hugo Haas (W3C), Mark Hale (Interwoven), Randy Hall (Intel), Bjoern Heckel (Epicentric), Erin Hoffman (Tradia), Steve Hole (MessagingDirect Ltd.), Mary Holstege (Calico Commerce), Jim Hughes (Fujitsu Software Corporation), Yin-Leng Husband (Hewlett-Packard, formerly of Compaq), Scott Isaacson (Novell), Murali Janakiraman (Rogue Wave), Eric Jenkins (Engenia Software), Jay Kasi (Commerce One), Jeffrey Kay (Engenia

Software), Richard Koo (Vitria Technology Inc.), Alan Kropp (Epicentric), Julian Kumar (Epicentric), Peter Lecuyer (Progress Software), Tony Lee (Vitria Technology Inc.), Amy Lewis (TIBCO), Bob Lojek (Intalio), Henry Lowe (OMG), Brad Lund (Intel), Matthew MacKenzie (XMLGlobal Technologies), Murray Maloney (Commerce One), Richard Martin (Active Data Exchange), Highland Mary Mountain (Intel), Alex Milowski (Lexica), Kevin Mitchell (XMLSolutions), Ed Mooney (Sun Microsystems), Dean Moses (Epicentric), Don Mullen (TIBCO), Rekha Nagarajan (Calico Commerce), Raj Nair (Cisco), Mark Needleman (Data Research Associates), Art Nevarez (Novell), Henrik Nielsen (Microsoft Corporation), Kevin Perkins (Compaq), Jags Ramnaryan (BEA Systems), Vilhelm Rosenqvist (NCR), Marwan Sabbouh (MITRE), Waqar Sadiq (Vitria Technology Inc.), Rich Salz (Zolera), Krishna Sankar (Cisco), George Scott (Tradia), Shane Sesta (Active Data Exchange), Lew Shannon (NCR), John-Paul Sicotte (MessagingDirect Ltd.), Simeon Simeonov (Allaire), Simeon Simeonov (Macromedia), Aaron Skonnard (DevelopMentor), Nick Smilonich (Unisys), Soumitro Tagore (Informix Software), James Tauber (Bowstreet), Lynne Thompson (Unisys), Patrick Thompson (Rogue Wave), Jim Trezzo (Oracle), Asir Vedamuthu (WebMethods), Randy Waldrop (WebMethods), Fred Waskiewicz (OMG), David Webber (XMLGlobal Technologies), Ray Whitmer (Netscape), Stuart Williams (Hewlett-Packard), Yan Xu (DataChannel), Amr Yassin (Philips Research), Susan Yee (Active Data Exchange), Jin Yu (Martsoft).

We also wish to thank all the people who have contributed to discussions on xml-dist-app@w3.org.

SOAP Version 1.2 Part 1: Messaging Framework

W3C Recommendation 24 June 2003

This version:
http://www.w3.org/TR/2003/REC-soap12-part1-20030624/
Latest version:
http://www.w3.org/TR/soap12-part1/
Previous versions:
http://www.w3.org/TR/2003/PR-soap12-part1-20030507/
Editors:
Martin Gudgin, Microsoft
Marc Hadley, Sun Microsystems
Noah Mendelsohn, IBM
Jean-Jacques Moreau, Canon
Henrik Frystyk Nielsen, Microsoft

Please refer to the **errata** for this document, which may include some normative corrections.

The English version of this specification is the only normative version. Non-normative translations may also be available.

Abstract

SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. "Part 1: Messaging

Framework" defines, using XML technologies, an extensible messaging framework containing a message construct that can be exchanged over a variety of underlying protocols.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document is a Recommendation of the W3C. This document has been produced by the XML Protocol Working Group, which is part of the Web Services Activity. It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

Comments on this document are welcome. Please send them to the public mailing-list xmlp-comments@w3.org (archive). It is inappropriate to send discussion email to this address.

Information about implementations relevant to this specification can be found in the Implementation Report at http://www.w3.org/2000/xp/Group/2/03/soap1.2implementation.html.

Patent disclosures relevant to this specification may be found on the Working Group's patent disclosure page, in conformance with W3C policy.

A list of current W3C Recommendations and other technical reports can be found at http://www.w3.org/TR.

Short Table of Contents

Table of Contents

Appendices

---

1. Introduction

SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.

Two major design goals for SOAP are simplicity and extensibility (see XMLP Requirements [XMLP Requirements]). SOAP attempts to meet these goals by omitting, from the messaging framework, features that are often found in distributed systems. Such features include but are not limited to "reliability", "security", "correlation", "routing", and "Message Exchange Patterns" (MEPs). While it is anticipated that many features will be defined, this specification provides specifics only for two MEPs. Other features are left to be defined as extensions by other specifications.

The SOAP Version 1.2 specification consists of three parts. Part 1 of the SOAP Version 1.2 specification (this document) defines the SOAP messaging framework consisting of:

1. The SOAP processing model defining the rules for processing a SOAP message (see **2. SOAP Processing Model**).

2. The SOAP Extensibility model defining the concepts of SOAP features and SOAP modules (see **3. SOAP Extensibility Model**).

3. The SOAP underlying protocol binding framework describing the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP nodes (see **4. SOAP Protocol Binding Framework**).

4. The SOAP message construct defining the structure of a SOAP message (see **5. SOAP Message Construct**).

The SOAP 1.2 Primer [SOAP Part 0] is a non-normative document intended to provide an easily understandable tutorial on the features of the SOAP Version 1.2 specifications.

SOAP 1.2 Part 2 [SOAP Part 2] describes a set of adjuncts that can be used in connection with the SOAP messaging framework.

Note:

In previous versions of this specification the SOAP name was an acronym. This is no longer the case.

1.1 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC 2119].

This specification uses a number of namespace prefixes throughout; they are listed in **Table 1**. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see XML Infoset [XML InfoSet]).

Table 1: Prefixes and Namespaces used in this specification.

Prefix	Namespace	Notes
env	"http://www.w3.org/2003/05/soap-envelope"	A normative XML Schema [XML Schema Part 1], [XML Schema Part 2] document for the "http://www.w3.org/2003/05/soap-envelope" namespace can be found at http://www.w3.org/2003/05/soap-envelope.
xs	"http://www.w3.org/2001/XMLSchema"	The namespace of the XML Schema [XML Schema Part 1], [XML Schema Part 2] specification

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URIs (see RFC 2396 [RFC 2396]).

All parts of this specification are normative, with the exception of examples and sections explicitly marked as "Non-Normative".

1.2 Conformance

This specification describes data formats, and the rules for generating, exchanging, and processing messages using those formats. This specification does not mandate the scope of any particular implementation, although it requires that no implementation violates any mandatory requirement.

For an implementation to claim conformance with the SOAP Version 1.2 specification, it MUST correctly implement all mandatory ("MUST") requirements expressed in Part 1 of the SOAP Version 1.2 specification (this document) that pertain to the activity being performed. Note that an implementation is not mandated to implement all the mandatory requirements. For example, a special purpose implementation that never sends a SOAP header block can claim conformance provided that it correctly implements the mandatory requirements that pertain to the messages it does send.

An implementation MAY implement any number of the Adjuncts specified in SOAP 1.2 Part 2 [SOAP Part 2]. Note that no conformance is associated with the convention for describing features and bindings (see **3. SOAP Extensibility Model** and **4. SOAP Protocol Binding Framework**). The implementation of an Adjunct MUST implement all the pertinent mandatory requirements expressed in the specification of the Adjunct to claim conformance with the Adjunct.

SOAP Version 1.2 can be used as the basis for other technologies that provide richer or more specialized services. To claim conformance with the SOAP Version 1.2 specification, the specifications and implementations of such technologies must be consistent with the pertinent mandatory requirements expressed in Part 1 of the SOAP Version 1.2 specification (this document). Rules for conformance with such new specifications are beyond the scope of the SOAP Version 1.2 specification; it is recommended that specifications for such technologies provide the appropriate conformance rules.

SOAP Version 1.2 is designed to enable at least the usage scenarios described in SOAP 1.2 Usage Scenarios [SOAP Usage Scenarios], and possibly other scenarios. Informal descriptions showing XML representations of concrete SOAP messages used in some common scenarios are provided in SOAP 1.2 Part 0 [SOAP Part 0].

1.3 Relation to Other Specifications

A SOAP message is specified as an XML Information Set [XML InfoSet]. While all SOAP message examples in this document are shown using XML 1.0 [XML 1.0] syntax, other representations MAY be used to transmit SOAP messages between nodes (see **4. SOAP Protocol Binding Framework**).

Some of the *information items* defined by this document (see **5. SOAP Message Construct**) are identified using namespace qualified names [Namespaces in XML]. See **Table 1** for a list of the namespace names defined in this document.

> **Note:**
>
> This specification uses the term *XML Expanded Name* to refer to the value space pair {absolute uri reference,local-name} for a value of type xsd:QName. Similar terminology is under consideration for inclusion in future versions of Namespace in XML [Namespaces in XML]. Should future versions of namespace in XML [Namespaces in XML] adopt alternative terminology, we anticipate that corresponding changes will be made to this recommendation in the form of an erratum, or in conjunction with some other future revision.

SOAP does not require that XML Schema processing (assessment or validation) be performed to establish the correctness or 'schema implied' values of *element* and *attribute information items* defined by Parts 1 and 2 of this specification. The values associated with *element* and *attribute information items* defined in this specification MUST be carried explicitly in the transmitted SOAP message except where stated otherwise (see **5. SOAP Message Construct**).

SOAP *attribute information items* have types described by XML Schema [XML Schema Part 2]. Unless otherwise stated, all lexical forms are supported for each such attribute, and lexical forms representing the same value in the XML Schema value space are considered equivalent for purposes of SOAP processing, e.g. the boolean lexical forms "1" and "true" are interchangeable. For brevity, text in this specification refers only to one lexical form for each value, e.g. "if the value of the `mustUnderstand` *attribute information item* is 'true'".

Specifications for the processing of application-defined data carried in a SOAP message but not defined by this specification MAY call for additional validation of the SOAP message in conjunction with application-level processing. In such cases, the choice of schema language and/or validation technology is at the discretion of the application.

SOAP uses XML Base [XML Base] for determining a base URI for relative URI references used as values in *information items* defined by this specification (see **6. Use of URIs in SOAP**).

The media type "application/soap+xml" SHOULD be used for XML 1.0 serializations of the SOAP message infoset (see SOAP 1.2 Part 2 [SOAP Part 2], The "application/soap+xml" Media Type).

1.3.1 Processing Requirements

The ability to use SOAP in a particular environment will vary depending on the actual constraints, choice of tools, processing model, or nature of the messages being exchanged. SOAP has been designed to have a relatively small number of dependencies on other XML specifications, none of which are perceived as having prohibitive processing requirements. Also, limiting use of SOAP to small messages instead of arbitrarily-sized messages and supporting only a few specific message types instead of implementing generalized processing could significantly lower processing requirements.

1.4 Example SOAP Message

The following example shows a sample notification message expressed in SOAP. The message contains two pieces of application-defined data not defined by this specification: a SOAP header block with a local name of `alertcontrol` and a body element with a local name of `alert` . In general, SOAP header blocks contain information which might be of use to SOAP intermediaries as well as the ultimate destination of the message. In this example an intermediary might prioritize the delivery of the message based on the priority and expiration information in the SOAP header block. The body contains the actual message payload, in this case the alert message.

Example 1: SOAP message containing a SOAP header block and a SOAP body

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/
alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
```

```
    <m:alert xmlns:m="http://example.org/alert">
     <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
   </env:Body>
  </env:Envelope>
```

1.5 SOAP Terminology

This section describes the terms and concepts introduced in Part 1 of the SOAP Version 1.2 specification (this document).

1.5.1 Protocol Concepts

SOAP

The formal set of conventions governing the format and processing rules of a SOAP message. These conventions include the interactions among SOAP nodes generating and accepting SOAP messages for the purpose of exchanging information along a SOAP message path.

SOAP node

The embodiment of the processing logic necessary to transmit, receive, process and/or relay a SOAP message, according to the set of conventions defined by this recommendation. A SOAP node is responsible for enforcing the rules that govern the exchange of SOAP messages (see **2. SOAP Processing Model**). It accesses the services provided by the underlying protocols through one or more SOAP bindings.

SOAP role

A SOAP receiver's expected function in processing a message. A SOAP receiver can act in multiple roles.

SOAP binding

The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange (see **4. SOAP Protocol Binding Framework**). Examples of SOAP bindings include carrying a SOAP message within an HTTP entity-body, or over a TCP stream.

SOAP feature

An extension of the SOAP messaging framework (see **3. SOAP Extensibility Model**). Examples of features include "reliability", "security", "correlation", "routing", and "Message Exchange Patterns" (MEPs).

SOAP module

A SOAP Module is a specification that contains the combined syntax and semantics of SOAP header blocks specified according to the rules in **3.3 SOAP Modules**. A SOAP module realizes zero or more SOAP features.

SOAP message exchange pattern (MEP)

A template for the exchange of SOAP messages between SOAP nodes enabled by one or more underlying SOAP protocol bindings (see **4. SOAP Protocol Binding Framework**). A SOAP MEP is an example of a SOAP feature (see **3.2 SOAP Message Exchange Patterns (MEPs)**).

SOAP application

An entity, typically software, that produces, consumes or otherwise acts upon SOAP messages in a manner conforming to the SOAP processing model (see **2. SOAP Processing Model**).

1.5.2 Data Encapsulation Concepts

SOAP message

The basic unit of communication between SOAP nodes.

SOAP envelope

The outermost *element information item* of a SOAP message.

SOAP header

A collection of zero or more SOAP header blocks each of which might be targeted at any SOAP receiver within the SOAP message path.

SOAP header block

An *element information item* used to delimit data that logically constitutes a single computational unit within the SOAP header. The type of a SOAP header block is identified by the XML expanded name of the header block *element information item*.

SOAP body

A collection of zero or more *element information items* targeted at an ultimate SOAP receiver in the SOAP message path (see **5.3 SOAP Body**).

SOAP fault

A SOAP *element information item* which contains fault information generated by a SOAP node.

1.5.3 Message Sender and Receiver Concepts

SOAP sender

A SOAP node that transmits a SOAP message.

SOAP receiver

A SOAP node that accepts a SOAP message.

SOAP message path

The set of SOAP nodes through which a single SOAP message passes. This includes the initial SOAP sender, zero or more SOAP intermediaries, and an ultimate SOAP receiver.

Initial SOAP sender

The SOAP sender that originates a SOAP message at the starting point of a SOAP message path.

SOAP intermediary

A SOAP intermediary is both a SOAP receiver and a SOAP sender and is targetable from within a SOAP message. It processes the SOAP header blocks targeted at it and acts to forward a SOAP message towards an ultimate SOAP receiver.

Ultimate SOAP receiver

The SOAP receiver that is a final destination of a SOAP message. It is responsible for processing the contents of the SOAP body and any SOAP header blocks targeted at it. In some circumstances, a SOAP message might not reach an ultimate SOAP receiver, for example because of a problem at a SOAP intermediary. An ultimate SOAP receiver cannot also be a SOAP intermediary for the same SOAP message (see **2. SOAP Processing Model**).

2. SOAP Processing Model

SOAP provides a distributed processing model that assumes a SOAP message originates at an initial SOAP sender and is sent to an ultimate SOAP receiver via zero or more SOAP intermediaries. Note that the SOAP distributed processing model can support many MEPs including but not limited to one-way messages, request/response interactions, and peer-to-peer conversations (see **3.2 SOAP Message Exchange Patterns (MEPs)** for a description of the relationship between SOAP message exchange patterns and the SOAP extensibility model).

This section defines the SOAP distributed processing model. The SOAP processing model specifies how a SOAP receiver processes a SOAP message. It applies to a single message only, in isolation from any other SOAP message. The SOAP processing model itself does not maintain any state or perform any correlation or coordination between messages, even, for example, when used in combination with a SOAP feature which involves sending multiple SOAP messages in sequence, each subsequent message depending on the response to the previous message. It is the responsibility of each such features to define any combined processing.

Section **3. SOAP Extensibility Model** describes how SOAP can be extended and how SOAP extensions might interact with the SOAP processing model and the SOAP protocol binding framework. Section **4. SOAP Protocol Binding Framework** defines a framework for describing the rules for how SOAP messages can be exchanged over a variety of underlying protocols.

2.1 SOAP Nodes

A SOAP node can be the initial SOAP sender, an ultimate SOAP receiver, or a SOAP intermediary. A SOAP node receiving a SOAP message MUST perform processing according to the SOAP processing model as described in this section and in the remainder of this specification. A SOAP node is identified by a URI, see **5.4.3 SOAP Node Element**

2.2 SOAP Roles and SOAP Nodes

In processing a SOAP message, a SOAP node is said to act in one or more SOAP roles, each of which is identified by a URI known as the SOAP role name. The roles assumed by a node MUST be invariant during the processing of an individual SOAP message. This specification deals only with the processing of individual SOAP messages. No statement is made regarding the possibility that a given SOAP node might or might not act in varying roles when processing more than one SOAP message.

Table 2 defines three role names which have special significance in a SOAP message (see **2.6 Processing SOAP Messages**).

Table 2: SOAP Roles defined by this specification

Short-name	Name	Description
`next`	"http://www.w3.org/2003/05/soap-envelope/role/next"	Each SOAP intermediary and the ultimate SOAP receiver MUST act in this role.
`none`	"http://www.w3.org/2003/05/soap-envelope/role/none"	SOAP nodes MUST NOT act in this role.
`ultimateReceiver`	"http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver"	The ultimate receiver MUST act in this role.

In addition to the SOAP role names defined in **Table 2**, other role names MAY be used as necessary to meet the needs of SOAP applications.

While the purpose of a SOAP role name is to identify a SOAP node or nodes, there are no routing or message exchange semantics associated with the SOAP role name. For example, SOAP roles MAY be named with a URI useable to route SOAP messages to an appropriate SOAP node. Conversely, it is also appropriate to use SOAP roles with names that are related more indirectly to message routing (e.g. "http://example.org/banking/anyAccountMgr") or which are unrelated to routing (e.g. a URI meant to identify "all cache management software". For example, a SOAP header block targeted at such a role might be used to carry an indication to any concerned software that the containing SOAP message is idempotent, and can safely be cached and replayed).

With the exception of the three SOAP role names defined in **Table 2**, this

specification does not prescribe the criteria by which a given node determines the set of roles in which it acts on a given message. For example, implementations can base this determination on factors including, but not limited to: hard coded choices in the implementation, information provided by the underlying protocol binding (e.g. the URI to which the message was physically delivered), or configuration information provided by users during system installation.

2.3 Targeting SOAP Header Blocks

A SOAP header block MAY carry a `role` *attribute information item* (see **5.2.2 SOAP role Attribute**) that is used to target the header block at SOAP nodes operating in the specified role. This specification refers to the value of the SOAP `role` *attribute information item* as the SOAP role for the corresponding SOAP header block.

A SOAP header block is said to be targeted at a SOAP node if the SOAP role for the header block is the name of a role in which the SOAP node operates. SOAP header blocks targeted at the special role "http://www.w3.org/2003/05/soap-envelope/role/none" are never formally processed. Such SOAP header blocks MAY carry data that is required for processing of other SOAP header blocks. Unless removed by the action of an intermediary (see **2.7 Relaying SOAP Messages**) , such blocks are relayed with the message to the ultimate receiver (see also **3.3 SOAP Modules**).

2.4 Understanding SOAP Header Blocks

It is likely that specifications for a wide variety of header functions (i.e. SOAP modules) will be developed over time (see **3.3 SOAP Modules**), and that some SOAP nodes might include the software necessary to implement one or more such extensions. A SOAP header block is said to be understood by a SOAP node if the software at that SOAP node has been written to fully conform to and implement the semantics specified for the XML expanded name of the outer-most *element information item* of that header block.

A SOAP header block MAY carry a `mustUnderstand` *attribute information item* (see **5.2.3 SOAP mustUnderstand Attribute**). When the value of such an *attribute information item* is "true", the SOAP header block is said to be mandatory.

Mandatory SOAP header blocks are presumed to somehow modify the semantics of other SOAP header blocks or SOAP body elements. Therefore, for every mandatory SOAP header block targeted to a node, that node MUST either process the header block or not process the SOAP message at all, and

instead generate a fault (see **2.6 Processing SOAP Messages** and **5.4 SOAP Fault**). Tagging SOAP header blocks as mandatory thus assures that such modifications will not be silently (and, presumably, erroneously) ignored by a SOAP node to which the header block is targeted.

The `mustUnderstand` *attribute information item* is not intended as a mechanism for detecting errors in routing, misidentification of nodes, failure of a node to serve in its intended role(s), etc. Any of these conditions can result in a failure to even attempt processing of a given SOAP header block from a SOAP envelope. This specification therefore does not require any fault to be generated based on the presence or value of the `mustUnderstand` *attribute information item* on a SOAP header block not targeted at the current processing node. In particular, it is not an error for an ultimate SOAP receiver to receive a message containing a mandatory SOAP header block that is targeted at a role other than the ones assumed by the ultimate SOAP receiver. This is the case, for example, when a SOAP header block has survived erroneously due to a routing or targeting error at a preceding intermediary.

2.5 Structure and Interpretation of SOAP Bodies

An ultimate SOAP receiver MUST correctly process the immediate children of the SOAP body (see **5.3 SOAP Body**). However, with the exception of SOAP faults (see **5.4 SOAP Fault**), Part 1 of this specification (this document) mandates no particular structure or interpretation of these elements, and provides no standard means for specifying the processing to be done.

2.6 Processing SOAP Messages

This section sets out the rules by which SOAP messages are processed. Nothing in this specification prevents the use of optimistic concurrency, roll back, or other techniques that might provide increased flexibility in processing order. Unless otherwise stated, processing of all generated SOAP messages, SOAP faults and application-level side effects MUST be semantically equivalent to performing the following steps separately, and in the order given.

1. Determine the set of roles in which the node is to act. The contents of the SOAP envelope, including any SOAP header blocks and the SOAP body, MAY be inspected in making such determination.

2. Identify all header blocks targeted at the node that are mandatory.

3. If one or more of the SOAP header blocks identified in the preceding step are not understood by the node then generate a single SOAP fault

with the `Value` of `Code` set to "env:MustUnderstand" (see **5.4.8 SOAP mustUnderstand Faults**). If such a fault is generated, any further processing MUST NOT be done. Faults relating to the contents of the SOAP body MUST NOT be generated in this step.

Note:

Throughout this document, the term "`Value` of `Code` " is used as a shorthand for "value of the `Value` child *element information item* of the `Code` *element information item*" (see **5.4.1 SOAP Code Element**).

4. Process all mandatory SOAP header blocks targeted at the node and, in the case of an ultimate SOAP receiver, the SOAP body. A SOAP node MAY also choose to process non-mandatory SOAP header blocks targeted at it.

5. In the case of a SOAP intermediary, and where the SOAP message exchange pattern and results of processing (e.g. no fault generated) require that the SOAP message be sent further along the SOAP message path, relay the message as described in section **2.7 Relaying SOAP Messages**.

In all cases where a SOAP header block is processed, the SOAP node MUST understand the SOAP header block and MUST do such processing in a manner fully conformant with the specification for that header block. The successful processing of one header block does not guarantee successful processing of another block with the same XML expanded name within the same message: the specification for the header block determines the circumstances in which such processing would result in a fault. An ultimate SOAP receiver MUST process the SOAP body, in a manner consistent with **2.5 Structure and Interpretation of SOAP Bodies**.

Failure is indicated by the generation of a fault (see **5.4 SOAP Fault**). SOAP message processing MAY result in the generation of at most one fault.

A message may contain or result in multiple errors during processing. Except where the order of detection is specifically indicated (as in **2.4 Understanding SOAP Header Blocks**), a SOAP node is at liberty to reflect any single fault from the set of possible faults prescribed for the errors encountered. The selection of a fault need not be predicated on the application of the "MUST", "SHOULD" or "MAY" keywords to the generation of the fault, with the exception that if one or more of the prescribed faults is qualified with the "MUST" keyword, then any one fault from the set of possible

faults MUST be generated.

SOAP nodes MAY make reference to any information in the SOAP envelope when processing a SOAP body or SOAP header block. For example, a caching function can cache the entire SOAP message, if desired.

The processing of one or more SOAP header blocks MAY control or determine the order of processing for other SOAP header blocks and/or the SOAP body. For example, one could create a SOAP header block to force processing of other SOAP header blocks in lexical order. In the absence of such a controlling SOAP header block, the order of header and body processing is at the discretion of the SOAP node. Header blocks MAY be processed in arbitrary order. Header block processing MAY precede, MAY be interleaved with, or MAY follow processing of the SOAP body. For example, processing of a "begin transaction" header block would typically precede body processing, a "logging" function might run concurrently with body processing and a "commit transaction" header block might be honored following completion of all other work.

Note:

The above rules apply to processing at a single node. SOAP extensions can be designed to ensure that SOAP header blocks are processed in an appropriate order, as the message moves along the message path towards the ultimate SOAP receiver. Specifically, such extensions might specify that a fault with a `Value` of `Code` set to "env:Sender" is generated if some SOAP header blocks have inadvertently survived past some intended point in the message path. Such extensions might depend on the presence or value of the `mustUnderstand` *attribute information item* in the surviving SOAP header blocks when determining whether an error has occurred.

2.7 Relaying SOAP Messages

As mentioned earlier in this section, it is assumed that a SOAP message originates at an initial SOAP sender and is sent to an ultimate SOAP receiver via zero or more SOAP intermediaries. While SOAP does not itself define any routing or forwarding semantics, it is anticipated that such functionality can be described as one or more SOAP features (see **3. SOAP Extensibility Model**). The purpose of this section is to describe how message forwarding interacts with the SOAP distributed processing model.

SOAP defines two different types of intermediaries: forwarding intermediaries and active intermediaries. These two types of intermediary are described in this section.

2.7.1 Relaying SOAP Header Blocks

The relaying of SOAP header blocks targeted at an intermediary SOAP node depends on whether the SOAP header blocks are processed or not by that node. A SOAP header block is said to be reinserted if the processing of that header block determines that the header block is to be reinserted in the forwarded message. The specification for a SOAP header block may call for the header block to be relayed in the forwarded message if the header block is targeted at a role played by the SOAP intermediary, but not otherwise processed by the intermediary. Such header blocks are said to be relayable.

A SOAP header block MAY carry a `relay` *attribute information item* (see **5.2.4 SOAP relay Attribute**). When the value of such an *attribute information item* is "true", the header block is said to be relayable. The forwarding of relayable header blocks is described in section **2.7.2 SOAP Forwarding Intermediaries**.

The `relay` *attribute information item* has no effect on SOAP header blocks targeted at a role other than one assumed by a SOAP intermediary.

The `relay` *attribute information item* has no effect on the SOAP processing model when the header block also carries a `mustUnderstand` *attribute information item* with a value of "true".

The `relay` *attribute information item* has no effect on the processing of SOAP messages by the SOAP ultimate receiver.

Table 3 summarizes the forwarding behavior of a SOAP node.

Table 3: SOAP Nodes Forwarding behavior

Role		Header block	
Short-name	**Assumed**	**Understood & Processed**	**Forwarded**
`next`	Yes	Yes	No, unless reinserted
		No	No, unless `relay` ="true"
user-defined	Yes	Yes	No, unless reinserted
		No	No, unless `relay` ="true"

	No	n/a	Yes
`ultimateReceiver`	Yes	Yes	n/a
		No	n/a
`none`	No	n/a	Yes

2.7.2 SOAP Forwarding Intermediaries

The semantics of one or more SOAP header blocks in a SOAP message, or the SOAP MEP used MAY require that the SOAP message be forwarded to another SOAP node on behalf of the initiator of the inbound SOAP message. In this case, the processing SOAP node acts in the role of a SOAP forwarding intermediary.

Forwarding SOAP intermediaries MUST process the message according to the SOAP processing model defined in **2.6 Processing SOAP Messages**. In addition, when generating a SOAP message for the purpose of forwarding, they MUST:

1. Remove all processed SOAP header blocks.
2. Remove all non-relayable SOAP header blocks that were targeted at the forwarding node but ignored during processing.
3. Retain all relayable SOAP header blocks that were targeted at the forwarding node but ignored during processing.

Forwarding SOAP intermediaries MUST also obey the specification for the SOAP forwarding features being used. The specification for each such a feature MUST describe the required semantics, including the rules describing how the forwarded message is constructed. Such rules MAY describe placement of inserted or reinserted SOAP header blocks. Inserted SOAP header blocks might be indistinguishable from one or more of the header blocks removed by the intermediary. Processing is defined here in terms of re-inserting header blocks (rather than leaving them in place) to emphasize the need to process them at each SOAP node along the SOAP message path.

2.7.2.1 Relayed Infoset

This section describes the behavior of SOAP forwarding intermediaries with respect to preservation of the XML infoset properties of a relayed SOAP message.

Unless overridden by the processing of SOAP features at an intermediary (see **2.7.2 SOAP Forwarding Intermediaries**), the following rules apply:

1. All XML infoset properties of a message MUST be preserved, except as specified in rules 2 through 22.
2. The *element information item* for a header block targeted at an intermediary MAY be removed, by that intermediary, from the [children] property of the SOAP `Header` *element information item* as detailed in **2.7.2 SOAP Forwarding Intermediaries**.
3. *Element information items* for additional header blocks MAY be added to the [children] property of the SOAP `Header` *element information item* as detailed in **2.7.2 SOAP Forwarding Intermediaries**.

   In this case, a SOAP `Header` *element information item* MAY be added, as the first member of the [children] property of the SOAP `Envelope` *element information item*, if it is NOT already present.
4. *White space character information items* MAY be removed from the [children] property of the SOAP `Envelope` *element information item*.
5. *White space character information items* MAY be added to the [children] property of the SOAP `Envelope` *element information item*.
6. *White space character information items* MAY be removed from the [children] property of the SOAP `Header` *element information item*.
7. *White space character information items* MAY be added to the [children] property of the SOAP `Header` *element information item*.
8. *Comment information items* MAY be added to the [children] property of the SOAP `Envelope` *element information item*.
9. *Comment information items* MAY be removed from the [children] property of the SOAP `Envelope` *element information item*.
10. *Comment information items* MAY be added to the [children] property of the SOAP `Header` *element information item*.
11. *Comment information items* MAY be removed from the [children] property of the SOAP `Header` *element information item*.
12. *Attribute information items* MAY be added to the [attributes] property of the SOAP `Envelope` *element information item*.
13. *Attribute information items* MAY be added to the [attributes] property of the SOAP `Header` *element information item*.
14. *Attribute information items* MAY be added to the [namespace attributes] property of the SOAP `Envelope` *element information item*.
15. *Attribute information items* MAY be added to the [namespace attributes] property of the SOAP `Header` *element information item*.
16. SOAP `role` *attribute information items* that are present in the [attributes] property of SOAP header block *element information items* may be transformed as described in **5.2.2 SOAP role Attribute**.
17. SOAP `mustUnderstand` *attribute information items* that are present in the [attributes] property of SOAP header block *element information items* may be transformed as described in **5.2.3 SOAP mustUnderstand Attribute**.
18. SOAP `relay` *attribute information items* that are present in the

[attributes] property of SOAP header block *element information items* may be transformed as described in **5.2.4 SOAP relay Attribute**.

19. The [base URI] property of the *document information item* need not be maintained.
20. The [base URI] property of *element information items* MAY be changed or removed.
21. The [character encoding scheme] property of the *document information item* MAY be changed or removed.
22. All *namespace information items* in the [in-scope namespaces] of *element information items* MUST be preserved. Additional *namespace information items* MAY be added.

Note:

The rules above allow for signing of SOAP header blocks, the SOAP body, and combinations of SOAP header blocks and the SOAP body.

In the absence of a canonicalization algorithm to normalize the infoset transformations and if the "http://www.w3.org/TR/2001/REC-xml-c14n-20010315" canonicalization algorithm is used then items 1-6 and 11-14 are incompatible with signing the SOAP envelope and items 1, 2, 5, 6, 12 and 14 are incompatible with signing the SOAP header.

Similarly, if the "http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments" canonicalization algorithm is used then items 7 and 8 are incompatible with signing the SOAP envelope and items 9 and 10 are incompatible with signing the SOAP header.

Note:

White space character information items are those whose [character code] property has a value of #x20, #x9, #xD or #xA.

2.7.3 SOAP Active Intermediaries

In addition to the processing performed by forwarding SOAP intermediaries, active SOAP intermediaries undertake additional processing that can modify the outbound SOAP message in ways *not* described in the inbound SOAP message. That is, they can undertake processing not described by SOAP header blocks in the incoming SOAP message. The potential set of services provided by an active SOAP intermediary includes, but is not limited to: security services, annotation services, and content manipulation services.

The results of such active processing could impact the interpretation of SOAP messages by downstream SOAP nodes. For example, as part of generating

an outbound SOAP message, an active SOAP intermediary might have removed and encrypted some or all of the SOAP header blocks found in the inbound SOAP message. It is strongly recommended that SOAP features provided by active SOAP intermediaries be described in a manner that allows such modifications to be detected by affected SOAP nodes in the message path.

2.8 SOAP Versioning Model

The version of a SOAP message is identified by the XML expanded name of the child *element information item* of the *document information item*. A SOAP Version 1.2 message has a child *element information item* of the *document information item* with a [local name] of `Envelope` and a [namespace name] of "http://www.w3.org/2003/05/soap-envelope" (see **5.1 SOAP Envelope**).

A SOAP node determines whether it supports the version of a SOAP message on a per message basis. In this context "support" means understanding the semantics of that version of a SOAP envelope. The versioning model is directed only at the SOAP `Envelope` *element information item*. It does not address versioning of SOAP header blocks, encodings, protocol bindings, or anything else.

A SOAP node MAY support multiple envelope versions. However, when processing a message, a SOAP node MUST use the semantics defined by the version of that message.

If a SOAP node receives a message whose version is not supported it MUST generate a fault (see **5.4 SOAP Fault**) with a `Value` of `Code` set to "env:VersionMismatch". Any other malformation of the message construct MUST result in the generation of a fault with a `Value` of `Code` set to "env:Sender".

Appendix **A. Version Transition From SOAP/1.1 to SOAP Version 1.2** defines a mechanism for transitioning from SOAP/1.1 to SOAP Version 1.2 using the `Upgrade` *element information item* (see **5.4.7 VersionMismatch Faults**).

3. SOAP Extensibility Model

SOAP provides a simple messaging framework whose core functionality is concerned with providing extensibility. The extensibility mechanisms described below can be used to add capabilities found in richer messaging environments.

3.1 SOAP Features

A SOAP feature is an extension of the SOAP messaging framework. Although SOAP poses no constraints on the potential scope of such features, example features may include "reliability", "security", "correlation", "routing", and message exchange patterns (MEPs) such as request/response, one-way, and peer-to-peer conversations.

The SOAP extensibility model provides two mechanisms through which features can be expressed: the SOAP Processing Model and the SOAP Protocol Binding Framework (see **2. SOAP Processing Model** and **4. SOAP Protocol Binding Framework**). The former describes the behavior of a single SOAP node with respect to the processing of an individual message. The latter mediates the act of sending and receiving SOAP messages by a SOAP node via an underlying protocol.

The SOAP Processing Model enables SOAP nodes that include the mechanisms necessary to implement one or more features to express such features within the SOAP envelope as SOAP header blocks (see **2.4 Understanding SOAP Header Blocks**). Such header blocks can be intended for any SOAP node or nodes along a SOAP message path (see **2.3 Targeting SOAP Header Blocks**). The combined syntax and semantics of SOAP header blocks are known as a SOAP module, and are specified according to the rules in **3.3 SOAP Modules**.

In contrast, a SOAP protocol binding operates between two adjacent SOAP nodes along a SOAP message path. There is no requirement that the same underlying protocol is used for all hops along a SOAP message path. In some cases, underlying protocols are equipped, either directly or through extension, with mechanisms for providing certain features. The SOAP Protocol Binding Framework provides a scheme for describing these features and how they relate to SOAP nodes through a binding specification (see **4. SOAP Protocol Binding Framework**).

Certain features might require end-to-end as opposed to hop-by-hop processing semantics. Although the SOAP Protocol Binding Framework allows end-to-end features to be expressed outside the SOAP envelope, no standard mechanism is provided for the processing by intermediaries of the resulting messages. A binding specification that expresses such features external to the SOAP envelope needs to define its own processing rules for those externally expressed features. A SOAP node is expected to conform to these processing rules (for example, describing what information is passed along with the SOAP message as it leaves the intermediary). The processing of SOAP envelopes in accordance with the SOAP Processing Model (see **2. SOAP Processing Model**) MUST NOT be overridden by binding

specifications.

It is recommended that, where practical, end-to-end features be expressed as SOAP header blocks, so that the rules defined by the SOAP Processing Model can be employed.

3.1.1 Requirements on Features

The specification of a feature MUST include the following:

1. A URI used to name the feature. This enables the feature to be unambiguously referenced in description languages or during negotiation.
2. The information (state) required at each node to implement the feature.

3. The processing required at each node in order to fulfill the obligations of the feature including any handling of communication failures that might occur in the underlying protocol (see also **4.2 Binding Framework**).

4. The information to be transmitted from node to node.

See **3.2 SOAP Message Exchange Patterns (MEPs)** for additional requirements on MEP features.

3.2 SOAP Message Exchange Patterns (MEPs)

A Message Exchange Pattern (MEP) is a template that establishes a pattern for the exchange of messages between SOAP nodes. MEPs are a type of feature, and unless otherwise stated, references in this specification to the term "feature" apply also to MEPs. The request-response MEP specified in SOAP 1.2 Part 2 [SOAP Part 2] illustrates the specification of a MEP feature.

The specification of a message exchange pattern MUST:

- As mandated by **3.1.1 Requirements on Features**, provide a URI to name the MEP.
- Describe the life cycle of a message exchange conforming to the pattern.
- Describe the temporal/causal relationships, if any, of multiple messages exchanged in conformance with the pattern (e.g. responses follow requests and are sent to the originator of the request.)
- Describe the normal and abnormal termination of a message exchange conforming to the pattern.

Underlying protocol binding specifications can declare their support for one or more named MEPs.

MEPs are SOAP features, so an MEP specification MUST conform to the requirements for SOAP feature specifications (see **3.1.1 Requirements on Features**). An MEP specification MUST also include:

1. Any requirements to generate additional messages (such as responses to requests in a request/response MEP).
2. Rules for the delivery or other disposition of SOAP faults generated during the operation of the MEP.

3.3 SOAP Modules

The term "SOAP module" refers to the specification of the syntax and semantics of one or more SOAP header blocks. A SOAP module realizes zero or more SOAP features. A module specification adheres to the following rules. It:

1. MUST identify itself with a URI. This enables the module to be unambiguously referenced in description languages or during negotiation.
2. MUST declare the features provided by a module (see **3.1 SOAP Features**).
3. MUST clearly and completely specify the content and semantics of the SOAP header blocks used to implement the behavior in question, including if appropriate any modifications to the SOAP processing model. The SOAP extensibility model does not limit the extent to which SOAP can be extended. Nor does it prevent extensions from modifying the SOAP processing model from that described in **2. SOAP Processing Model**
4. MAY utilize the property conventions defined in SOAP 1.2 Part 2 [SOAP Part 2], section A Convention for Describing Features and Bindings, in describing the functionality that the module provides. If these conventions are followed, the module specification MUST clearly describe the relationship between the abstract properties and their representations in the SOAP envelope. Note that it is possible to write a feature specification purely in terms of abstract properties, and then write a separate module specification which implements that feature, mapping the properties defined in the feature specification to SOAP header blocks in the SOAP module.
5. MUST clearly specify any known interactions with or changes to the interpretation of the SOAP body. Furthermore, it MUST clearly specify

any known interactions with or changes to the interpretation of other SOAP features and SOAP modules. For example, we can imagine a module which encrypts and removes the SOAP body, inserting instead a SOAP header block containing a checksum and an indication of the encryption mechanism used. The specification for such a module would indicate that the decryption algorithm on the receiving side is to be run *prior* to any other modules which rely on the contents of the SOAP body.

4. SOAP Protocol Binding Framework

SOAP enables exchange of SOAP messages using a variety of underlying protocols. The formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange is called a binding. The SOAP Protocol Binding Framework provides general rules for the specification of protocol bindings; the framework also describes the relationship between bindings and SOAP nodes that implement those bindings. The HTTP binding in SOAP 1.2 Part 2 [SOAP Part 2] illustrates the specification of a binding. Additional bindings can be created by specifications that conform to the binding framework introduced in this chapter.

A SOAP binding specification:

- Declares the features provided by a binding.
- Describes how the services of the underlying protocol are used to transmit SOAP message infosets.
- Describes how the services of the underlying protocol are used to honor the contract formed by the features supported by that binding.
- Describes the handling of all potential failures that can be anticipated within the binding.
- Defines the requirements for building a conformant implementation of the binding being specified.

A binding does not provide a separate processing model and does not constitute a SOAP node by itself. Rather a SOAP binding is an integral part of a SOAP node (see **2. SOAP Processing Model**).

4.1 Goals of the Binding Framework

The goals of the binding framework are:

1. To set out the requirements and concepts that are common to all binding specifications.
2. To facilitate homogeneous description in situations where multiple

bindings support common features, promoting reuse across bindings.
3. To facilitate consistency in the specification of optional features.

Two or more bindings can offer a given optional feature, such as reliable delivery, using different means. One binding might exploit an underlying protocol that directly facilitates the feature (e.g., the protocol is reliable), and the other binding might provide the necessary logic itself (e.g., reliability is achieved via logging and retransmission). In such cases, the feature can be made available to applications in a consistent manner, regardless of which binding is used.

4.2 Binding Framework

The creation, transmission, and processing of a SOAP message, possibly through one or more intermediaries, is specified in terms of a distributed state machine. The state consists of information known to a SOAP node at a given point in time, including but not limited to the contents of messages being assembled for transmission or received for processing. The state at each node can be updated either by local processing, or by information received from an adjacent node.

Section **2. SOAP Processing Model** of this specification describes the processing that is common to all SOAP nodes when receiving a message. The purpose of a binding specification is to augment those core SOAP rules with additional processing that is particular to the binding, and to specify the manner in which the underlying protocol is used to transmit information between adjacent nodes in the message path.

The distributed state machine that manages the transmission of a given SOAP message through its message path is the combination of the core SOAP processing (see **2. SOAP Processing Model**) operating at each node, in conjunction with the binding specifications connecting each pair of nodes. A binding specification MUST enable one or more MEPs.

In cases where multiple features are supported by a binding specification, the specifications for those features MUST provide any information necessary for their successful use in combination. Similarly, any dependencies of one feature on another (i.e. if successful use of one feature depends on use or non-use of another) MUST be specified. This binding framework does not provide any explicit mechanism for controlling the use of such interdependent features.

The binding framework provides no fixed means of naming or typing the information comprising the state at a given node. Individual feature and binding specifications are free to adopt their own conventions for specifying

state. Note, however, that consistency across bindings and features is likely to be enhanced in situations where multiple feature specifications adopt consistent conventions for representing state. For example, multiple features might benefit from a consistent specification for an authentication credential, a transaction ID, etc. The HTTP binding in SOAP 1.2 Part 2 [SOAP Part 2] illustrates one such convention.

As described in **5. SOAP Message Construct**, each SOAP message is specified as an XML infoset that consists of a *document information item* with exactly one child: the SOAP `Envelope` *element information item*. Therefore, the minimum responsibility of a binding in transmitting a message is to specify the means by which the SOAP message infoset is transferred to and reconstituted by the binding at the receiving SOAP node and to specify the manner in which the transmission of the envelope is effected using the facilities of the underlying protocol.

The binding framework does not require that every binding use the XML 1.0 [XML 1.0] serialization as the "on the wire" representation of the XML infoset; compressed, encrypted, fragmented representations and so on can be used if appropriate. A binding, if using XML 1.0 serialization of the XML infoset, MAY mandate that a particular character encoding or set of encodings be used.

Bindings MAY provide for streaming when processing messages. That is, SOAP nodes MAY begin processing a received SOAP message as soon as the necessary information is available. SOAP processing is specified in terms of SOAP message infosets (see **5. SOAP Message Construct**). Although streaming SOAP receivers will acquire such XML infosets incrementally, SOAP processing MUST yield results identical to those that would have been achieved if the entire SOAP envelope were available prior to the start of processing. For example, as provided in **2.6 Processing SOAP Messages**, identification of targeted SOAP header blocks, and checking of all `mustUnderstand` attributes is to be done before successful processing can proceed. Depending on the representation used for the XML infoset, and the order in which it is transmitted, this rule might limit the degree to which streaming can be achieved.

Bindings MAY depend on state that is modeled as being outside of the SOAP message infoset (e.g. retry counts), and MAY transmit such information to adjacent nodes. For example, some bindings take a message delivery address (typically a URI) that is not within the envelope.

5. SOAP Message Construct

A SOAP message is specified as an XML infoset that consists of a *document*

information item with exactly one member in its [children] property, which MUST be the SOAP `Envelope` *element information item* (see **5.1 SOAP Envelope**). This *element information item* is also the value of the [document element] property. The [notations] and [unparsed entities] properties are both empty. The [base URI], [character encoding scheme] and [version] properties can have any legal value. The [standalone] property either has a value of "yes" or has no value.

The XML infoset of a SOAP message MUST NOT contain a *document type declaration information item*.

SOAP messages sent by initial SOAP senders MUST NOT contain *processing instruction information items*. SOAP intermediaries MUST NOT insert *processing instruction information items* in SOAP messages they relay. SOAP receivers receiving a SOAP message containing a *processing instruction information item* SHOULD generate a SOAP fault with the `Value` of `Code` set to "env:Sender". However, in the case where performance considerations make it impractical for an intermediary to detect *processing instruction information items* in a message to be relayed, the intermediary MAY leave such *processing instruction information items* unchanged in the relayed message.

Element information items defined by this specification that only have *element information items* defined as allowable members of their [children] property can also have zero or more *character information item* children whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0]. Unless otherwise indicated, such *character information items* are considered insignificant.

Comment information items MAY appear as children and/or descendants of the [document element] *element information item* but not before or after that *element information item* . There are some restrictions in the processing model with respect to when *comment information items* can be added and/or removed (see **2.7.2.1 Relayed Infoset**).

5.1 SOAP Envelope

The SOAP `Envelope` *element information item* has:

- A [local name] of `Envelope` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- Zero or more namespace qualified *attribute information items* amongst its [attributes] property.
- One or two *element information items* in its [children] property in order

as follows:

1. An optional `Header` *element information item* (see **5.2 SOAP Header**).
2. A mandatory `Body` *element information item* (see **5.3 SOAP Body**).

5.1.1 SOAP encodingStyle Attribute

The `encodingStyle` *attribute information item* indicates the encoding rules used to serialize parts of a SOAP message.

The `encodingStyle` *attribute information item* has:

- A [local name] of `encodingStyle` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".

The `encodingStyle` *attribute information item* is of type *xs:anyURI*. Its value identifies a set of serialization rules that can be used to deserialize the SOAP message.

The `encodingStyle` *attribute information item* MAY appear on the following:

1. A SOAP header block (see **5.2.1 SOAP header block**).
2. A child *element information item* of the SOAP `Body` *element information item* (see **5.3.1 SOAP Body child Element**) if that child is not a SOAP Fault *element information item* (see **5.4 SOAP Fault**).
3. A child *element information item* of the SOAP `Detail` *element information item* (see **5.4.5.1 SOAP detail entry**).
4. Any descendent of 1, 2, and 3 above.

The `encodingStyle` *attribute information item* MUST NOT appear on any element other than above in a SOAP message infoset.

The scope of the `encodingStyle` *attribute information item* is that of its owner *element information item* and that *element information item's* descendants, unless a descendant itself carries such an *attribute information item*. If no `encodingStyle` *attribute information item* is in scope for a particular *element information item* or the value of such an *attribute information item* is "http://www.w3.org/2003/05/soap-envelope/encoding/none" then no claims are made regarding the encoding style of that *element information item* and its descendants.

Example 2: Values for the `encodingStyle` *attribute information item*.

```
"http://www.w3.org/2003/05/soap-encoding"
"http://example.org/encoding/"
"http://www.w3.org/2003/05/soap-envelope/encoding/
none"
```

5.2 SOAP Header

The SOAP `Header` *element information item* provides a mechanism for extending a SOAP message in a decentralized and modular way (see **3. SOAP Extensibility Model** and **2.4 Understanding SOAP Header Blocks**).

The `Header` *element information item* has:

- A [local name] of `Header` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- Zero or more namespace qualified *attribute information items* in its [attributes] property.
- Zero or more namespace qualified *element information item*s in its [children] property.

Each child *element information item* of the SOAP Header is called a SOAP header block.

5.2.1 SOAP header block

Each SOAP header block *element information item*:

- MUST have a [namespace name] property which has a value, that is the name of the element MUST be namespace qualified.
- MAY have any number of *character information item* children. Child *character information items* whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0] are considered significant.
- MAY have any number of *element information item* children. Such *element information items* MAY be namespace qualified.
- MAY have zero or more *attribute information items* in its [attributes] property. Among these MAY be any or all of the following, which have special significance for SOAP processing:
    - `encodingStyle` *attribute information item* (see **5.1.1 SOAP encodingStyle Attribute**).
    - `role` *attribute information item* (see **5.2.2 SOAP role Attribute**).

- `mustUnderstand` *attribute information item* (see **5.2.3 SOAP mustUnderstand Attribute**).
- `relay` *attribute information item* (see **5.2.4 SOAP relay Attribute**).

Example 3: SOAP Header with a single SOAP header block

```
<env:Header xmlns:env="http://www.w3.org/2003/05/soap-
envelope" >
 <t:Transaction xmlns:t="http://example.org/2001/06/
tx"
               env:mustUnderstand="true" >
   5
 </t:Transaction>
</env:Header>
```

5.2.2 SOAP role Attribute

A SOAP role is used to indicate the SOAP node to which a particular SOAP header block is targeted (see **2.2 SOAP Roles and SOAP Nodes**).

The `role` *attribute information item* has the following XML infoset properties:

- A [local name] of `role` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- A [specified] property with a value of "true".

The type of the `role` *attribute information item* is *xs:anyURI*. The value of the `role` *attribute information item* is a URI that names a role that a SOAP node can assume.

Omitting the SOAP `role` *attribute information item* is equivalent to supplying that attribute with a value of "http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver".

SOAP senders SHOULD NOT generate, but SOAP receivers MUST accept the SOAP `role` *attribute information item* with a value of "http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver".

If relaying the message, a SOAP intermediary MAY omit a SOAP `role` *attribute information item* if its value is "http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver" (see **2.7 Relaying SOAP Messages**).

A SOAP sender generating a SOAP message SHOULD use the `role`

attribute information item only on SOAP header blocks. A SOAP receiver MUST ignore this *attribute information item* if it appears on descendants of a SOAP header block or on a SOAP body child *element information item* (or its descendents).

5.2.3 SOAP mustUnderstand Attribute

The SOAP `mustUnderstand` *attribute information item* is used to indicate whether the processing of a SOAP header block is mandatory or optional (see **2.4 Understanding SOAP Header Blocks**)

The `mustUnderstand` *attribute information item* has the following XML infoset properties:

- A [local name] of `mustUnderstand` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- A [specified] property with a value of "true".

The type of the `mustUnderstand` *attribute information item* is *xs:boolean*.

Omitting this *attribute information item* is defined as being semantically equivalent to including it with a value of "false".

SOAP senders SHOULD NOT generate, but SOAP receivers MUST accept the SOAP `mustUnderstand` *attribute information item* with a value of "false" or "0".

If generating a SOAP `mustUnderstand` *attribute information item*, a SOAP sender SHOULD use the canonical representation "true" of the attribute value (see XML Schema [XML Schema Part 2]). A SOAP receiver MUST accept any valid lexical representation of the attribute value.

If relaying the message, a SOAP intermediary MAY substitute "true" for the value "1", or "false" for "0". In addition, a SOAP intermediary MAY omit a SOAP `mustUnderstand` *attribute information item* if its value is "false" (see **2.7 Relaying SOAP Messages**).

A SOAP sender generating a SOAP message SHOULD use the `mustUnderstand` *attribute information item* only on SOAP header blocks. A SOAP receiver MUST ignore this *attribute information item* if it appears on descendants of a SOAP header block or on a SOAP body child *element information item* (or its descendents).

5.2.4 SOAP relay Attribute

The SOAP `relay` *attribute information item* is used to indicate whether a SOAP header block targeted at a SOAP receiver must be relayed if not processed (see **2.7.1 Relaying SOAP Header Blocks**).

The `relay` *attribute information item* has the following XML infoset properties:

- A [local name] of `relay` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- A [specified] property with a value of "true".

The type of the relay *attribute information item* is *xs:boolean*.

Omitting this *attribute information item* is defined as being semantically equivalent to including it with a value of "false".

SOAP senders SHOULD NOT generate, but SOAP receivers MUST accept the SOAP `relay` *attribute information item* with a value of "false" or "0".

If generating a SOAP `relay` *attribute information item*, a SOAP sender SHOULD use the canonical representation "true" of the attribute value (see XML Schema [XML Schema Part 2]). A SOAP receiver MUST accept any valid lexical representation of the attribute value.

If relaying the message, a SOAP intermediary MAY substitute "true" for the value "1", or "false" for "0". In addition, a SOAP intermediary MAY omit a SOAP `relay` *attribute information item* if its value is "false" (see **2.7 Relaying SOAP Messages**).

A SOAP sender generating a SOAP message SHOULD use the `relay` *attribute information item* only on SOAP header blocks. A SOAP receiver MUST ignore this *attribute information item* if it appears on descendants of a SOAP header block or on a SOAP body child *element information item* (or its descendents).

5.3 SOAP Body

A SOAP body provides a mechanism for transmitting information to an ultimate SOAP receiver (see **2.5 Structure and Interpretation of SOAP Bodies**).

The `Body` *element information item* has:

- A [local name] of `Body` .

- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- Zero or more namespace qualified *attribute information items* in its [attributes] property.
- Zero or more namespace qualified *element information items* in its [children] property.

The `Body` *element information item* MAY have any number of *character information item* children whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0]. These are considered significant.

5.3.1 SOAP Body child Element

All child *element information items* of the SOAP `Body` *element information item*:

- SHOULD have a [namespace name] property which has a value, that is the name of the element SHOULD be namespace qualified.

  Note:

  Namespace qualified elements tend to produce messages whose interpretation is less ambiguous than those with unqualified elements. The use of unqualified elements is therefore discouraged.

- MAY have any number of *character information item* children. Child *character information items* whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0] are considered significant.
- MAY have any number of *element information item* children. Such *element information items* MAY be namespace qualified.
- MAY have zero or more *attribute information items* in its [attributes] property. Among these MAY be the following, which has special significance for SOAP processing:

  - `encodingStyle` *attribute information item* (see **5.1.1 SOAP encodingStyle Attribute**).

SOAP defines one particular direct child of the SOAP body, the SOAP fault, which is used for reporting errors (see **5.4 SOAP Fault**).

5.4 SOAP Fault

A SOAP fault is used to carry error information within a SOAP message.

The `Fault` *element information item* has:

- A [local name] of `Fault` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- Two or more child *element information item*s in its [children] property in order as follows:

    1. A mandatory `Code` *element information item* (see **5.4.1 SOAP Code Element**).
    2. A mandatory `Reason` *element information item* (see **5.4.2 SOAP Reason Element**).
    3. An optional `Node` *element information item* (see **5.4.3 SOAP Node Element**).
    4. An optional `Role` *element information item* (see **5.4.4 SOAP Role Element**).
    5. An optional `Detail` *element information item* (see **5.4.5 SOAP Detail Element**).

To be recognized as carrying SOAP error information, a SOAP message MUST contain a single SOAP `Fault` *element information item* as the only child *element information item* of the SOAP `Body` .

When generating a fault, SOAP senders MUST NOT include additional *element information items* in the SOAP `Body` . A message whose `Body` contains a `Fault` plus additional *element information items* has no SOAP-defined semantics.

A SOAP `Fault` *element information item* MAY appear within a SOAP header block, or as a descendant of a child *element information item* of the SOAP `Body` ; in such cases, the element has no SOAP-defined semantics.

5.4.1 SOAP Code Element

The `Code` *element information item* has:

- A [local name] of `Code` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .
- One or two child *element information item*s in its [children] property, in order, as follows:

    1. A mandatory `Value` *element information item* as described below (see **5.4.1.1 SOAP Value element (with Code parent)**)

2. An optional `Subcode` *element information item* as described below (see **5.4.1.2 SOAP Subcode element**).

5.4.1.1 SOAP Value element (with Code parent)

The `Value` *element information item* has:

- A [local name] of `Value` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .

The type of the `Value` *element information item* is *env:faultCodeEnum*. SOAP defines a small set of SOAP fault codes covering high level SOAP faults (see **5.4.6 SOAP Fault Codes**).

5.4.1.2 SOAP Subcode element

The `Subcode` *element information item* has:

- A [local name] of `Subcode` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .
- One or two child *element information item*s in its [children] property, in order, as follows:

    1. A mandatory `Value` *element information item* as described below (see **5.4.1.3 SOAP Value element (with Subcode parent)**).
    2. An optional `Subcode` *element information item* (see **5.4.1.2 SOAP Subcode element**).

5.4.1.3 SOAP Value element (with Subcode parent)

The `Value` *element information item* has:

- A [local name] of `Value` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .

The type of the `Value` *element information item* is *xs:QName*. The value of this element is an application defined subcategory of the value of the `Value` child *element information item* of the `Subcode` *element information item's* parent *element information item* (see **5.4.6 SOAP Fault Codes**).

5.4.2 SOAP Reason Element

The `Reason` *element information item* is intended to provide a human readable explanation of the fault.

The `Reason` *element information item* has:

- A [local name] of `Reason` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .
- One or more `Text` *element information item* children (see **5.4.2.1 SOAP Text Element**). Each child `Text` *element information item* SHOULD have a different value for its `xml:lang` *attribute information item*.

The type of the `Reason` *element information item* is *env:faultReason*.

5.4.2.1 SOAP Text Element

The `Text` *element information item* is intended to carry the text of a human readable explanation of the fault.

The `Text` *element information item* has:

- A [local name] of `Text` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .
- A mandatory *attribute information item* with a [local name] of `lang` and [namespace name] of "http://www.w3.org/XML/1998/namespace". Note that the definition in of the `lang` *attribute information item* requires that the [prefix] is "xml" or any capitalization thereof (see XML 1.0 [XML 1.0], Language Identification).

- Any number of *character information item* children. Child *character information items* whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0] are considered significant.

The type of the `Text` *element information item* is *env:reasontext*

This *element information item* is similar to the 'Reason-Phrase' defined by HTTP [RFC 2616] and SHOULD provide information explaining the nature of the fault. It is not intended for algorithmic processing.

5.4.3 SOAP Node Element

The `Node` *element information item* is intended to provide information about which SOAP node on the SOAP message path caused the fault to happen (see **2. SOAP Processing Model**).

The `Node` *element information item* has:

- A [local name] of `Node` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .

The type of the `Node` *element information item* is *xs:anyURI*.

As described in section **2.1 SOAP Nodes**, each SOAP node is identified by a URI. The value of the `Node` *element information item* is the URI that identifies the SOAP node that generated the fault. SOAP nodes that do not act as the ultimate SOAP receiver MUST include this *element information item*. An ultimate SOAP receiver MAY include this *element information item* to indicate explicitly that it generated the fault.

5.4.4 SOAP Role Element

The `Role` *element information item* identifies the role the node was operating in at the point the fault occurred.

The `Role` *element information item* has:

- A [local name] of `Role` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .

The type of the `Role` *element information item* is *xs:anyURI*.

The value of the `Role` *element information item* MUST be one of the roles assumed by the node during processing of the message (see **2.2 SOAP Roles and SOAP Nodes**).

5.4.5 SOAP Detail Element

The `Detail` *element information item* is intended for carrying application specific error information related to the SOAP `Body` .

The `Detail` *element information item* has:

- A [local name] of `Detail` .
- A [namespace name] of `http://www.w3.org/2003/05/soap-envelope` .
- Zero or more *attribute information item*s in its [attributes] property.
- Zero or more child *element information item*s in its [children] property.

The `Detail` *element information item* MAY have any number of *character information item* children whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0]. These are considered significant.

The `Detail` *element information item* MAY be present in a SOAP fault in which case it carries additional information relative to the SOAP fault codes describing the fault (see **5.4.6 SOAP Fault Codes**). For example, the `Detail` *element information item* might contain information about a message not containing the proper credentials, a timeout, etc. The presence of the `Detail` *element information item* has no significance as to which parts of the faulty SOAP message were processed.

All child *element information items* of the `Detail` *element information item* are called detail entries (see **5.4.5.1 SOAP detail entry**).

5.4.5.1 SOAP detail entry

Each detail entry:

- MAY have a [namespace name] property which has a value, that is the name of the element MAY be namespace qualified.

- MAY have any number of *element information item* children.
- MAY have any number of *character information item* children. Child *character information items* whose character code is amongst the white space characters as defined by XML 1.0 [XML 1.0] are considered significant.
- MAY have zero or more *attribute information items* in its [attributes] property. Among these MAY be the following, which has special significance for SOAP processing:

    - `encodingStyle` *attribute information item* (see **5.1.1 SOAP encodingStyle Attribute**).

If present, the SOAP `encodingStyle` *attribute information item* indicates the encoding style used for the detail entry (see **5.1.1 SOAP encodingStyle Attribute**).

5.4.6 SOAP Fault Codes

SOAP fault codes are XML expanded names, and are intended to provide a means by which faults are classified. A hierarchical list of SOAP codes and associated supporting information is included in every SOAP fault message, with each such code identifying the fault category at an increasing level of detail.

The values of the `Value` child *element information item* of the `Code` *element information item* are restricted to those defined by the `env:faultCodeEnum` type (see **Table 4**). Additional fault subcodes MAY be created for use by applications or features. Such subcodes are carried in the `Value` child *element information item* of the `Subcode` *element information item*.

SOAP fault codes are to be interpreted as modifiers of the contents of the `Detail` *element information item* in the sense that they provide the context for the `Detail` *element information item*. A SOAP node MUST understand all SOAP fault codes in a SOAP fault message in order to be able to interpret the `Detail` *element information item* in a SOAP fault.

Example 4: Sample SOAP fault where the `Detail` *element information item* is to be interpreted in the context of the "env:Sender" and "m: MessageTimeout" fault codes.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
soap-envelope"
              xmlns:m="http://www.example.org/
timeouts"
              xmlns:xml="http://www.w3.org/XML/1998/
namespace">
 <env:Body>
  <env:Fault>
   <env:Code>
     <env:Value>env:Sender</env:Value>
     <env:Subcode>
      <env:Value>m:MessageTimeout</env:Value>
     </env:Subcode>
   </env:Code>
   <env:Reason>
     <env:Text xml:lang="en">Sender Timeout</env:Text>
   </env:Reason>
   <env:Detail>
     <m:MaxTime>P5M</m:MaxTime>
   </env:Detail>
  </env:Fault>
```

```
  </env:Body>
</env:Envelope>
```

This specification does not define a limit for how many `Subcode` *element information items* a SOAP fault might contain. However, while not a requirement of this specification, it is anticipated that most practical examples can be supported by relatively few `Subcode` *element information items*.

Table 4: SOAP Fault Codes

Local Name	Meaning
VersionMismatch	The faulting node found an invalid *element information item* instead of the expected `Envelope` *element information item*. The namespace, local name or both did not match the `Envelope` *element information item* required by this recommendation (see **2.8 SOAP Versioning Model** and **5.4.7 VersionMismatch Faults**)
MustUnderstand	An immediate child *element information item* of the SOAP `Header` *element information item* targeted at the faulting node that was not understood by the faulting node contained a SOAP `mustUnderstand` *attribute information item* with a value of "true" (see **5.2.3 SOAP mustUnderstand Attribute** and **5.4.8 SOAP mustUnderstand Faults**)
DataEncodingUnknown	A SOAP header block or SOAP body child *element information item* targeted at the faulting SOAP node is scoped (see **5.1.1 SOAP encodingStyle Attribute**) with a data encoding that the faulting node does not support.
Sender	The message was incorrectly formed or did not contain the appropriate information in order to succeed. For example, the message could lack the proper authentication or payment information. It is generally an indication that the message is not to be resent without change (see also **5.4 SOAP Fault** for a description of the SOAP fault `detail` sub-element).

Receiver	The message could not be processed for reasons attributable to the processing of the message rather than to the contents of the message itself. For example, processing could include communicating with an upstream SOAP node, which did not respond. The message could succeed if resent at a later point in time (see also **5.4 SOAP Fault** for a description of the SOAP fault `detail` sub-element).

5.4.7 VersionMismatch Faults

When a SOAP node generates a fault with a `Value` of `Code` set to "env:VersionMismatch", it SHOULD provide an `Upgrade` SOAP header block in the generated fault message. The `Upgrade` SOAP header block, as described below, details the XML qualified names (per XML Schema [XML Schema Part 2]) of the supported SOAP envelopes that the SOAP node supports (see **2.8 SOAP Versioning Model**).

5.4.7.1 SOAP Upgrade Header Block

The `Upgrade` SOAP header block consists of an `Upgrade` *element information item* containing an ordered list of XML qualified names of SOAP envelopes that the SOAP node supports in the order most to least preferred.

The `Upgrade` *element information item* has:

- A [local name] of `Upgrade` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- One or more `SupportedEnvelope` *element information item*s in its [children] property in **5.4.7.2 SOAP SupportedEnvelope Element**.

The `Upgrade` *element information item* MUST NOT have an `encodingStyle` *attribute information item*.

5.4.7.2 SOAP SupportedEnvelope Element

The `SupportedEnvelope` *element information item* has:

- A [local name] of `SupportedEnvelope` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".

- A qname *attribute information item* in its [attributes] property as described in **5.4.7.3 SOAP QName Attribute**.

5.4.7.3 SOAP QName Attribute

The qname *attribute information item* has the following XML infoset properties:

- A [local name] of qname .
- A [namespace name] which has no value.
- A [specified] property with a value of "true".

The type of the qname *attribute information item* is *xs:QName*. Its value is the XML qualified name of a SOAP Envelope *element information item* that the faulting node can understand.

Note:

When serializing the qname *attribute information item* there needs to be an in-scope namespace declaration for the namespace name of the SOAP Envelope *element information item* that the faulting node can understand. The value of the *attribute information item* uses the prefix of such a namespace declaration.

5.4.7.4 VersionMismatch Example

The following example illustrates the case of a SOAP node that supports both SOAP Version 1.2 and SOAP/1.1 but which prefers SOAP Version 1.2 (see appendix **A. Version Transition From SOAP/1.1 to SOAP Version 1.2** for a mechanism for transitioning from SOAP/1.1 to SOAP Version 1.2). This is indicated by including an Upgrade SOAP header block with two SupportedEnvelope *element information items*, the first containing the local name and namespace name of the SOAP Version 1.2 Envelope *element information item*, the latter the local name and namespace name of the SOAP/1.1 Envelope element.

Example 5: Version mismatch fault generated by a SOAP node. The message includes a SOAP Upgrade header block indicating support for both SOAP Version 1.2 and SOAP/1.1 but with a preference for SOAP Version 1.2.

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/
```

```
soap-envelope"
              xmlns:xml="http://www.w3.org/XML/1998/
namespace">
 <env:Header>
  <env:Upgrade>
   <env:SupportedEnvelope qname="ns1:Envelope"
             xmlns:ns1="http://www.w3.org/2003/05/
soap-envelope"/>
   <env:SupportedEnvelope qname="ns2:Envelope"
             xmlns:ns2="http://schemas.xmlsoap.org/
soap/envelope/"/>
  </env:Upgrade>
 </env:Header>
 <env:Body>
  <env:Fault>
   <env:Code><env:Value>env:VersionMismatch</env:
Value></env:Code>
    <env:Reason>
      <env:Text xml:lang="en">Version Mismatch</env:
Text>
    </env:Reason>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

5.4.8 SOAP mustUnderstand Faults

When a SOAP node generates a fault with a `Value` of `Code` set to "env: MustUnderstand", it SHOULD provide `NotUnderstood` SOAP header blocks in the generated fault message. The `NotUnderstood` SOAP header blocks, as described below, detail the XML qualified names (per XML Schema [XML Schema Part 2]) of the particular SOAP header block(s) which were not understood.

A SOAP node MAY generate a SOAP fault for any one or more SOAP header blocks that were not understood in a SOAP message. It is not a requirement that the fault contain the XML qualified names of all such SOAP header blocks.

5.4.8.1 SOAP NotUnderstood Element

Each `NotUnderstood` header block *element information item* has:

- A [local name] of `NotUnderstood` .

- A [namespace name] of "http://www.w3.org/2003/05/soap-envelope".
- A `qname` *attribute information item* in its [attributes] property as described in **5.4.8.2 SOAP QName Attribute**.

The `NotUnderstood` *element information item* MUST NOT have an `encodingStyle` *attribute information item*.

5.4.8.2 SOAP QName Attribute

The `qname` *attribute information item* has the following XML infoset properties:

- A [local name] of `qname` .
- A [namespace name] which has no value.
- A [specified] property with a value of "true".

The type of the `qname` *attribute information item* is *xs:QName*. Its value is the XML qualified name of a SOAP header block which the faulting node failed to understand.

Note:

When serializing the `qname` *attribute information item* there needs to be an in-scope namespace declaration for the namespace name of the SOAP header block that was not understood and the value of the *attribute information item* uses the prefix of such a namespace declaration. The prefix used need not be the same as the one used in the SOAP message that was not understood.

5.4.8.3 NotUnderstood Example

Consider the following example message:

Example 6: SOAP envelope that will cause a fault if `Extension1` or `Extension2` are not understood

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env='http://www.w3.org/2003/05/
soap-envelope'>
  <env:Header>
    <abc:Extension1 xmlns:abc='http://example.
org/2001/06/ext'
       env:mustUnderstand='true'/>
    <def:Extension2 xmlns:def='http://example.com/
stuff'
       env:mustUnderstand='true' />
```

```
    </env:Header>
    <env:Body>
      . . .
    </env:Body>
  </env:Envelope>
```

The message in the above example will result in the fault message shown in the example below if the ultimate receiver of the SOAP message does not understand the two SOAP header blocks `abc:Extension1` and `def:Extension2` .

Example 7: SOAP fault generated as a result of not understanding `Extension1` and `Extension2`

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env='http://www.w3.org/2003/05/
soap-envelope'
              xmlns:xml='http://www.w3.org/XML/1998/
namespace'>
 <env:Header>
  <env:NotUnderstood qname='abc:Extension1'
                     xmlns:abc='http://example.
org/2001/06/ext' />
  <env:NotUnderstood qname='def:Extension2'
                     xmlns:def='http://example.com/
stuff' />
 </env:Header>
 <env:Body>
  <env:Fault>
   <env:Code><env:Value>env:MustUnderstand</env:
Value></env:Code>
   <env:Reason>
     <env:Text xml:lang='en'>One or more mandatory
       SOAP header blocks not understood
     </env:Text>
   </env:Reason>
  </env:Fault>
 </env:Body>
</env:Envelope>
```

6. Use of URIs in SOAP

SOAP uses URIs for some identifiers including, but not limited to, values of the `encodingStyle` (see **5.1.1 SOAP encodingStyle Attribute**) and `role` (see **5.2.2 SOAP role Attribute**) *attribute information items*. To SOAP, a URI is simply a formatted string that identifies a web resource via its name, location, or any other characteristics.

Although this section only applies to URIs directly used by *information items* defined by this specification, it is RECOMMENDED that application-defined data carried within a SOAP envelope use the same mechanisms and guidelines defined here for handling URIs.

URIs used as values in *information items* identified by the "http://www.w3.org/2003/05/soap-envelope" and "http://www.w3.org/2003/05/soap-encoding" XML namespaces can be either relative or absolute.

SOAP does not define a base URI but relies on the mechanisms defined in XML Base [XML Base] and RFC 2396 [RFC 2396] for establishing a base URI against which relative URIs can be made absolute.

The underlying protocol binding MAY define a base URI which can act as the base URI for the SOAP envelope (see **4. SOAP Protocol Binding Framework** and SOAP 1.2 Part 2 [SOAP Part 2], section HTTP binding).

SOAP does not define any equivalence rules for URIs in general as these are defined by the individual URI schemes and by RFC 2396 [RFC 2396]. However, because of inconsistencies with respect to URI equivalence rules in many current URI parsers, it is RECOMMENDED that SOAP senders do not rely on any special equivalence rules in SOAP receivers in order to determine equivalence between URI values used in a SOAP message.

The use of IP addresses in URIs SHOULD be avoided whenever possible (see RFC 1900 [RFC 1900]). However, when used, the literal format for IPv6 addresses in URIs as described by RFC 2732 [RFC 2732] SHOULD be supported.

SOAP does not place any a priori limit on the length of a URI. Any SOAP node MUST be able to handle the length of any URI that it publishes and both SOAP senders and SOAP receivers SHOULD be able to deal with URIs of at least 2048 characters in length.

7. Security Considerations

The SOAP Messaging Framework does not directly provide any mechanisms for dealing with access control, confidentiality, integrity and non-repudiation.

Such mechanisms can be provided as SOAP extensions using the SOAP extensibility model (see **3. SOAP Extensibility Model**). This section describes the security considerations that designers and implementors need to take into consideration when designing and using such mechanisms.

SOAP implementors need to anticipate rogue SOAP applications sending intentionally malicious data to a SOAP node (see **2. SOAP Processing Model**). It is strongly recommended that a SOAP node receiving a SOAP message is capable of evaluating to what level it can trust the sender of that SOAP message and its contents.

7.1 SOAP Nodes

SOAP can carry application-defined data as SOAP header blocks or as SOAP body contents. Processing a SOAP header block might include dealing with side effects such as state changes, logging of information, or the generation of additional messages. It is strongly recommended that, for any deployment scenario, only carefully specified SOAP header blocks with well understood security implications of any side effects be processed by a SOAP node.

Similarly, processing the SOAP body might imply the occurrence of side effects that could, if not properly understood, have severe consequences for the receiving SOAP node. It is strongly recommended that only well-defined body contents with known security implications be processed.

Security considerations, however, are not just limited to recognizing the immediate child elements of a SOAP header block and the SOAP body. Implementors need to pay special attention to the security implications of all data carried within a SOAP message that can cause the remote execution of any actions in the receiver's environment. This includes not only data expressed as XML infoset properties but data that might be encoded as property values including binary data or parameters, for example URI query strings. Before accepting data of any type, an application ought to be aware of the particular security implications associated with that data within the context it is being used.

SOAP implementors need to be careful to ensure that if processing of the various parts of a SOAP message is provided through modular software architecture, that each module is aware of the overall security context. For example, the SOAP body ought not to be processed without knowing the context in which it was received.

7.2 SOAP Intermediaries

SOAP inherently provides a distributed processing model that might involve a SOAP message passing through multiple SOAP nodes (see **2. SOAP Processing Model**). SOAP intermediaries are by definition men-in-the-middle, and represent an opportunity for man-in-the-middle attacks. Security breaches on systems that run SOAP intermediaries can result in serious security and privacy problems. A compromised SOAP intermediary, or an intermediary implemented or configured without regard to security and privacy considerations, might be used in the commission of a wide range of potential attacks.

In analyzing the security implications of potential SOAP related security problems, it is important to realize that the scope of security mechanisms provided by the underlying protocol might not be the same scope as the whole message path of the SOAP message. There is no requirement in SOAP that all hops between participating SOAP nodes use the same underlying protocol and even if this were the case, the very use of SOAP intermediaries is likely to reach beyond the scope of transport-level security.

7.3 Underlying Protocol Bindings

The effects on security of not implementing a MUST or SHOULD, or doing something the specification says MUST NOT or SHOULD NOT be done can be very subtle. Binding specification authors ought to describe, in detail, the security implications of not following recommendations or requirements as most implementors will not have had the benefit of the experience and discussion that produced the specification (see **4. SOAP Protocol Binding Framework**).

In addition, a binding specification might not address or provide countermeasures for all aspects of the inherent security risks. The binding specification authors ought to identify any such risks as might remain and indicate where further countermeasures would be needed above and beyond those provided for in the binding specification.

Authors of binding specifications need to be aware that SOAP extension modules expressed as SOAP header blocks could affect the underlying protocol in unforeseen ways. A SOAP message carried over a particular protocol binding might result in seemingly conflicting features. An example of this is a SOAP message carried over HTTP, using the HTTP basic authentication mechanism in combination with a SOAP based authentication mechanism. It is strongly recommended that a binding specification describes any such interactions between the extensions and the underlying protocols.

7.3.1 Binding to Application-Specific Protocols

Some underlying protocols could be designed for a particular purpose or application profile. SOAP bindings to such protocols MAY use the same endpoint identification (e.g., TCP port number) as the underlying protocol, in order to reuse the existing infrastructure associated with that protocol.

However, the use of well-known ports by SOAP might incur additional, unintended handling by intermediaries and underlying implementations. For example, HTTP is commonly thought of as a "Web browsing" protocol, and network administrators might place certain restrictions upon its use, or could interpose services such as filtering, content modification, routing, etc. Often, these services are interposed using port number as a heuristic.

As a result, binding definitions for underlying protocols with well-known default ports or application profiles SHOULD document potential interactions with commonly deployed infrastructure at those default ports or in conformance with default application profiles. Binding definitions SHOULD also illustrate the use of the binding on a non-default port as a means of avoiding unintended interaction with such services.

8. References

8.1 Normative References

[SOAP Part 2]
W3C Proposed Recommendation "SOAP Version 1.2 Part 2: Adjuncts", Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part2-20030624.)

[RFC 2616]
IETF "RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk Nielsen, T. Berners-Lee, January 1997. (See http://www.ietf.org/rfc/rfc2616.txt.)

[RFC 2119]
IETF "RFC 2119: Keywords for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997. (See http://www.ietf.org/rfc/rfc2119.txt.)

[XML Schema Part 1]
W3C Recommendation "XML Schema Part 1: Structures", Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, 2 May 2001. (See http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.)

[XML Schema Part 2]
W3C Recommendation "XML Schema Part 2: Datatypes", Paul V. Biron, Ashok Malhotra, 2 May 2001. (See http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.)

[RFC 2396]
IETF "RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, August 1998. (See http://www.ietf.org/rfc/rfc2396.txt.)

[Namespaces in XML]
W3C Recommendation "Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman, 14 January 1999. (See http://www.w3.org/TR/1999/REC-xml-names-19990114/.)

[XML 1.0]
W3C Recommendation "Extensible Markup Language (XML) 1.0 (Second Edition)", Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, 6 October 2000. (See http://www.w3.org/TR/2000/REC-xml-20001006.)

[XML InfoSet]
W3C Recommendation "XML Information Set", John Cowan, Richard Tobin, 24 October 2001. (See http://www.w3.org/TR/2001/REC-xml-infoset-20011024/.)

[XML Base]
W3C Recommendation "XML Base", Jonathan Marsh, 27 June 2001. (See http://www.w3.org/TR/2001/REC-xmlbase-20010627/.)

[RFC 2732]
IETF "RFC 2732: Format for Literal IPv6 Addresses in URL's", R. Hinden, B. Carpenter, L. Masinter, December 1999. (See http://www.ietf.org/rfc/rfc2732.txt.)

8.2 Informative References

[SOAP Part 0]
W3C Proposed Recommendation "SOAP Version 1.2 Part 0: Primer", Nilo Mitra, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part0-20030624.)

[XMLP Comments]
XML Protocol Comments Archive (See http://lists.w3.org/Archives/Public/xmlp-comments/.)

[XMLP Dist-App]
XML Protocol Discussion Archive (See http://lists.w3.org/Archives/Public/xml-dist-app/.)

[XMLP Charter]
XML Protocol Charter (See http://www.w3.org/2000/09/XML-Protocol-Charter.)

[XMLP Requirements]
W3C Working Draft "XML Protocol (XMLP) Requirements", Vidur Apparao, Alex Ceponkus, Paul Cotton, David Ezell, David Fallside,

Martin Gudgin, Oisin Hurley, John Ibbotson, R. Alexander Milowski, Kevin Mitchell, Jean-Jacques Moreau, Eric Newcomer, Henrik Frystyk Nielsen, Mark Nottingham, Waqar Sadiq, Stuart Williams, Amr Yassin, 24 June 2003. *This is work in progress.* (See http://www.w3.org/TR/2002/WD-xmlp-reqs-20020626.)

[SOAP Usage Scenarios]
W3C Working Draft "SOAP Version 1.2 Usage Scenarios", John Ibbotson 24 June 2003. *This is work in progress.* (See http://www.w3.org/TR/2002/WD-xmlp-scenarios-20020626/.)

[SOAP 1.1]
W3C Note "Simple Object Access Protocol (SOAP) 1.1", Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Nielsen, Satish Thatte, Dave Winer, 8 May 2000. (See http://www.w3.org/TR/SOAP/.)

[RFC 1900]
IETF "RFC 1900: Renumbering Needs Work", B. Carpenter, Y. Rekhter, February 1996. (See http://www.ietf.org/rfc/rfc1900.txt.)

A. Version Transition From SOAP/1.1 to SOAP Version 1.2

This appendix describes the version management rules for a SOAP node. If a SOAP node supports versioning from SOAP 1.1 to SOAP 1.2, then the SOAP node MUST implement the rules described in this appendix.

The rules for dealing with the possible SOAP/1.1 and SOAP Version 1.2 interactions are as follows:

1. A SOAP/1.1 node receiving a SOAP Version 1.2 message will according to SOAP/1.1 generate a version mismatch SOAP fault based on a SOAP/1.1 message construct. That is, the envelope will have a [local name] of `Envelope` and a [namespace name] of "http://schemas.xmlsoap.org/soap/envelope/".

2. A SOAP Version 1.2 node receiving a SOAP/1.1 message either:

   - MAY process the message as a SOAP/1.1 message (if supported), or

   - MUST generate a version mismatch SOAP fault based on a SOAP/1.1 message construct following SOAP/1.1 semantics using a SOAP/1.1 binding to the underlying protocol (see SOAP 1.1 [SOAP 1.1]). The SOAP fault SHOULD include an `Upgrade`

SOAP header block as defined in this specification (see **5.4.7 VersionMismatch Faults**) indicating support for SOAP Version 1.2. This allows a receiving SOAP/1.1 node to correctly interpret the SOAP fault generated by the SOAP Version 1.2 node.

The example below shows a version mismatch SOAP fault generated by a SOAP Version 1.2 node as a result of receiving a SOAP/1.1 message. The fault message is a SOAP/1.1 message with an `Upgrade` SOAP header block indicating support for SOAP Version 1.2.

Example 8: SOAP Version 1.2 node generating a SOAP/1.1 version mismatch fault message including an `Upgrade` SOAP header block indicating support for SOAP Version 1.2.

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/
soap/envelope/">
 <env:Header>
  <env:Upgrade>
   <env:SupportedEnvelope qname="ns1:Envelope"
             xmlns:ns1="http://www.w3.org/2003/05/
soap-envelope"/>
   </env:Upgrade>
  </env:Header>
  <env:Body>
   <env:Fault>
    <faultcode>env:VersionMismatch</faultcode>
    <faultstring>Version Mismatch</faultstring>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

Note:

SOAP nodes wishing to support both SOAP/1.1 and SOAP Version 1.2 are required to use a protocol binding associated with the appropriate version of SOAP.

Note:

An existing SOAP/1.1 node generating a version mismatch SOAP fault is not likely to indicate which versions it supports using the `Upgrade` *element information item* (see **5.4.7 VersionMismatch Faults**). If nothing is indicated then this means that SOAP/1.1 is the only

supported version. Note, however that incompatibilities between underlying protocol bindings might prevent a SOAP/1.1 node from generating a version mismatch SOAP fault when receiving a SOAP Version 1.2 message. For instance, a SOAP/1.1 node supporting the SOAP/1.1 HTTP binding (see SOAP 1.1 [SOAP 1.1]) receiving a SOAP Version 1.2 message using the SOAP 1.2 HTTP protocol binding (see SOAP 1.2 Part 2 [SOAP Part 2], SOAP HTTP Binding) might not understand the difference between the two bindings and generate an HTTP specific response as a result.

B. Acknowledgements (Non-Normative)

This specification is the work of the W3C XML Protocol Working Group.

Participants in the Working Group are (at the time of writing, and by alphabetical order): Carine Bournez (W3C), Michael Champion (Software AG), Glen Daniels (Macromedia, formerly of Allaire), David Fallside (IBM), Dietmar Gaertner (Software AG), Tony Graham (Sun Microsystems), Martin Gudgin (Microsoft Corporation, formerly of DevelopMentor), Marc Hadley (Sun Microsystems), Gerd Hoelzing (SAP AG), Oisin Hurley (IONA Technologies), John Ibbotson (IBM), Ryuji Inoue (Matsushita Electric), Kazunori Iwasa (Fujitsu Limited), Mario Jeckle (DaimlerChrysler R. & Tech), Mark Jones (AT&T), Anish Karmarkar (Oracle), Jacek Kopecky (Systinet/ Idoox), Yves Lafon (W3C), Michah Lerner (AT&T), Noah Mendelsohn (IBM, formerly of Lotus Development), Jeff Mischkinsky (Oracle), Nilo Mitra (Ericsson), Jean-Jacques Moreau (Canon), Masahiko Narita (Fujitsu Limited), Eric Newcomer (IONA Technologies), Mark Nottingham (BEA Systems, formerly of Akamai Technologies), David Orchard (BEA Systems, formerly of Jamcracker), Andreas Riegg (DaimlerChrysler R. & Tech), Hervé Ruellan (Canon), Jeff Schlimmer (Microsoft Corporation), Miroslav Simek (Systinet/ Idoox), Pete Wenzel (SeeBeyond), Volker Wiechers (SAP AG).

Previous participants were: Yasser alSafadi (Philips Research), Bill Anderson (Xerox), Vidur Apparao (Netscape), Camilo Arbelaez (WebMethods), Mark Baker (Idokorro Mobile (Planetfred), formerly of Sun Microsystems), Philippe Bedu (EDF (Electricité de France)), Olivier Boudeville (EDF (Electricité de France)), Don Box (Microsoft Corporation, formerly of DevelopMentor), Tom Breuel (Xerox), Dick Brooks (Group 8760), Winston Bumpus (Novell), David Burdett (Commerce One), Charles Campbell (Informix Software), Alex Ceponkus (Bowstreet), David Chappell (Sonic Software), Miles Chaston (Epicentric), David Clay (Oracle), David Cleary (Progress Software), Conleth O'Connell (Vignette), Ugo Corda (Xerox), Paul Cotton (Microsoft Corporation), Fransisco Cubera (IBM), Jim d'Augustine (eXcelon), Ron Daniel (Interwoven), Dug Davis (IBM), Ray Denenberg (Library of Congress), Paul Denning (MITRE), Frank DeRose (Tibco), Mike Dierken (DataChannel), Andrew

Eisenberg (Progress Software), Brian Eisenberg (DataChannel), Colleen Evans (Sonic Software), John Evdemon (XMLSolutions), David Ezell (Hewlett-Packard), Eric Fedok (Active Data Exchange), Chris Ferris (Sun Microsystems), Daniela Florescu (Propel), Dan Frantz (BEA Systems), Michael Freeman (Engenia Software), Scott Golubock (Epicentric), Rich Greenfield (Library of Congress), Hugo Haas (W3C), Mark Hale (Interwoven), Randy Hall (Intel), Bjoern Heckel (Epicentric), Erin Hoffman (Tradia), Steve Hole (MessagingDirect Ltd.), Mary Holstege (Calico Commerce), Jim Hughes (Fujitsu Software Corporation), Yin-Leng Husband (Hewlett-Packard, formerly of Compaq), Scott Isaacson (Novell), Murali Janakiraman (Rogue Wave), Eric Jenkins (Engenia Software), Jay Kasi (Commerce One), Jeffrey Kay (Engenia Software), Richard Koo (Vitria Technology Inc.), Alan Kropp (Epicentric), Julian Kumar (Epicentric), Peter Lecuyer (Progress Software), Tony Lee (Vitria Technology Inc.), Amy Lewis (TIBCO), Bob Lojek (Intalio), Henry Lowe (OMG), Brad Lund (Intel), Matthew MacKenzie (XMLGlobal Technologies), Murray Maloney (Commerce One), Richard Martin (Active Data Exchange), Highland Mary Mountain (Intel), Alex Milowski (Lexica), Kevin Mitchell (XMLSolutions), Ed Mooney (Sun Microsystems), Dean Moses (Epicentric), Don Mullen (Tibco), Rekha Nagarajan (Calico Commerce), Raj Nair (Cisco), Mark Needleman (Data Research Associates), Art Nevarez (Novell), Henrik Nielsen (Microsoft Corporation), Kevin Perkins (Compaq), Jags Ramnaryan (BEA Systems), Vilhelm Rosenqvist (NCR), Marwan Sabbouh (MITRE), Waqar Sadiq (Vitria Technology Inc.), Rich Salz (Zolera), Krishna Sankar (Cisco), George Scott (Tradia), Shane Sesta (Active Data Exchange), Lew Shannon (NCR), John-Paul Sicotte (MessagingDirect Ltd.), Simeon Simeonov (Allaire), Simeon Simeonov (Macromedia), Aaron Skonnard (DevelopMentor), Nick Smilonich (Unisys), Soumitro Tagore (Informix Software), James Tauber (Bowstreet), Lynne Thompson (Unisys), Patrick Thompson (Rogue Wave), Jim Trezzo (Oracle), Asir Vedamuthu (WebMethods), Randy Waldrop (WebMethods), Fred Waskiewicz (OMG), David Webber (XMLGlobal Technologies), Ray Whitmer (Netscape), Stuart Williams (Hewlett-Packard), Yan Xu (DataChannel), Amr Yassin (Philips Research), Susan Yee (Active Data Exchange), Jin Yu (Martsoft).

The people who have contributed to discussions on xml-dist-app@w3.org are also gratefully acknowledged.

SOAP Version 1.2 Part 2: Adjuncts

W3C Recommendation 24 June 2003

This version:
http://www.w3.org/TR/2003/REC-soap12-part2-20030624/
Latest version:
http://www.w3.org/TR/soap12-part2/
Previous versions:
http://www.w3.org/TR/2003/PR-soap12-part2-20030507/
Editors:
Martin Gudgin, Microsoft
Marc Hadley, Sun Microsystems
Noah Mendelsohn, IBM
Jean-Jacques Moreau, Canon
Henrik Frystyk Nielsen, Microsoft

Please refer to the **errata** for this document, which may include some normative corrections.

The English version of this specification is the only normative version. Non-normative translations may also be available.

Abstract

SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. SOAP Version 1.2 Part 2: Adjuncts defines a set of adjuncts that may be used with SOAP Version 1.2 Part 1: Messaging Framework. This specification depends on SOAP Version 1.2 Part 1: Messaging Framework [SOAP Part 1].

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document is a Recommendation of the W3C. This document has been produced by the XML Protocol Working Group, which is part of the Web Services Activity. It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

Comments on this document are welcome. Please send them to the public mailing-list xmlp-comments@w3.org (archive). It is inappropriate to send discussion email to this address.

Information about implementations relevant to this specification can be found in the Implementation Report at http://www.w3.org/2000/xp/Group/2/03/soap1.2implementation.html.

Patent disclosures relevant to this specification may be found on the Working Group's patent disclosure page, in conformance with W3C policy.

A list of current W3C Recommendations and other technical reports can be found at http://www.w3.org/TR.

Short Table of Contents

Table of Contents

Appendices

---

1. Introduction

SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchange of structured information in a decentralized, distributed environment. The SOAP specification consists of three parts. Part 2 (this document) defines a set of adjuncts that MAY be used with the SOAP messaging framework:

1. The SOAP Data Model represents application-defined data structures and values as a directed, edge-labeled graph of nodes (see **2. SOAP Data Model**).

2. The SOAP Encoding defines a set of rules for encoding instances of data that conform to the SOAP Data Model for inclusion in SOAP messages (see **3. SOAP Encoding**).

3. The SOAP RPC Representation defines a convention for how to use the SOAP Data Model for representing RPC calls and responses (see **4. SOAP RPC Representation**).

4. The section for describing features and bindings defines a convention for describing features and binding in terms of properties and property values (see **5. A Convention for Describing Features and Bindings**).

5. The section on SOAP-Supplied Message Exchange Patterns and Features defines a request response message exchange pattern and a message

exchange pattern supporting non-SOAP requests for SOAP responses, (see **6. SOAP-Supplied Message Exchange Patterns and Features**).

6. The SOAP Web Method feature defines a feature for control of methods used on the World Wide Web (see **6.4 SOAP Web Method Feature**).

7. The SOAP HTTP Binding defines a binding of SOAP to HTTP (see [RFC 2616]) following the rules of the SOAP Protocol Binding Framework, [SOAP Part 1] (see **7. SOAP HTTP Binding**).

SOAP 1.2 Part 0 [SOAP Part 0] is a non-normative document intended to provide an easily understandable tutorial on the features of the SOAP Version 1.2 specifications.

SOAP 1.2 Part 1 [SOAP Part 1] defines the SOAP messaging framework.

Note:

In previous versions of this specification the SOAP name was an acronym. This is no longer the case.

1.1 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC 2119].

This specification uses a number of namespace prefixes throughout; they are listed in **Table 1**. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see XML Infoset [XML InfoSet]).

Table 1: Prefixes and Namespaces used in this specification

Prefix	Namespace	Notes
env	"http://www.w3.org/2003/05/soap-envelope"	Defined by SOAP 1.2 Part 1 [SOAP Part 1].
enc	"http://www.w3.org/2003/05/soap-encoding"	A normative XML Schema [XML Schema Part 1], [XML Schema Part 2] document for the "http://www.w3.org/2003/05/soap-encoding" namespace can be found at http://www.w3.org/2003/05/soap-encoding.

rpc	"http://www.w3.org/2003/05/soap-rpc"	A normative XML Schema [XML Schema Part 1], [XML Schema Part 2] document for the "http://www.w3.org/2003/05/soap-rpc" namespace can be found at http://www.w3.org/2003/05/soap-rpc.
xs	"http://www.w3.org/2001/XMLSchema"	Defined in the W3C XML Schema specification [XML Schema Part 1], [XML Schema Part 2].
xsi	"http://www.w3.org/2001/XMLSchema-instance"	Defined in the W3C XML Schema specification [XML Schema Part 1], [XML Schema Part 2].

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URIs (see RFC 2396 [RFC 2396]).

This specification uses the Extended Backus-Naur Form (EBNF) as described in XML 1.0 [XML 1.0].

With the exception of examples and sections explicitly marked as "Non-Normative", all parts of this specification are normative.

2. SOAP Data Model

The SOAP Data Model represents application-defined data structures and values as a directed edge-labeled graph of nodes. Components of this graph are described in the following sections.

The purpose of the SOAP Data Model is to provide a mapping of non-XML based data to some wire representation. It is important to note that use of the SOAP Data Model, the accompanying SOAP Encoding (see **3. SOAP Encoding**), and/or the SOAP RPC Representation (see **4. SOAP RPC Representation**) is OPTIONAL. Applications which already model data in XML may not need to use the SOAP Data Model. Due to their optional nature, it is NOT a requirement to implement the SOAP Data Model, the SOAP Encoding and/or the SOAP RPC Representation as part of a SOAP node.

2.1 Graph Edges

Edges in the graph are said to *originate* at a graph node and *terminate* at a graph node. An edge that originates at a graph node is known as an *outbound edge* with respect to that graph node. An edge that terminates at a graph node is known as an *inbound edge* with respect to that graph node. An edge MAY originate and

terminate at the same graph node. An edge MAY have only an originating graph node, that is be outbound only. An edge MAY have only a terminating graph node, that is be inbound only.

The outbound edges of a given graph node MAY be distinguished by label or by position. Position is a total order on such edges; thus, if any outbound edges from a given node are distinguished by position, then all outbound edges from that node are so distinguished.

2.1.1 Edge labels

An edge label is an XML qualified name. Two edge labels are equal if and only if their XML expanded names are equal. I.e. both of the following are true:

1. Their local name values are the same.
2. Either of the following is true:
    1. Both of their namespace name values are missing.
    2. Their namespace name values are both present and are both the same.

See **2.3 Values** for uses of edge labels and position to distinguish the members of encoded values, and XML Schema [XML Schema Part 2] for more information about comparing XML qualified names.

2.2 Graph Nodes

A graph node has zero or more outbound edges. A graph node that has no outbound edges has an optional lexical value. All graph nodes have an optional type name of type *xs:QName* in the namespace named "http://www.w3.org/2001/XMLSchema" (see XML Schema [XML Schema Part 2]).

2.2.1 Single and Multi Reference Nodes

A graph node may be *single reference* or *multi reference*. A single reference graph node has a single inbound edge. A multi reference graph node has multiple inbound edges.

2.3 Values

A simple value is represented as a graph node with a lexical value.

A compound value is represented as a graph node with zero or more outbound edges as follows:

1. A graph node whose outbound edges are distinguished solely by their labels is known as a "struct". The outbound edges of a struct MUST be labeled with distinct names (see **2.1.1 Edge labels**).

2. A graph node whose outbound edges are distinguished solely by position is known as an "array". The outbound edges of an array MUST NOT be labeled.

3. SOAP Encoding

SOAP Encoding provides a means of encoding instances of data that conform to the data model described in **2. SOAP Data Model**. This encoding MAY be used to transmit data in SOAP header blocks and/or SOAP bodies. Other data models, alternate encodings of the SOAP Data Model as well as unencoded data MAY also be used in SOAP messages (see SOAP 1.2 Part 1 [SOAP Part 1], SOAP encodingStyle Attribute for specification of alternative encoding styles and see **4. SOAP RPC Representation** for restrictions on data models and encodings used to represent SOAP Remote Procedure Calls (RPC)).

The serialization rules defined in this section are identified by the URI "http://www.w3.org/2003/05/soap-encoding". SOAP messages using this particular serialization SHOULD indicate that fact by using the SOAP `encodingStyle` attribute information item (see SOAP 1.2 Part 1 [SOAP Part 1] SOAP encodingStyle Attribute).

3.1 Mapping between XML and the SOAP Data Model

XML allows very flexible encoding of data. SOAP Encoding defines a narrower set of rules for encoding the graphs described in **2. SOAP Data Model**. This section defines the encoding at a high level, and the subsequent sub-sections describe the encoding rules in more detail. The encodings described in this section can be used in conjunction with the mapping of RPC requests and responses specified in **4. SOAP RPC Representation**.

The encodings are described below from the perspective of a de-serializer. In each case, the presence of an XML serialization is presumed, and the mapping to a corresponding graph is described.

More than one encoding is typically possible for a given graph. When serializing a graph for transmission inside a SOAP message, a representation that deserializes to the identical graph MUST be used; when multiple such representations are possible, any of them MAY be used. When receiving an encoded SOAP message, all representations MUST be accepted.

3.1.1 Encoding Graph Edges and Nodes

Each graph edge is encoded as an *element information item* and each *element information item* represents a graph edge. **3.1.3 Encoding Compound Values** describes the relationship between edge labels and the [local name] and [namespace name] properties of such *element information items*.

The graph node at which an edge terminates is determined by examination of the serialized XML as follows:

1. If the *element information item* representing the edge does not have a `ref` *attribute information item* (see **3.1.5.2 ref Attribute Information Item**) among its attributes then that *element information item* is said to *represent* a node in the graph and the edge terminates at that node. In such cases the *element information item* represents both a graph edge and a graph node

2. If the *element information item* representing the edge does have a `ref` *attribute information item* (see **3.1.5.2 ref Attribute Information Item**) among its attributes, then the value of that *attribute information item* MUST be identical to the value of exactly one `id` *attribute information item* (see **3.1.5.1 id Attribute Information Item**) in the same envelope. In this case the edge terminates at the graph node represented by the *element information item* on which the `id` *attribute information item* appears. That *element information item* MUST be in the scope of an `encodingStyle` attribute with a value of "http://www.w3.org/2003/05/soap-encoding" (see SOAP 1.2 Part 1 [SOAP Part 1], SOAP encodingStyle Attribute).

All nodes in the graph are encoded as described in 1 above. Additional inbound edges for multi reference graph nodes are encoded as described in 2 above.

3.1.2 Encoding Simple Values

The lexical value of a graph node representing a simple value is the sequence of Unicode characters identified by the *character information item* children of the *element information item* representing that node. The *element information item* representing a simple value node MAY have among its attributes a 'nodeType' *attribute information item* (see **3.1.7 nodeType Attribute Information Item**). Note that certain Unicode characters cannot be represented in XML (see XML 1.0 [XML 1.0]).

3.1.3 Encoding Compound Values

An outbound edge of a graph node is encoded as an *element information item* child of the *element information item* that represents the node (see **3.1.1 Encoding Graph Edges and Nodes**). Particular rules apply depending on what kind of compound value the graph node represents. These rules are as follows:

1. For a graph edge which is distinguished by label, the [local name] and

[namespace name] properties of the child *element information item* together determine the value of the edge label.

2. For a graph edge which is distinguished by position:
    - The ordinal position of the graph edge corresponds to the position of the child *element information item* relative to its siblings
    - The [local name] and [namespace name] properties of the child *element information item* are not significant.
3. The element information item representing a compound value node MAY have among its attributes a `nodeType` *attribute information item* (see **3.1.7 nodeType Attribute Information Item**).
4. The following rules apply to the encoding of a graph node that represents an "array":
    - The *element information item* representing an array node MAY have among its attributes an `itemType` *attribute information item* (see **3.1.4.1 itemType Attribute Information Item**).
    - The *element information item* representing an array node MAY have among its attributes an `arraySize` *attribute information item* (see **3.1.6 arraySize Attribute Information Item**).
5. If a graph edge does not terminate in a graph node then it can either be omitted from the serialization or it can be encoded as an *element information item* with an *xsi:nil attribute information item* whose value is "true".

3.1.4 Computing the Type Name Property

The type name property of a graph node is a {namespace name, local name} pair computed as follows:

1. If the *element information item* representing the graph node has an `xsi:type` *attribute information item* among its attributes then the type name property of the graph node is the value of the `xsi:type` *attribute information item*.

    Note:

    This attribute is of type *xs:QName* (see XML Schema [XML Schema Part 2]); its value consists of the pair {namespace name, local name}. Neither the prefix used to construct the QName nor any information relating to any definition of the type is considered to be part of the value. The SOAP graph carries only the qualified name of the type.

2. Otherwise if the parent *element information item* of the *element information item* representing the graph node has an `enc:itemType` *attribute information item* (see **3.1.4.1 itemType Attribute Information Item**) among its attributes then the type name property of the graph node is the value of the `enc:itemType` *attribute information item*

3. Otherwise the value of the type name property of the graph node is unspecified.

Note:

These rules define how the type name property of a graph node in a graph is computed from a serialized encoding. This specification does not mandate validation using any particular schema language or type system. Nor does it include built in types or provide any standardized faults to reflect value/type name conflicts.

However, nothing prohibits development of additional specifications to describe the use of SOAP Encoding with particular schema languages or type systems. Such additional specifications MAY mandate validation using particular schema language, and MAY specify faults to be generated if validation fails. Such additional specifications MAY specify augmentations to the deserialized graph based on information determined from such a validation. The use by SOAP Encoding of xsi:type is intended to facilitate integration with the W3C XML Schema language (see **C. Using W3C XML Schema with SOAP Encoding**). Other XML based schema languages, data schemas and programmatic type systems MAY be used but only to the extent that they are compatible with the serialization described in this specification.

3.1.4.1 itemType Attribute Information Item

The `itemType` *attribute information item* has the following Infoset properties:

- A [local name] of `itemType` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-encoding".
- A [specified] property with a value of "true".

The type of the `itemType` *attribute information item* is *xs:QName*. The value of the `itemType` *attribute information item* is used to compute the type name property (see **3.1.4 Computing the Type Name Property**) of members of an array.

3.1.5 Unique identifiers

3.1.5.1 id Attribute Information Item

The `id` *attribute information item* has the following Infoset properties:

- A [local name] of `id` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-encoding".
- A [specified] property with a value of "true".

The type of the `id` *attribute information item* is *xs:ID*. The value of the `id` *attribute information item* is a unique identifier that can be referred to by a `ref` *attribute information item* (see **3.1.5.2 ref Attribute Information Item**).

3.1.5.2 ref Attribute Information Item

The `ref` *attribute information item* has the following Infoset properties:

- A [local name] of `ref` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-encoding".
- A [specified] property with a value of "true".

The type of the `ref` *attribute information item* is *xs:IDREF*. The value of the `ref` *attribute information item* is a reference to a unique identifier defined by an `id` *attribute information item* (see **3.1.5.1 id Attribute Information Item**).

3.1.5.3 Constraints on id and ref Attribute Information Items

The value of a `ref` *attribute information item* MUST also be the value of exactly one `id` *attribute information item*.

A `ref` *attribute information item* and an `id` *attribute information item* MUST NOT appear on the same *element information item*.

3.1.6 arraySize Attribute Information Item

The `arraySize` *attribute information item* has the following Infoset properties:

- A [local name] of `arraySize` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-encoding".

The type of the `arraySize` *attribute information item* is *enc:arraySize*. The value of the `arraySize` *attribute information item* MUST conform to the following EBNF grammar

```
[1]  arraySizeValue     ::=  ("*" | concreteSize)
                             nextConcreteSize*
[2]  nextConcreteSize   ::=  whitespace concreteSize
[3]  concreteSize       ::=  [0-9]+
[4]  white space        ::=  (#x20 | #x9 | #xD | #xA)+
```

The array's dimensions are represented by each item in the list of sizes (unspecified size in case of the asterisk). The number of items in the list represents the number of dimensions in the array. The asterisk, if present, MUST only appear in the first position in the list. The default value of the `arraySize` *attribute information item* is "*", that is by default arrays are considered to have a single dimension of unspecified size.

3.1.7 nodeType Attribute Information Item

The `nodeType` *attribute information item* has the following Infoset properties:

- A [local name] of `nodeType` .
- A [namespace name] of "http://www.w3.org/2003/05/soap-encoding".
- A [specified] property with a value of "true".

The type of the `nodeType` *attribute information item* is *enc:nodeType*.

The value of the `nodeType` *attribute information item* MUST, if present, be one of the strings "simple" or "struct" or "array". The value indicates what kind of a value this node represents - a simple value, a compound struct value or a compound array value respectively.

3.2 Decoding Faults

During deserialization a SOAP receiver:

- SHOULD generate an "env:Sender" SOAP fault with a subcode of `enc:MissingID` if the message contains a `ref` *attribute information item* but no corresponding `id` *attribute information item* (see **3.1.5.3 Constraints on id and ref Attribute Information Items**).
- SHOULD generate an "env:Sender" SOAP fault with a subcode of `enc:DuplicateID` if the message contains two or more `id` *attribute information item* that have the same value. (see **3.1.5.3 Constraints on id and ref Attribute Information Items**).
- MAY generate an "env:Sender" SOAP fault with a subcode of `enc:`

`UntypedValue` if the type name property of an encoded graph node is unspecified.

4. SOAP RPC Representation

One of the design goals of SOAP is to facilitate the exchange of messages that map conveniently to definitions and invocations of method and procedure calls in commonly used programming languages. For that purpose, this section defines a uniform representation of remote procedure call (RPC) requests and responses. It does not define actual mappings to any particular programming language. The representation is entirely platform independent and considerable effort has been made to encourage usage that is consistent with the Web in general.

As mentioned in section **2. SOAP Data Model**, use and implementation of the SOAP RPC Representation is OPTIONAL.

The SOAP `encodingStyle` attribute information item (see SOAP 1.2 Part 1 [SOAP Part 1] SOAP encodingStyle Attribute) is used to indicate the encoding style of the RPC representation. The encoding thus specified MUST support the **2. SOAP Data Model**. The encoding style defined in **3. SOAP Encoding** supports such constructs and is therefore suitable for use with the SOAP RPC Representation.

This SOAP RPC Representation is not predicated on any SOAP protocol binding. When SOAP is bound to HTTP, an RPC invocation maps naturally to an HTTP request and an RPC response maps to an HTTP response. (see **7. SOAP HTTP Binding**). However, the SOAP RPC Representation is not limited to the SOAP HTTP Binding.

To invoke an RPC, the following information is needed:

- The address of the target SOAP node.
- A procedure or method name.
- The identities and values of any arguments to be passed to the procedure or method. Arguments used to identify Web resources SHOULD be distinguished from those representing data or control information (see **4.1.1 Identification of RPC Resources**.)

- Values for properties as required by any features of the binding to be used. For example, "GET" or "POST" for the `http://www.w3.org/2003/05/soap/features/web-method/Method` property of the **6.4 SOAP Web Method Feature**.

- Optional header data.

SOAP RPC relies on the protocol binding to provide a mechanism for carrying the

URI of the target SOAP node. For HTTP the request URI indicates the resource against which the invocation is being made. Other than requiring it to be a valid URI, SOAP places no restriction on the form of an identifier (see RFC 2396 [RFC 2396] for more information on URIs). The section **4.1.1 Identification of RPC Resources** further discusses the use of URIs for identifying RPC resources.

The SOAP RPC Representation employs the **6.2 SOAP Request-Response Message Exchange Pattern** and **6.3 SOAP Response Message Exchange Pattern**. Use of the SOAP RPC Representation with other MEPs MAY be possible, but is beyond the scope of this specification.

4.1 Use of RPC on the World Wide Web

The following guidelines SHOULD be followed when deploying SOAP RPC applications on the World Wide Web.

4.1.1 Identification of RPC Resources

The World Wide Web identifies resources with URIs, but common programming conventions convey identification information in the arguments to procedures, or in the names of those procedures. For example, the call:

```
updateQuantityInStock(PartNumber="123", NewQuantity="200")
```

suggests that the resource to be updated is the `QuantityInStock` for `PartNumber` "123". Accordingly, when mapping to or from a programming language method or procedure call, any arguments that serve to identify resources (such as the part number above) should when practical be represented in the URI to which the SOAP message is addressed. When mapping to or from a programming language method or procedure call, the name of which identifies or qualifies the identification of a resource (such as QuantityInStock above), such naming or qualification should when practical be represented in the URI to which the SOAP message is addressed. No standard means of representation of arguments or method names is provided by this specification.

> **Note:**
>
> Conventions for specific URI encodings of procedure names and arguments, as well as for controlling the inclusion of such arguments in the SOAP RPC body could be established in conjunction with the development of Web Service interface description languages. They could be developed when SOAP is bound to particular programming languages or could be established on an application or procedure-specific basis.

4.1.2 Distinguishing Resource Retrievals from other RPCs

The World Wide Web depends on mechanisms that optimize commonly performed

information retrieval tasks. Specifically, protocols such as HTTP [RFC 2616] provide a GET method which is used to perform safe retrievals, i.e. to perform retrievals that are idempotent, free of side effects, and for which security considerations do not preclude the use of cached results or URI-based resource identification.

Certain procedure or method calls represent requests for information retrieval. For example, the call:

```
getQuantityInStock(PartNumber="123")
```

might be used to retrieve the quantity established in the example above.

The following conventions can be employed to implement SOAP retrievals and other RPCs on the Web:

- The conventions described in **4.1.1 Identification of RPC Resources** are used to identify the resource with a URI.

- In cases where all the arguments have been represented in the URI, no SOAP header blocks are to be transmitted and the operation is a safe retrieval, the **6.4 SOAP Web Method Feature** and the **6.3 SOAP Response Message Exchange Pattern** are used. Accordingly, no SOAP envelope is transmitted for the request, and the `http://www.w3.org/2003/05/soap/features/web-method/Method` property is set to "GET". The results of the retrieval are a SOAP RPC response as described in **4.2.2 RPC Response**

- In cases where the operation to be performed is not a retrieval, when SOAP header blocks are to be transmitted (a digital signature, for example), or when a retrieval is not safe, the **6.4 SOAP Web Method Feature** and the **6.2 SOAP Request-Response Message Exchange Pattern** are used. The request envelope is encoded as described in **4.2.1 RPC Invocation**, and the results are as described in **4.2.2 RPC Response**. The `http://www.w3.org/2003/05/soap/features/web-method/Method` property is set to "POST".

The SOAP RPC Representation does not define any other value for the `http://www.w3.org/2003/05/soap/features/web-method/Method` .

4.2 RPC and SOAP Body

RPC invocations (except for safe retrievals: see **4.1.2 Distinguishing Resource Retrievals from other RPCs**) and responses are both carried in the SOAP `Body` element (see SOAP 1.2 Part 1 [SOAP Part 1] SOAP Body) using the following representation:

4.2.1 RPC Invocation

An RPC invocation is modeled as follows:

- The invocation is represented by a single struct containing an outbound edge for each [in] or [in/out] parameter. The struct is named identically to the procedure or method name and the conventions of **B. Mapping Application Defined Names to XML Names** SHOULD be used to represent method names that are not legal XML names.
- Each outbound edge has a label corresponding to the name of the parameter. The conventions of **B. Mapping Application Defined Names to XML Names** SHOULD be used to represent parameter names that are not legal XML names.

Applications MAY process invocations with missing parameters but also MAY fail to process the invocation and return a fault.

4.2.2 RPC Response

An RPC response is modeled as follows:

- The response is represented by a single struct containing an outbound edge for the return value and each [out] or [in/out] parameter. The name of the struct is not significant.
- Each parameter is represented by an outbound edge with a label corresponding to the name of the parameter. The conventions of **B. Mapping Application Defined Names to XML Names** SHOULD be used to represent parameter names that are not legal XML names.
- A non-void return value is represented as follows:

  1. There MUST be an outbound edge with a local name of `result` and a namespace name of "http://www.w3.org/2003/05/soap-rpc" which terminates in a terminal node

  2. The type of that terminal node is a xs:QName and its value is the name of the outbound edge which terminates in the actual return value.

  If the return value of the procedure is void then an outbound edge with a local name of `result` and a namespace name of "http://www.w3.org/2003/05/soap-rpc" MUST NOT be present.

- Invocation faults are handled according to the rules in **4.4 RPC Faults**. If a protocol binding adds additional rules for fault expression, those MUST also be followed.

4.2.3 SOAP Encoding Restriction

When using SOAP encoding (see **3. SOAP Encoding**) in conjunction with the RPC convention described here, the SOAP `Body` MUST contain only a single child *element information item*, that child being the serialized RPC invocation or response struct.

4.3 RPC and SOAP Header

Additional information relevant to the encoding of an RPC invocation but not part of the formal procedure or method signature MAY be expressed in a SOAP envelope carrying an RPC invocation or response. Such additional information MUST be expressed as SOAP header blocks.

4.4 RPC Faults

The SOAP RPC Representation introduces additional SOAP fault subcode values to be used in conjunction with the fault codes described in SOAP 1.2 Part 1 [SOAP Part 1] SOAP Fault Codes.

Errors arising during RPC invocations are reported according to the following rules:

1. A fault with a `Value` of `Code` set to "env:Receiver" SHOULD be generated when the receiver cannot handle the message because of some temporary condition, e.g. when it is out of memory.

   Note:

   Throughout this document, the term "`Value` of `Code` " is used as a shorthand for "value of the `Value` child *element information item* of the `Code` *element information item*" (see SOAP 1.2 Part 1 [SOAP Part 1], SOAP Code Element).

2. A fault with a `Value` of `Code` set to "env:DataEncodingUnknown" SHOULD be generated when the arguments are encoded in a data encoding unknown to the receiver.

3. A fault with a `Value` of `Code` set to "env:Sender" and a `Value` of `Subcode` set to "rpc:ProcedureNotPresent" MAY be generated when the receiver does not support the procedure or method specified.

   Note:

   Throughout this document, the term "`Value` of `Subcode` " is used as a shorthand for "value of the `Value` child *element information item* of the `Subcode` *element information item*" (see SOAP 1.2 Part 1 [SOAP Part

1], SOAP Subcode element).

4. A fault with a `Value` of `Code` set to "env:Sender" and a `Value` of `Subcode` set to "rpc:BadArguments" MUST be generated when the receiver cannot parse the arguments or when there is a mismatch in number and/or type of the arguments between what the receiver expects and what was sent.

5. Other faults arising in an extension or from the application SHOULD be generated as described in SOAP 1.2 Part 1 [SOAP Part 1] SOAP Fault Codes.

In all cases the values of the `Detail` and `Reason` *element information items* are implementation defined. Details of their use MAY be specified by an external document.

Note:

Senders might receive different faults from those listed above in response to an RPC invocation if the receiver does not support the (optional) RPC convention described here.

5. A Convention for Describing Features and Bindings

This section describes a convention describing Features (including MEPs) and Bindings in terms of properties and property values. The convention is sufficient to describe the distributed states of Feature and Binding specifications as mandated by the Binding Framework (see SOAP 1.2 Part 1 [SOAP Part 1] SOAP Protocol Binding Framework) and it is used to describe a Request-Response MEP (see **6.2 SOAP Request-Response Message Exchange Pattern**), a Response MEP (see **6.3 SOAP Response Message Exchange Pattern**), the SOAP Web Method feature (see **6.4 SOAP Web Method Feature**) and the SOAP HTTP Binding (see **7. SOAP HTTP Binding**) elsewhere in this document. Along with the convention itself, an informal model is defined that describes how properties propagate through a SOAP system. Note that this model is intended to be illustrative only, and is not meant to imply any constraints on the structure or layering of any particular SOAP implementation.

5.1 Model and Properties

In general, a SOAP message is the information that one SOAP node wishes to exchange with another SOAP node according to a particular set of features, including a MEP. In addition, there may be information essential to exchanging a message that is not part of the message itself. Such information is sometimes called message metadata. In the model, the message, any message metadata, and the various information items that enable features are represented as abstractions called properties.

5.1.1 Properties

Under the convention, properties are represented as follows:

- Properties are named with URIs.
- Where appropriate, property values SHOULD have an XML Schema [XML Schema Part 1] [XML Schema Part 2] type listed in the specification which introduces the property.

5.1.2 Property Scope

Properties within a SOAP node differ in terms of their scope and the origins of their values. As shown in the figure below, we make the distinction between per message-exchange and more widely scoped properties by assigning them to different containers called Message Exchange Context and Environment Context respectively. All properties, regardless of their scope, are shared by a SOAP node and a particular Binding.

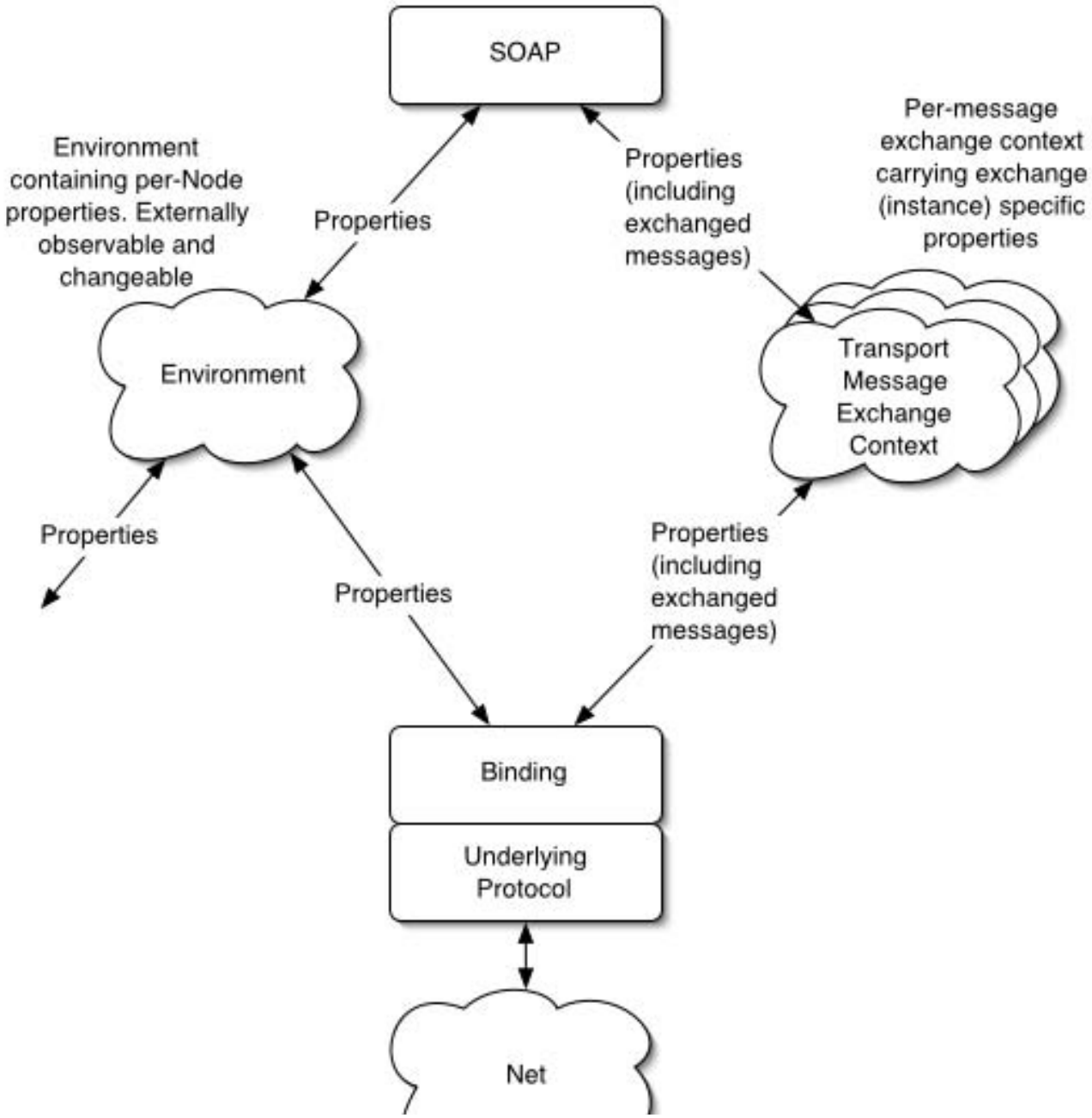


Figure 1: Model describing properties shared between SOAP and Binding

5.1.2.1 Message Exchange Context

A message exchange context is a collection of properties whose scope is limited to an instance of a given message exchange pattern. An example of a message exchange context property is the identifier of the message exchange pattern in use.

5.1.2.2 Environment Context

The environment context is a collection of properties whose scope extends beyond an instance of a given message exchange pattern. Examples of environment context properties are the IP address of the SOAP node or the current date and time.

The values of properties in Environment may depend upon local circumstances (as depicted by the external arrow from Environment in the figure above). More specifically, the properties in the example could be influenced by an operating system user ID on whose behalf a message exchange is being executed. The mapping of information in a particular implementation to such properties is outside the scope of the binding framework although the abstract representation of such information as properties is not.

5.1.3 Properties and Features

A feature may be expressed through multiple properties and a single property may enable more than one feature. For example, the properties called User ID and Password may be used to enable a feature called Authentication. As a second example, a single property called Message ID could be used to enable one feature called Transaction and a second feature called Message Correlation.

6. SOAP-Supplied Message Exchange Patterns and Features

6.1 Property Conventions for SOAP Message Exchange Patterns

Table 2 describes the properties (in accordance with the property naming conventions defined in this document) that support the description of message exchange patterns (MEPs). Other properties may be involved in the specification of particular MEPs, but the properties in this table are generally applicable to all MEPs.

Table 2: Property definitions supporting the description of MEPs

Property Name	Property Description	Property Type
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName`	The name of the MEP in operation.	xs:anyURI
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason`	A value that denotes a pattern specific, binding independent reason for the failure of a message exchange. Underlying protocol binding specifications may define properties to convey more binding specific details of the failure.	xs:anyURI
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role`	The identifier of the pattern specific role of the local SOAP node participating in the message exchange.	xs:anyURI
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State`	The identifier of the current state of the message exchange. This value is managed by the binding instance and may be inspected by other entities monitoring the progress of the message exchange.	xs:anyURI

6.2 SOAP Request-Response Message Exchange Pattern

This section defines the message exchange pattern (MEP) called "Request-Response". The description is an abstract presentation of the operation of this MEP. It is not intended to describe a real implementation or to suggest how a real implementation should be structured.

6.2.1 SOAP Feature Name

This message exchange pattern is identified by the URI (see SOAP 1.2 Part 1 [SOAP Part 1] SOAP Features):

- "http://www.w3.org/2003/05/soap/mep/request-response/"

6.2.2 Description

The SOAP Request-Response MEP defines a pattern for the exchange of a SOAP message acting as a request followed by a SOAP message acting as a response. In the absence of failure in the underlying protocol, this MEP consists of exactly two SOAP messages.

In the normal operation of a message exchange conforming to the Request-Response MEP, a request message is first transferred from the requesting SOAP node to the responding SOAP node. Following the successful processing of the request message by the responding SOAP node, a response message is transferred from the responding SOAP node to the requesting SOAP node.

Abnormal operation during a Request-Response message exchange might be caused by a failure to transfer the request message, a failure at the responding SOAP node to process the request message, or a failure to transfer the response message. Such failures might be silent at either or both of the requesting and responding SOAP nodes involved, or might result in the generation of a SOAP or binding-specific fault (see **6.2.4 Fault Handling**). Also, during abnormal operation each SOAP node involved in the message exchange might differ in its determination of the successful completion of the message exchange.

The scope of a Request-Response MEP is limited to the exchange of a request message and a response message between one requesting and one responding SOAP node. This pattern does not mandate any correlation between multiple requests nor specific timing for multiple requests. Implementations MAY choose to support multiple ongoing requests (and associated response processing) at the same time.

6.2.3 State Machine Description

The Request-Response MEP defines a set of properties described in **Table 3**.

Table 3: Property definitions for Request-Response MEP

Property Name	Property Description	Property Type

`http://www.w3.org/2003/05/soap/mep/OutboundMessage`	An abstract structure that represents the current outbound message in the message exchange. This abstracts both SOAP Envelope and any other information structures that are transferred along with the envelope.	Not specified
`http://www.w3.org/2003/05/soap/mep/InboundMessage`	An abstract structure that represents the current inbound message in the message exchange. This abstracts both SOAP Envelope and any other information structures that are transferred along with the envelope.	Not specified
`http://www.w3.org/2003/05/soap/mep/ImmediateDestination`	The identifier of the immediate destination of an outbound message.	xs:anyURI
`http://www.w3.org/2003/05/soap/mep/ImmediateSender`	The identifier of the immediate sender of an inbound message.	xs:anyURI

To initiate a message exchange conforming to the Request-Response MEP, the requesting SOAP node instantiates a local message exchange context. **Table 4** describes how the context is initialized.

Table 4: Instantiation of a Message Exchange Context for a requesting SOAP node

Property Name	Property Value	Notes

http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName	"http://www.w3.org/2003/05/soap/mep/request-response/"	
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason	"None"	A relative URI whose base URI is the value of http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role	"RequestingSOAPNode/"	A relative URI whose base URI is the value of http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State	"Init"	A relative URI whose base URI is the value of http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role
http://www.w3.org/2003/05/soap/mep/OutboundMessage	An abstraction of the request message	
http://www.w3.org/2003/05/soap/mep/ImmediateDestination	An identifier (URI) that denotes the responding SOAP node	

There may be other properties related to the operation of the message exchange context instance. Such properties are initialized according to their own feature specifications.

Once the message exchange context is initialized, control of the context is passed to a (conforming) local binding instance.

The diagram below shows the logical state transitions at the requesting and responding SOAP nodes during the lifetime of the message exchange. At each SOAP node, the local binding instance updates (logically) the value of the http://www.w3.org/2003/05/soap/bindingFramework/ ExchangeContext/State property to reflect the current state of the message exchange. The state

names are relative URIs, relative to a base URI value carried in the `http://www.w3.org/2003/05/soap/bindingFramework/ ExchangeContext/Role` property of the local message exchange context.

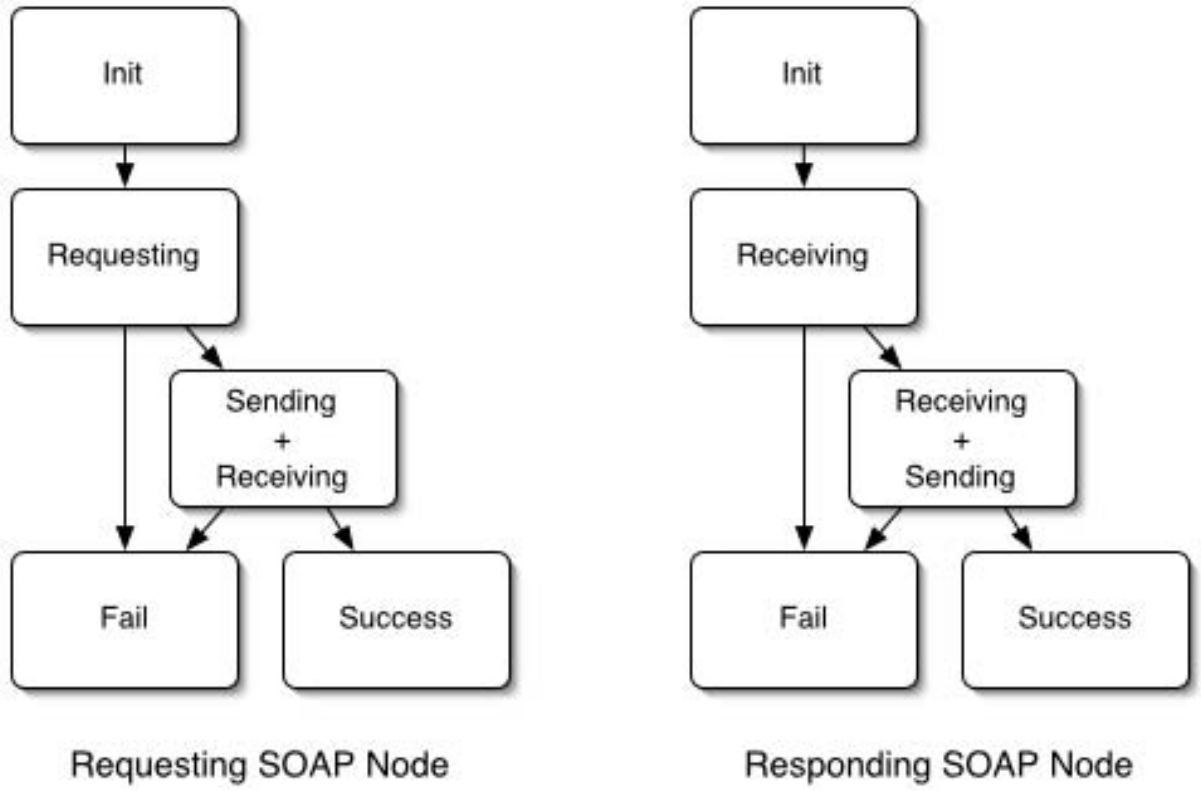


Figure 2: Request-Response MEP State Transition Diagram.

When the local binding instance at the responding SOAP node starts to receive an inbound request message, it (logically) instantiates a message exchange context. **Table 5** describes the properties that the binding initializes as part of the context's instantiation.

Table 5: Instantiation of Message Exchange Context for an inbound request message at a responding SOAP node

Property Name	Property Value	Notes
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName`	"http://www.w3.org/2003/05/soap/mep/request-response/"	Initialized as early as possible during the life cycle of the message exchange.
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason`	"None"	A relative URI whose base URI is the value of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName`

`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role`	"RespondingSOAPNode"	A relative URI whose base URI is the value of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName` Initialized as early as possible during the life cycle the message exchange.
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State`	"Init"	A relative URI whose base URI is the value of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role`

When the requesting and responding SOAP nodes transition between states, the local binding instance (logically) updates a number of properties. **Table 6** and **Table 7** describe these updates for the requesting and the responding SOAP nodes, respectively.

Table 6: Requesting SOAP Node State Transitions

CurrentState	Transition Condition	NextState	Action
"Init"	Unconditional	"Requesting"	Initiate transmission of request message abstracted in `http://www.w3.org/2003/05/soap/mep/OutboundMessage` .
"Requesting"	Message transmission failure	"Fail"	Set `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason` to "transmissionFailure"

	Start receiving response message	"Sending +Receiving"	Set `http://www.w3.org/2003/05/soap/mep/ImmediateSender` to denote the sender of the response message (may differ from the values in `http://www.w3.org/2003/05/soap/mep/ImmediateDestination`). Start making an abstraction of the response message available in `http://www.w3.org/2003/05/soap/mep/InboundMessage` .
"Sending +Receiving"	Message exchange failure	"Fail"	Set `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason` to "exchangeFailure"
	Completed sending request message. Completed receiving response message.	"Success"	

Table 7: Responding SOAP Node State Transitions

CurrentState	Transition Condition	NextState	Action
"Init"	Start receiving request message	"Receiving"	Set `http://www.w3.org/2003/05/soap/mep/ImmediateSender` to denote the sender of the request message (if determinable). Start making an abstraction of the request message available in `http://www.w3.org/2003/05/soap/mep/InboundMessage` . Pass control of message exchange context to

			SOAP processor.
"Receiving"	Message reception failure	"Fail"	Set `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason` to "receptionFailure".
	Start of response message available in `http://www.w3.org/2003/05/soap/mep/OutboundMessage`	"Receiving +Sending"	Initiate transmission of response message abstracted in `http://www.w3.org/2003/05/soap/mep/OutboundMessage` .
"Receiving +Sending"	Message exchange failure	"Fail"	Set `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason` to "exchangeFailure".
	Completed receiving request message. Completed sending response message.	"Success"	

Bindings that implement this MEP MAY provide for streaming of SOAP responses. That is, responding SOAP nodes MAY begin transmission of a SOAP response while a SOAP request is still being received and processed. When SOAP nodes implement bindings that support streaming, the following rules apply:

- All the rules in SOAP 1.2 Part 1 [SOAP Part 1] Binding Framework regarding streaming of individual SOAP messages MUST be obeyed for both request and response SOAP messages.
- When using streaming SOAP bindings, requesting SOAP nodes MUST avoid deadlock by accepting and if necessary processing SOAP response information while the SOAP request is being transmitted.

  Note:

  Depending on the implementation used and the size of the messages involved, this rule MAY require that SOAP applications stream application-level response processing in parallel with request generation.

- A requesting SOAP node MAY enter the "Fail" state, and thus abort transmission of the outbound SOAP request, based on information contained in an incoming streamed SOAP response.

6.2.4 Fault Handling

During the operation of the Request-Response MEP, the participating SOAP nodes may generate SOAP faults.

If a SOAP fault is generated by the responding SOAP node while it is in the "Receiving" state, the SOAP fault is made available in `http://www.w3.org/2003/05/soap/mep/OutboundMessage` and the state machine transitions to the "Receiving+Sending" state.

This MEP makes no claims about the disposition or handling of SOAP faults generated by the requesting SOAP node during any processing of the response message that follows the "Success" state in the requesting SOAP node's state transition table (see **Table 6**).

6.3 SOAP Response Message Exchange Pattern

This section defines the message exchange pattern (MEP) called "SOAP Response". The description is an abstract presentation of the operation of this MEP. It is not intended to describe a real implementation or to suggest how a real implementation should be structured.

6.3.1 SOAP Feature Name

This message exchange pattern is identified by the URI (see SOAP 1.2 Part 1 [SOAP Part 1] SOAP Features):

- "http://www.w3.org/2003/05/soap/mep/soap-response/"

6.3.2 Description

The SOAP Response MEP defines a pattern for the exchange of a non-SOAP message acting as a request followed by a SOAP message acting as a response. In the absence of failure in the underlying protocol, this MEP consists of exactly two messages, only one of which is a SOAP message:

- A request transmitted in a binding-specific manner that does not include a SOAP envelope and hence does not involve any SOAP processing by the receiving SOAP node.

- A response message which contains a SOAP envelope. The MEP is completed by the processing of the SOAP envelope following the rules of the SOAP processing model (see SOAP 1.2 Part 1 [SOAP Part 1], section SOAP Processing Model).

Abnormal operation during a SOAP Response message exchange might be caused

by a failure to transfer the request message or the response message. Such failures might be silent at either or both of the requesting and responding SOAP nodes involved, or might result in the generation of a SOAP or binding-specific fault (see section **6.3.4 Fault Handling**). Also, during abnormal operation each SOAP node involved in the message exchange might differ in its determination of the successful completion of the message exchange.

The scope of a SOAP Response MEP is limited to the request for an exchange of a response message between one requesting and one responding SOAP node. This pattern does not mandate any correlation between multiple requests nor specific timing for multiple requests. Implementations MAY choose to support multiple ongoing requests (and associated response processing) at the same time.

Note:

This MEP cannot be used in conjunction with features expressed as SOAP header blocks in the request because there is no SOAP envelope in which to carry them.

6.3.3 State Machine Description

The SOAP Response MEP defines a set of properties described in **Table 8**.

Table 8: Property definitions for SOAP Response MEP

Property Name	Property Description	Property Type
`http://www.w3.org/2003/05/soap/mep/OutboundMessage`	An abstract structure that represents the current outbound message in the message exchange. This abstracts both SOAP Envelope Infoset (which MAY be null) and any other information structures that are transferred along with the envelope.	Not specified

`http://www.w3.org/2003/05/soap/mep/InboundMessage`	An abstract structure that represents the current inbound message in the message exchange. This abstracts both SOAP Envelope Infoset (which MAY be null) and any other information structures that are transferred along with the envelope.	Not specified
`http://www.w3.org/2003/05/soap/mep/ImmediateDestination`	The identifier of the immediate destination of an outbound message.	xs:anyURI
`http://www.w3.org/2003/05/soap/mep/ImmediateSender`	The identifier of the immediate sender of an inbound message.	xs:anyURI

To initiate a message exchange conforming to the SOAP Response MEP, the requesting SOAP node instantiates a local message exchange context. **Table 9** describes how the context is initialized.

Table 9: Instantiation of a Message Exchange Context for a requesting SOAP node

Property Name	Property Value	Notes
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName`	"http://www.w3.org/2003/05/soap/mep/soap-response/"	
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason`	"None"	A relative URI whose base URI is the value of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName`

http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role	"RequestingSOAPNode"	A relative URI whose base URI is the value of http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName
http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State	"Init"	A relative URI whose base URI is the value of http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role
http://www.w3.org/2003/05/soap/mep/OutboundMessage	An abstraction of the request message that does not include a SOAP envelope infoset.	
http://www.w3.org/2003/05/soap/mep/ImmediateDestination	An identifier (URI) that denotes the responding SOAP node	

There may be other properties related to the operation of the message exchange context instance. Such properties are initialized according to their own feature specifications.

Once the message exchange context is initialized, control of the context is passed to a (conforming) local binding instance.

The diagram below shows the logical state transitions at the requesting and responding SOAP nodes during the lifetime of the message exchange. At each SOAP node, the local binding instance updates (logically) the value of the http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State property to reflect the current state of the message exchange. The state names are relative URIs, relative to a Base URI value carried in the http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role property of the local message exchange context.

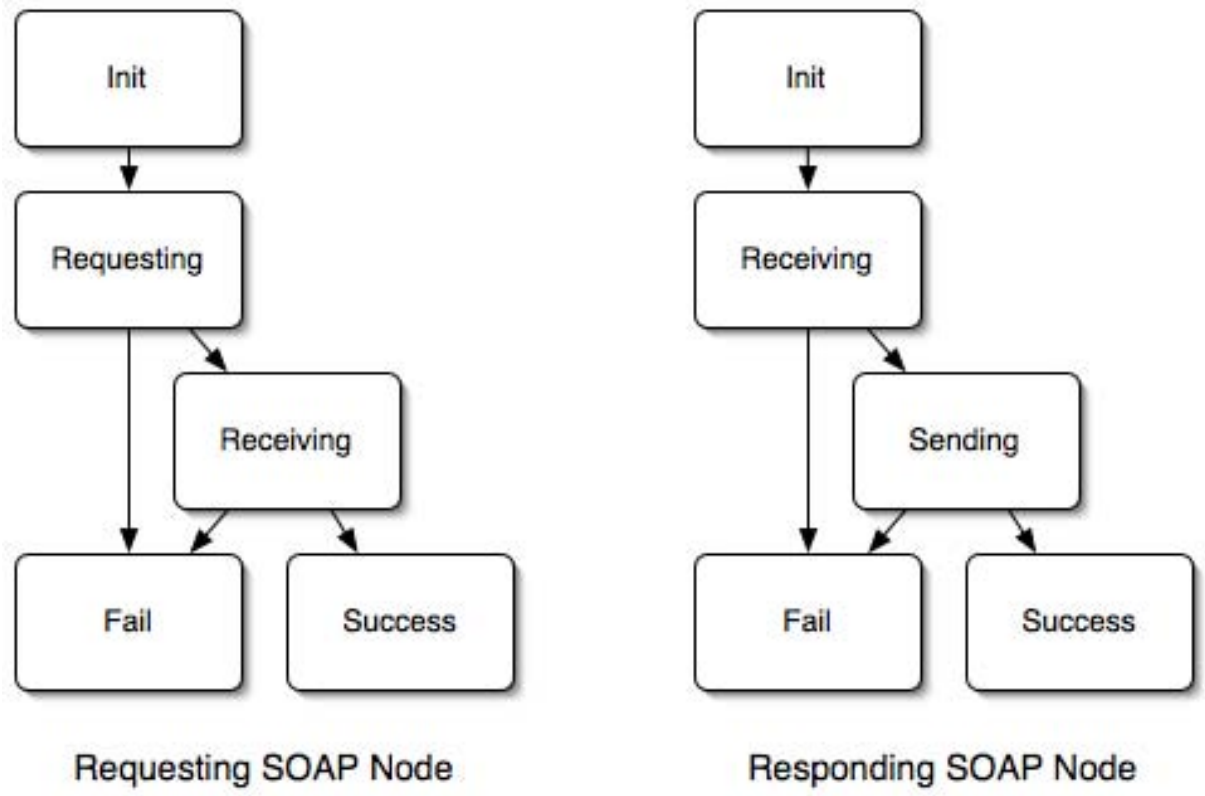


Figure 3: SOAP Response MEP State Transition Diagram

When the local binding instance at the responding SOAP node starts to receive an inbound request message, it (logically) instantiates a message exchange context. **Table 10** describes the properties that the binding initializes as part of the context's instantiation.

Table 10: Instantiation of Message Exchange Context for an inbound request message

Property Name	Property Value	Notes
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName`	"http://www.w3.org/2003/05/soap/mep/soap-response/"	Initialized as early as possible during the life cycle of the message exchange.
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason`	"None"	A relative URI whose base URI is the value of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName`

`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role`	"RespondingSOAPNode"	A relative URI whose base URI is the value of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName` Initialized as early as possible during the life cycle the message exchange.
`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State`	"Init"	A relative URI whose base URI is the value of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role`

When the requesting and responding SOAP nodes transition between states, the local binding instance (logically) updates a number of properties. **Table 11** and **Table 12** describe these updates for the requesting and the responding SOAP nodes, respectively.

Table 11: Requesting SOAP Node State Transitions

CurrentState	Transition Condition	NextState	Action
"Init"	Unconditional	"Requesting"	Initiate request
"Requesting"	Message transmission failure	"Fail"	Set `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason` to "transmissionFailure"

	Start receiving response message	"Receiving"	Set `http://www.w3.org/2003/05/soap/mep/ImmediateSender` to denote the sender of the response message (may differ from the values in `http://www.w3.org/2003/05/soap/mep/ImmediateDestination`). Start making an abstraction of the response message available in `http://www.w3.org/2003/05/soap/mep/InboundMessage` .
"Receiving"	Message exchange failure	"Fail"	Set `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason` to "exchangeFailure"
	Completed receiving response message.	"Success"	

Table 12: Responding SOAP Node State Transitions

CurrentState	Transition Condition	NextState	Action
"Init"	Start receiving request	"Receiving"	Set `http://www.w3.org/2003/05/soap/mep/ImmediateSender` to denote the sender of the request message (if determinable). Pass control of message exchange context to SOAP processor.
"Receiving"	Message reception failure	"Fail"	Set `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason` to "receptionFailure".

	Start of response message available in `http://www.w3.org/2003/05/soap/mep/OutboundMessage`	"Sending"	Initiate transmission of response message abstracted in `http://www.w3.org/2003/05/soap/mep/OutboundMessage` .
"Sending"	Message exchange failure	"Fail"	Set `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/FailureReason` to "exchangeFailure".
	Completed sending response message.	"Success"	

6.3.4 Fault Handling

During the operation of the SOAP Response MEP, the participating SOAP nodes may generate SOAP faults.

If a SOAP fault is generated by the responding SOAP node while it is in the "Receiving" state, the SOAP fault is made available in `http://www.w3.org/2003/05/soap/mep/OutboundMessage` and the state machine transitions to the "Sending" state.

This MEP makes no claims about the disposition or handling of SOAP faults generated by the requesting SOAP node during any processing of the response message that follows the "Success" state in the requesting SOAP node's state transition table (see **Table 11**).

6.4 SOAP Web Method Feature

This section defines the "SOAP Web Method Feature".

6.4.1 SOAP Feature Name

The SOAP Web Method feature is identified by the URI (see SOAP 1.2 Part 1 [SOAP Part 1] SOAP Features):

- "http://www.w3.org/2003/05/soap/features/web-method/"

6.4.2 Description

Underlying protocols designed for use on the World Wide Web provide for manipulation of resources using a small set of Web methods such as GET, PUT, POST, and DELETE. These methods are formally defined in the HTTP specification

[RFC 2616], but other underlying protocols might also support them. Bindings to HTTP or such other protocols SHOULD use the SOAP Web Method feature to give applications control over the Web methods to be used when sending a SOAP message.

Bindings supporting this feature SHOULD use the appropriate embodiment of that method if provided by the underlying protocol; for example, the HTTP binding provided with this specification represents the "GET" Web method as an HTTP GET request, and the "POST" method as an HTTP POST request (see **7. SOAP HTTP Binding**). Bindings supporting this feature SHOULD provide to the receiving node indication of the Web method used for transmission.

The SOAP Web Method feature MAY be implemented by bindings to underlying transports that have no preferred embodiment of particular Web methods (e.g. do not distinguish GET from POST). Such bindings SHOULD provide to the receiving node indication of the Web method used for transmission, but need take no other action in support of the feature.

6.4.3 SOAP Web Method Feature State Machine

The SOAP Web Method feature defines a single property, which is described in **Table 13**.

Table 13: Property definition for the SOAP Web Method feature

Property Name	Property Description	Property Type
`http://www.w3.org/2003/05/soap/features/web-method/Method`	One of "GET", "POST", "PUT", "DELETE" (or others which may subsequently be added to the repertoire of Web methods.)	Not specified

This specification provides for the use of the SOAP Web Method feature in conjunction with the **6.2 SOAP Request-Response Message Exchange Pattern** and **6.3 SOAP Response Message Exchange Pattern** message exchange patterns. This feature MAY be used with other MEPs if and only if provided for in the specifications of those MEPs.

A node sending a request message MUST provide a value for the `http://www.w3.org/2003/05/soap/features/web-method/Method` property. A protocol binding supporting this feature SHOULD set the value of the `http://www.w3.org/2003/05/soap/features/web-method/Method` property at the receiving node to match that provided by the sender; the means of transmission for the method property is binding-specific.

A responding node SHOULD respond in a manner consistent with the Web method requested (e.g. a "GET" should result in retrieval of a representation of the

identified resource) or SHOULD fault in an application-specific manner if the Web method cannot be supported.

Bindings implementing this feature MUST employ a Message Exchange Pattern with semantics that are compatible with the Web method selected. For example, the SOAP Response Message Exchange Pattern (see **6.3 SOAP Response Message Exchange Pattern**) is compatible with GET.

6.5 SOAP Action Feature

This section defines the "SOAP Action Feature".

6.5.1 SOAP Feature Name

The SOAP Action feature is identified by the URI (see SOAP 1.2 Part 1 [SOAP Part 1] SOAP Features):

- "http://www.w3.org/2003/05/soap/features/action/"

6.5.2 Description

Many SOAP 1.2 underlying protocol bindings will likely utilize the "application/soap +xml" media type (described in **A. The application/soap+xml Media Type**) to transmit XML serializations of SOAP messages. The media type specifies an optional `action` parameter (see **A.3 The action Parameter**), which can be used to optimize dispatch or routing, among other things. The Action Feature specifies well-known URIs to indicate support for the `action` parameter in bindings which use MIME, and also to refer to value of the parameter itself.

6.5.3 SOAP Action Feature State Machine

The SOAP Action feature defines a single property, which is described in **Table 14**.

Table 14: Property definition for the SOAP Action feature

Property Name	Property Type
`http://www.w3.org/2003/05/soap/features/action/Action`	`xsd:anyURI`

If the `http://www.w3.org/2003/05/soap/features/action/Action` property has a value at a SOAP sender utilizing a binding supporting this feature, the sender MUST use the property value as the value of the `action` parameter in the media type designator.

Conversely, if a value arrives in the `action` parameter of the media type

designator at a SOAP receiver, the receiver MUST make that value available as the value of the `http://www.w3.org/2003/05/soap/features/action/Action` property.

7. SOAP HTTP Binding

7.1 Introduction

The SOAP HTTP Binding provides a binding of SOAP to HTTP. The binding conforms to the SOAP Protocol Binding Framework (see SOAP 1.2 Part 1 [SOAP Part 1] SOAP Protocol Binding Framework) and supports the message exchange patterns and features described in **6. SOAP-Supplied Message Exchange Patterns and Features**.

7.1.1 Optionality

The SOAP HTTP Binding is optional and SOAP nodes are NOT required to implement it. A SOAP node that correctly and completely implements the SOAP HTTP Binding may to be said to "conform to the SOAP 1.2 HTTP Binding."

The SOAP version 1.2 specification does not preclude development of other bindings to HTTP or bindings to other protocols, but communication with nodes using such other bindings is not a goal. Note that other bindings of SOAP to HTTP MAY be written to provide support for SOAP Message exchange patterns other than **6.2 SOAP Request-Response Message Exchange Pattern** or the **6.3 SOAP Response Message Exchange Pattern**. Such alternate bindings MAY therefore make use of HTTP features and status codes not required for this binding. For example, another binding might provide for a 202 or 204 HTTP response status to be returned in response to an HTTP POST or PUT (e.g. a one-way "push" MEP with confirmation).

7.1.2 Use of HTTP

The SOAP HTTP binding defines a base URI according to the rules in HTTP/1.1 [RFC 2616]. I.e. the base URI is the HTTP Request-URI or the value of the HTTP Content-Location header field.

This binding of SOAP to HTTP is intended to make appropriate use of HTTP as an application protocol. For example, successful responses are sent with status code 200, and failures are indicated as 4XX or 5XX. This binding is not intended to fully exploit the features of HTTP, but rather to use HTTP specifically for the purpose of communicating with other SOAP nodes implementing the same binding. Therefore, this HTTP binding for SOAP does not specify the use and/or meaning of all possible HTTP methods, header fields and status responses. It specifies only those which are pertinent to the **6.2 SOAP Request-Response Message Exchange Pattern** or the **6.3 SOAP Response Message Exchange Pattern**, or which are likely to be

introduced by HTTP mechanisms (such as proxies) acting between the SOAP nodes.

Certain optional features provided by this binding depend on capabilities provided by HTTP/1.1, for example content negotiation. Implementations SHOULD thus use HTTP/1.1 [RFC 2616] (or later compatible versions that share the same major version number). Implementations MAY also be deployed using HTTP/1.0, although in this case certain optional binding features may not be provided.

Note:

SOAP HTTP Binding implementations need to account for the fact that HTTP/1.0 intermediaries (which may or may not also be SOAP intermediaries) may alter the representation of SOAP messages, even in situations where both the initial SOAP sender and ultimate SOAP receiver use HTTP/1.1.

7.1.3 Interoperability with non-SOAP HTTP Implementations

Particularly when used with the **6.3 SOAP Response Message Exchange Pattern**, the HTTP messages produced by this binding are likely to be indistinguishable from those produced by non-SOAP implementations performing similar operations. Accordingly, some degree of interoperation can be made possible between SOAP nodes and other HTTP implementations when using this binding. For example, a conventional Web server (i.e. one not written specifically to conform to this specification) might be used to respond to SOAP-initiated HTTP GET's with representations of `Content-Type` "application/soap+xml". Such interoperation is not a normative feature of this specification.

Even though HTTP often is used on the well-known TCP port 80, the use of HTTP is not limited to that port. As a result, it is possible to have a dedicated HTTP server for handling SOAP processing on a distinct TCP port. Alternatively, it is possible to use a separate virtual host for dealing with SOAP processing. Such configuration, however, is a matter of convenience and is not a requirement of this specification (see SOAP 1.2 Part 1 [SOAP Part 1] Binding to Application-Specific Protocols).

7.1.4 HTTP Media-Type

Conforming implementations of this binding:

1. MUST be capable of sending and receiving messages serialized using media type "application/soap+xml" whose proper use and parameters are described in **A. The application/soap+xml Media Type**.
2. MAY send requests and responses using other media types providing that such media types provide for at least the transfer of SOAP XML Infoset.
3. MAY, when sending requests, provide an HTTP Accept header field. This header field:

- SHOULD indicate an ability to accept at minimum "application/soap +xml".
- MAY additionally indicate willingness to accept other media types that satisfy 2 above.

7.2 Binding Name

This binding is identified by the URI (see SOAP 1.2 Part 1 [SOAP Part 1] SOAP Protocol Binding Framework):

- "http://www.w3.org/2003/05/soap/bindings/HTTP/"

7.3 Supported Message Exchange Patterns

An implementation of the SOAP HTTP Binding MUST support the following message exchange patterns (MEPs):

- "http://www.w3.org/2003/05/soap/mep/request-response/" (see **6.2 SOAP Request-Response Message Exchange Pattern**)
- "http://www.w3.org/2003/05/soap/mep/soap-response/" (see **6.3 SOAP Response Message Exchange Pattern**)

7.4 Supported Features

An implementation of the SOAP HTTP Binding MUST support the following features:

- "http://www.w3.org/2003/05/soap/features/web-method/" (see **6.4 SOAP Web Method Feature**)
- "http://www.w3.org/2003/05/soap/features/action/" (see **6.5 SOAP Action Feature**)

The possible values of `http://www.w3.org/2003/05/soap/features/web-method/Method` property are restricted in this HTTP binding according to the MEP in use (as present in `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName`):

Table 15: Possible values of the Web-Method Method property

`http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName`	**`http://www.w3.org/2003/05/soap/features/web-method/Method`**
"http://www.w3.org/2003/05/soap/mep/request-response/"	"POST"

"http://www.w3.org/2003/05/soap/mep/soap-response/"	"GET"

Note:

other SOAP Version 1.2 bindings to HTTP may permit other combinations of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/ExchangePatternName` and `http://www.w3.org/2003/05/soap/features/web-method/Method` .

7.5 MEP Operation

For binding instances conforming to this specification:

- A SOAP node instantiated at an HTTP client may assume the role (i.e. the property `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role`) of "RequestingSOAPNode".
- A SOAP node instantiated at an HTTP server may assume the role (i.e. the property `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role`) of "RespondingSOAPNode".

The remainder of this section describes the MEP state machine and its relation to the HTTP protocol. In the state tables below, the states are defined as values of the property `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/State` (see **6.2 SOAP Request-Response Message Exchange Pattern** and **6.3 SOAP Response Message Exchange Pattern**), and are of type `xs:anyURI` . For brevity, relative URIs are used, the base URI being the value of `http://www.w3.org/2003/05/soap/bindingFramework/ExchangeContext/Role` .

The message exchange pattern in use is indicated by the HTTP method used in the request. HTTP GET corresponds to the SOAP-Response MEP, HTTP POST corresponds to the SOAP Request-Response MEP.

7.5.1 Behavior of Requesting SOAP Node

The overall flow of the behavior of a requesting SOAP node follows a state machine description consistent with either **6.2 SOAP Request-Response Message Exchange Pattern** or **6.3 SOAP Response Message Exchange Pattern** (differences are indicated as necessary.) This binding supports streaming and, as a result, requesting SOAP nodes MUST avoid deadlock by accepting and if necessary processing SOAP response information while the SOAP request is being transmitted (see **6.2.3 State Machine Description**). The following subsections describe each state in detail.

7.5.1.1 Init

In the "Init" state, a HTTP request is formulated according to **Table 16** and transmission of the request is initiated.

Table 16: HTTP Request Fields

Field	Value
HTTP Method	According to the `http://www.w3.org/2003/05/soap/features/web-method/Method` property. POST and GET are the only values supported by this binding.
Request URI	The value of the URI carried in the `http://www.w3.org/2003/05/soap/mep/ImmediateDestination` property of the message exchange context.
Content-Type header field	The media type of the request entity body (if present) otherwise, omitted (see **7.1 Introduction** for a description of permissible media types). If the SOAP envelope infoset in the `http://www.w3.org/2003/05/soap/mep/OutboundMessage` property is null, then the `Content-Type` header field MAY be omitted.
action parameter	According to the value of the `http://www.w3.org/2003/05/soap/features/action/Action` property.
Accept header field (optional)	List of media types that are acceptable in response to the request message.
Additional header fields	Generated in accordance with the rules for the binding specific expression of any optional features in use for this message exchange. For example, a Content-Encoding header field (see HTTP [RFC 2616], section 14.11) may be used to express an optional compression feature.
HTTP entity body	SOAP message serialized according to the rules for carrying SOAP messages in the media type given by the Content-Type header field. Rules for carrying SOAP messages in media type "application/soap+xml" are given in **A. The application/soap+xml Media Type**. If the SOAP envelope infoset in the `http://www.w3.org/2003/05/soap/mep/OutboundMessage` property is null, the entity body is omitted

7.5.1.2 Requesting

In the "Requesting" state, sending of the request continues while waiting for the start of the response message. **Table 17** details the transitions that take place when a requesting SOAP node receives an HTTP status line and response header fields. For some status codes there is a choice of possible next state. In cases where "Fail" is one of the choices, the transition is dependent on whether a SOAP message is present in the HTTP response. If a SOAP message is present, the next

state is "Sending+Receiving" or "Receiving", otherwise the next state is "Fail". The choice between "Sending+Receiving" and "Receiving" depends of the MEP in use: "Sending+Receiving" is the next state for Request-Response while "Receiving" is the next state for SOAP-Response.

Table 17: HTTP status code dependent transitions

Status Code	Reason phrase	Significance/Action	NextState
2xx	Successful		
200	OK	The response message follows in HTTP response entity body. Start making an abstraction of the response message available in `http://www.w3.org/2003/05/soap/mep/InboundMessage` .	"Sending +Receiving" or "Receiving"
3xx	Redirection	The requested resource has moved and the HTTP request SHOULD be retried using the URI carried in the associated Location header field as the new value for the `http://www.w3.org/2003/05/soap/mep/ImmediateDestination` property.	"Init"
4xx	Client Error		
400	Bad Request	Indicates a problem with the received HTTP request message.	"Sending +Receiving", "Receiving" or "Fail"
401	Unauthorized	Indicates that the HTTP request requires authorization. If the simple authentication feature is unavailable or the operation of simple authentication ultimately fails, then the message exchange is regarded as having completed unsuccessfully.	"Requesting" or "Fail"
405	Method not allowed	Indicates that the peer HTTP server does not support the requested HTTP method at the given request URI. The message exchange is regarded as having completed unsuccessfully.	"Fail"

415	Unsupported Media Type	Indicates that the peer HTTP server does not support the Content-type used to encode the request message. The message exchange is regarded as having completed unsuccessfully.	"Fail"
5xx	Server Error		
500	Internal Server Error	Indicates a server problem or a problem with the received request	"Sending +Receiving", "Receiving" or "Fail"

Table 17 refers to some but not all of the existing HTTP/1.1 [RFC 2616] status codes. In addition to these status codes, HTTP provides an open-ended mechanism for supporting status codes defined by HTTP extensions (see RFC 2817 [RFC 2817] for a registration mechanism for new status codes). HTTP status codes are divided into status code classes as described in HTTP [RFC 2616], section 6.1.1. The SOAP HTTP binding follows the rules of any HTTP application which means that an implementation of the SOAP HTTP binding must understand the class of any status code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 status code of that class, with the exception that an unrecognized response must not be cached.

Note:

There may be elements in the HTTP infrastructure configured to modify HTTP response entity bodies for 4xx and 5xx status code responses. For example, some HTTP origin servers have such a feature as a configuration option. This behavior may interfere with the use of 4xx and 5xx status code responses carrying SOAP fault messages in HTTP and it is recommended that such behavior is disabled for resources accepting SOAP/HTTP requests. If the rewriting behavior cannot be disabled, SOAP/HTTP cannot be used in such configurations.

7.5.1.3 Sending+Receiving

In the "Sending+Receiving" state (**6.2 SOAP Request-Response Message Exchange Pattern** only), the transmission of the request message and receiving of the response message is completed. The response message is assumed to contain a SOAP envelope serialized according to the rules for carrying SOAP messages in the media type given in the Content-Type header field.

The response MAY be of content type other than "application/soap+xml". Such usage is considered non-normative, and accordingly is not modeled in the state machine. Interpretation of such responses is at the discretion of the receiver.

7.5.1.4 Receiving

In the "Receiving" state (**6.3 SOAP Response Message Exchange Pattern** only), receiving of the response message is completed. The response message is assumed to contain a SOAP envelope serialized according to the rules for carrying SOAP messages in the media type given in the Content-Type header field.

The response MAY be of content type other than "application/soap+xml". Such a result is particularly likely when a SOAP request sent with a `http://www.w3.org/2003/05/soap/features/web-method/Method` of "GET" is directed (intentionally or otherwise) to a non-SOAP HTTP server. Such usage is considered non-normative, and accordingly is not modeled in the state machine. Interpretation of such responses is at the discretion of the receiver.

7.5.1.5 Success and Fail

"Success" and "Fail" are the terminal states of the Request-Response and SOAP-Response MEPs. Control over the message exchange context returns to the local SOAP node.

7.5.2 Behavior of Responding SOAP Node

The overall flow of the behavior of a responding SOAP node follows a state machine description consistent with either **6.2 SOAP Request-Response Message Exchange Pattern** or **6.3 SOAP Response Message Exchange Pattern** (differences are indicated as necessary). The following subsections describe each state in detail.

7.5.2.1 Init

In the "Init" state, the binding waits for the start of an inbound request message. **Table 18** describes the errors that a responding SOAP node might generate while in the "Init" state. In this state no SOAP message has been received, therefore the SOAP node cannot generate a SOAP fault.

Table 18: Errors generated in the Init state

Problem with Message	HTTP Status Code	HTTP Reason Phrase (informative)
Malformed Request Message	400	Bad request
HTTP Method is neither POST nor GET	405	Method Not Allowed
Unsupported message encapsulation method	415	Unsupported Media

7.5.2.2 Receiving

In the "Receiving" state, the binding receives the request and any associated message and waits for the start of a response message to be available. **Table 19** describes the HTTP response header fields generated by the responding SOAP node. **Table 20** describes the HTTP status codes associated with SOAP faults that can be generated by the responding SOAP node.

Table 19: HTTP Response Headers Fields

Field	Value
Status line	200, or set according to **Table 20** if a SOAP fault was generated.
Content-Type header field	The media type of the response body, see **7.1 Introduction** for a description of permissible media types.
Additional header fields	Generated in accordance with the rules for the binding specific expression of any optional features in use for this message exchange. For example, a Content-Encoding header field (see HTTP [RFC 2616], section 14.11) may be used to express an optional compression feature.
HTTP Entity Body	SOAP message serialized according to the rules for carrying SOAP messages in the media type given by the Content-Type header field. Rules for carrying SOAP messages in "application/soap+xml" are given in **A. The application/soap+xml Media Type**.

Table 20: SOAP Fault to HTTP Status Mapping

SOAP Fault	HTTP Status Code	HTTP Reason Phrase (informative)
env:VersionMismatch	500	Internal server error
env:MustUnderstand	500	Internal server error
env:Sender	400	Bad request
env:Receiver	500	Internal server error
env:DataEncodingUnknown	500	Internal server error

7.5.2.3 Receiving+Sending

In the "Receiving+Sending" state (**6.2 SOAP Request-Response Message Exchange Pattern** only) the binding completes receiving of the request message and transmission of the response message.

7.5.2.4 Sending

In the "Sending" state (**6.3 SOAP Response Message Exchange Pattern** only) the binding completes transmission of the response message.

7.5.2.5 Success and Fail

"Success" and "Fail" are the terminal states for the Request-Response and SOAP-Response MEPs. From the point-of-view of the local node this message exchange has completed.

7.6 Security Considerations

The SOAP HTTP Binding (see **7. SOAP HTTP Binding**) can be considered as an extension of the HTTP application protocol. As such, all of the security considerations identified and described in section 15 of the HTTP specification [RFC 2616] apply to the SOAP HTTP Binding in addition to those described in SOAP 1.2 Part 1 [SOAP Part 1] Security Considerations. Implementors of the SOAP HTTP Binding should carefully review this material.

8. References

8.1 Normative References

[SOAP Part 1]
W3C Proposed Recommendation "SOAP Version 1.2 Part 1: Messaging Framework", Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part1-20030624.)

[RFC 2616]
IETF "RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, T. Berners-Lee, January 1997. (See http://www.ietf.org/rfc/rfc2616.txt.)

[RFC 2119]
IETF "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997. (See http://www.ietf.org/rfc/rfc2119.txt.)

[XML Schema Part 1]
W3C Recommendation "XML Schema Part 1: Structures", Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, 2 May 2001. (See http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.)

[XML Schema Part 2]
W3C Recommendation "XML Schema Part 2: Datatypes", Paul V. Biron, Ashok Malhotra, 2 May 2001. (See http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.)

[RFC 2396]

IETF "RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, R. Fielding, L. Masinter, August 1998. (See http://www.ietf.org/rfc/rfc2396.txt.)

[Namespaces in XML]
W3C Recommendation "Namespaces in XML", Tim Bray, Dave Hollander, Andrew Layman, 14 January 1999. (See http://www.w3.org/TR/1999/REC-xml-names-19990114/.)

[XML 1.0]
W3C Recommendation "Extensible Markup Language (XML) 1.0 (Second Edition)", Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, 6 October 2000. (See http://www.w3.org/TR/2000/REC-xml-20001006.)

[XML InfoSet]
W3C Recommendation "XML Information Set", John Cowan, Richard Tobin, 24 October 2001. (See http://www.w3.org/TR/2001/REC-xml-infoset-20011024/.)

[RFC 3023]
IETF "RFC 3023: XML Media Types", M. Murata, S. St. Laurent, D. Kohn, July 1998. (See http://www.ietf.org/rfc/rfc3023.txt.)

8.2 Informative References

[SOAP Part 0]
W3C Proposed Recommendation "SOAP Version 1.2 Part 0: Primer", Nilo Mitra, 24 June 2003 (See http://www.w3.org/TR/2003/REC-soap12-part0-20030624.)

[SOAP MediaType]
IETF Internet Draft "The 'application/soap+xml' media type", M. Baker, M. Nottingham, "draft-baker-soap-media-reg-03.txt" May 29, 2003. Note that this Internet Draft expires in November 2003. (See http://www.ietf.org/internet-drafts/draft-baker-soap-media-reg-03.txt.)

[XMLP Comments]
XML Protocol Comments Archive (See http://lists.w3.org/Archives/Public/xmlp-comments/.)

[XMLP Dist-App]
XML Protocol Discussion Archive (See http://lists.w3.org/Archives/Public/xml-dist-app/.)

[XMLP Charter]
XML Protocol Charter (See http://www.w3.org/2000/09/XML-Protocol-Charter.)

[RFC 2045]
IETF "RFC2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed, N. Borenstein, November 1996. (See http://www.ietf.org/rfc/rfc2045.txt.)

[RFC 2026]
IETF "RFC 2026: The Internet Standards Process -- Revision 3", section 4.2.3, S. Bradner, October 1996. (See http://www.ietf.org/rfc/rfc2026.txt.)

[RFC 2817]
IETF "RFC 2817: Upgrading to TLS Within HTTP/1.1", R. Khare, S. Lawrence, May 2000. (See http://www.ietf.org/rfc/rfc2817.txt.)

[CharMod]
W3C Working Draft "Character Model for the World Wide Web 1.0", Martin J. Durst, Francois Yergeau, Richard Ishida, Misha Wolf, Asmus Freytag, Tex Texin, 30 April 2002. (See http://www.w3.org/TR/charmod/.)

A. The "application/soap+xml" Media Type

This appendix defines the "application/soap+xml" media type which can be used to describe SOAP 1.2 messages serialized as XML.

Note:

At the time of this document's publication, the process for registering this media type with IANA was underway using the [SOAP MediaType] Internet Draft. A future version of the SOAP specification may reference the media type in the IANA registry.

A.1 Registration

MIME media type name:

application

MIME subtype name:

soap+xml

Required parameters:

none

Optional parameters:

charset

This parameter has identical semantics to the charset parameter of the "application/xml" media type as specified in RFC 3023 [RFC 3023].

action

See section **A.3 The action Parameter**.

Encoding considerations:

Identical to those of "application/xml" as described in RFC 3023 [RFC 3023], section 3.2, as applied to the SOAP envelope infoset.

Security considerations:

See section **A.2 Security Considerations**.

Interoperability considerations:

There are no known interoperability issues.

Published specification:

This document (SOAP 1.2 Part 2) and SOAP 1.2 Part 1 [SOAP Part 1].

Applications which use this media type:

No known applications currently use this media type.

Additional information:

File extension:

SOAP messages are not required or expected to be stored as files.

Fragment identifiers:

Identical to that of "application/xml" as described in RFC 3023 [RFC 3023], section 5.

Base URI:

As specified in RFC 3023 [RFC 3023], section 6. Also see SOAP 1.2 Part 1 [SOAP Part 1], section Use of URIs in SOAP.

Macintosh File Type code:

TEXT

Person and email address to contact for further information:

Mark Baker <mbaker@idokorro.com>

Intended usage:

COMMON

Author/Change controller:

The SOAP 1.2 specification set is a work product of the World Wide Web Consortium's XML Protocol Working Group. The W3C has change control over these specifications.

A.2 Security Considerations

Because SOAP can carry application defined data whose semantics is independent from that of any MIME wrapper (or context within which the MIME wrapper is used), one should not expect to be able to understand the semantics of the SOAP message based on the semantics of the MIME wrapper alone. Therefore, whenever using the "application/soap+xml" media type, it is strongly RECOMMENDED that the security implications of the context within which the SOAP message is used is fully understood. The security implications are likely to involve both the specific SOAP binding to an underlying protocol as well as the application-defined semantics of the data carried in the SOAP message (though one must be careful

when doing this, as discussed in SOAP 1.2 Part 1 [SOAP Part 1], section Binding to Application-Specific Protocols.

Also, see SOAP 1.2 Part 1 [SOAP Part 1], the entire section Security Considerations.

In addition, as this media type uses the "+xml" convention, it shares the same security considerations as described in RFC 3023 [RFC 3023], section 10.

A.3 The `action` Parameter

This optional parameter can be used to specify the URI that identifies the intent of the message. In SOAP 1.2, it serves a similar purpose as the `SOAPAction` HTTP header field did in SOAP 1.1. Namely, its value identifies the intent of the message.

The value of the action parameter is an absolute URI-reference as defined by RFC 2396 [RFC 2396]. SOAP places no restrictions on the specificity of the URI or that it is resolvable.

Although the purpose of the action parameter is to indicate the intent of the SOAP message there is no mechanism for automatically computing the value based on the SOAP envelope. In other words, the value has to be determined out of band.

It is recommended that the same value be used to identify sets of message types that are logically connected in some manner, for example part of the same "service". It is strongly RECOMMENDED that the URI be globally unique and stable over time.

The presence and content of the action parameter MAY be used by servers such as firewalls to appropriately filter SOAP messages and it may be used by servers to facilitate dispatching of SOAP messages to internal message handlers etc. It SHOULD NOT be used as an insecure form of access authorization.

Use of the action parameter is OPTIONAL. SOAP Receivers MAY use it as a hint to optimize processing, but SHOULD NOT require its presence in order to operate.

B. Mapping Application Defined Names to XML Names

This appendix details an algorithm for taking an application defined name, such as the name of a variable or field in a programming language, and mapping it to the Unicode characters that are legal in the names of XML elements and attributes as defined in Namespace in XML [Namespaces in XML]

Hex Digits

```
[5]  hexDigit  ::=  [0-9A-F]
```

B.1 Rules for Mapping Application Defined Names to XML Names

1. An XML Name has two parts: *Prefix* and *LocalPart*. Let *Prefix* be determined per the rules and constraints specified in Namespaces in XML [Namespaces in XML].
2. Let *T* be a name in an application, represented as a sequence of characters encoded in a particular character encoding.

3. Let `M` be the implementation-defined function for transcoding of the characters used in the application defined name to an equivalent string of Unicode characters.

   Note:

   Ideally, if this transcoding is from a non-Unicode encoding, it should be both reversible and Unicode Form C normalizing (that is, combining sequences will be in the prescribed canonical order). It should be noted that some transcodings cannot be perfectly reversible and that Normalization Form C (NFC) normalization may alter the original sequence in a few cases (see Character Model for the World Wide Web [CharMod]). To ensure that matching names continue to match after mapping, Unicode sequences should be normalized using Unicode Normalization Form C.

   Note:

   This transcoding is explicitly to Unicode scalar values ("code points") and not to any particular character encoding scheme of Unicode, such as UTF-8 or UTF-16.

   Note:

   Note: Properly formed surrogate pair sequences must be converted to their respective scalar values ("code points") [That is, the sequence U+D800 U+DC00 should be transcoded to the character U+10000]. If the transcoding begins with a Unicode encoding, non-conforming (non-shortest form) UTF-8 and UTF-16 sequences must be converted to their respective scalar values.

   Note:

   The number of characters in *T* is not necessarily the same as the number of characters in `M`, because transcoding may be one-to-many or many-to-one. The details of transcoding may be implementation-defined. There may be (very rarely) cases where there is no equivalent

Unicode representation for *T*; such cases are not covered here.

4. Let *C* be the sequence of Unicode scalar values (characters) represented by `M(T)`

5. Let *N* be the number of characters in *C*. Let C_1, C_2, ..., C_N be the characters of *C*, in order from most to least significant (logical order).

6. For each *i* between 1 (one) and *N*, let X_i be the Unicode character string defined by the following rules:

   Case:

   1. If C_i is undefined (that is, some character or sequence of characters as defined in the application's character sequence *T* contains no mapping to Unicode), then X_i is implementation-defined.

   2. If *i*<=*N*-1 and C_i is "_" (U+005F LOW LINE) and C_{i+1} is "x" (U+0078 LATIN SMALL LETTER X), then let X_i be "_x005F_".

   3. If *i*=1, and *N*>=3, and C_1 is "x" (U+0078 LATIN SMALL LETTER X) or "X" (U+0058 LATIN CAPITAL LETTER X), and C_2 is "m" (U+006D LATIN SMALL LETTER M) or "M" (U+004D LATIN CAPITAL LETTER M), and C_3 is "l" (U+006C LATIN SMALL LETTER L) or "L" (U+004C LATIN CAPITAL LETTER L) (in other words, a string three letters or longer starting with the text "xml" or any re-capitalization thereof), then if C_1 is "x" (U+0078 LATIN SMALL LETTER X) then let X_1 be "_x0078_"; otherwise, if C_1 is "X" (U+0058 LATIN CAPITAL LETTER X) then let X_1 be "_x0058_".

   4. If C_i is not a valid XML NCName character (see Namespaces in XML [Namespaces in XML]) or if *i*=1 (one) and C_1 is not a valid first character of an XML NCName then:

      Let U_1, U_2, ... , U_6 be the six hex digits **[PROD: 5]** such that C_i is "U+" U_1 U_2 ... U_6 in the Unicode scalar value.

      Case:

      1. If U_1=0, U_2=0, U_3=0, and U_4=0, then let X_i="_x" U_5 U_6 "_".

         This case implies that C_i is a character in the Basic Multilingual Plane (Plane 0) of Unicode and can be wholly represented by a single UTF-16 code point sequence U+U_5U_6.

2. Otherwise, let X_i be "_x" U_1 U_2 U_3 U_4 U_5 U_6 "_".

5. Otherwise, let X_i be M_i. That is, any character in X that is a valid character in an XML NCName is simply copied.

7. Let *LocalPart* be the character string concatenation of X_1, X_2, ... , X_N in order from most to least significant.

8. Let XML Name be the QName per Namespaces in XML [Namespaces in XML]

B.2 Examples

```
Hello world -> Hello_x0020_world
Hello_xorld -> Hello_x005F_xorld
Helloworld_ -> Helloworld_

          x -> x
        xml -> _x0078_ml
       -xml -> _x002D_xml
       x-ml -> x-ml

     Ælfred -> Ælfred
   •γνωστος -> •γνωστος
••••        -> _x1709__x1705__x170E__x1708_
•••         -> _x13D9__x13DA__x13A5_
```

C. Using W3C XML Schema with SOAP Encoding (Non-Normative)

As noted in **3.1.4 Computing the Type Name Property** SOAP graph nodes are labeled with type names, but conforming processors are not required to perform validation of encoded SOAP messages.

These sections describe techniques that can be used when validation with W3C XML schemas is desired for use by SOAP applications. Any errors or faults resulting from such validation are beyond those covered by the normative recommendation; from the perspective of SOAP, such faults are considered to be application-level failures.

C.1 Validating Using the Minimum Schema

Although W3C XML schemas are conventionally exchanged in the form of schema

documents (see XML Schema [XML Schema Part 1]), the schema recommendation is build on an abstract definition of schemas, to which all processors need to conform. The schema recommendation provides that all such schemas include definitions for a core set of built in types, such as integers, dates, and so on (see XML Schema [XML Schema Part 1], Built-in Simple Type Definition). Thus, it is possible to discuss validation of a SOAP message against such a minimal schema, which is the one that would result from providing no additional definitions or declarations (i.e. no schema document) to a schema processor.

The minimal schema provides that any well formed XML document will validate, except that where an xsi:type is provided, the type named must be built in, and the corresponding element must be valid per that type. Thus, validation of a SOAP 1.2 message using a minimal schema approximates the behavior of the built-in types of SOAP 1.1.

C.2 Validating Using the SOAP Encoding Schema

Validation against the minimal schema (see **C.1 Validating Using the Minimum Schema**) will not succeed where encoded graph nodes have multiple inbound edges. This is because elements representing such graph nodes will carry `id` *attribute information items* which are not legal on elements of type "xs:string", "xs:integer" etc. The SOAP Encoding of such graphs MAY be validated against the SOAP Encoding schema. In order for the encoding to validate, edge labels, and hence the [local name] and [namespace name] properties of the *element information items*, need to match those defined in the SOAP Encoding schema. Validation of the encoded graph against the SOAP Encoding schema would result in the type name property of the nodes in the graph being assigned the relevant type name.

C.3 Validating Using More Specific Schemas

It may be that schemas could be constructed to describe the encoding of certain graphs. Validation of the encoded graph against such a schema would result in the type name property of the graph nodes being assigned the relevant type name. Such a schema can also supply default or fixed values for one or more of the `itemType` , `arraySize` or `nodeType` *attribute information items*; the values of such defaulted attributes affect the deserialized graph in the same manner as if the attributes had been explicitly supplied in the message. Errors or inconsistencies thus introduced (e.g. if the value of the attribute is erroneous or inappropriate) should be reported as application-level errors; faults from the "http://www.w3.org/2003/05/soap-encoding" namespace should be reported only if the normative parts of this specification are violated.

D. Acknowledgements (Non-Normative)

This document is the work of the W3C XML Protocol Working Group.

Participants in the Working Group are (at the time of writing, and by alphabetical order): Carine Bournez (W3C), Michael Champion (Software AG), Glen Daniels (Macromedia, formerly of Allaire), David Fallside (IBM), Dietmar Gaertner (Software AG), Tony Graham (Sun Microsystems), Martin Gudgin (Microsoft Corporation, formerly of DevelopMentor), Marc Hadley (Sun Microsystems), Gerd Hoelzing (SAP AG), Oisin Hurley (IONA Technologies), John Ibbotson (IBM), Ryuji Inoue (Matsushita Electric), Kazunori Iwasa (Fujitsu Limited), Mario Jeckle (DaimlerChrysler R. & Tech), Mark Jones (AT&T), Anish Karmarkar (Oracle), Jacek Kopecky (Systinet/Idoox), Yves Lafon (W3C), Michah Lerner (AT&T), Noah Mendelsohn (IBM, formerly of Lotus Development), Jeff Mischkinsky (Oracle), Nilo Mitra (Ericsson), Jean-Jacques Moreau (Canon), Masahiko Narita (Fujitsu Limited), Eric Newcomer (IONA Technologies), Mark Nottingham (BEA Systems, formerly of Akamai Technologies), David Orchard (BEA Systems, formerly of Jamcracker), Andreas Riegg (DaimlerChrysler R. & Tech), Hervé Ruellan (Canon), Jeff Schlimmer (Microsoft Corporation), Miroslav Simek (Systinet/Idoox), Pete Wenzel (SeeBeyond), Volker Wiechers (SAP AG).

Previous participants were: Yasser alSafadi (Philips Research), Bill Anderson (Xerox), Vidur Apparao (Netscape), Camilo Arbelaez (WebMethods), Mark Baker (Idokorro Mobile (Planetfred), formerly of Sun Microsystems), Philippe Bedu (EDF (Electricité de France)), Olivier Boudeville (EDF (Electricité de France)), Don Box (Microsoft Corporation, formerly of DevelopMentor), Tom Breuel (Xerox), Dick Brooks (Group 8760), Winston Bumpus (Novell), David Burdett (Commerce One), Charles Campbell (Informix Software), Alex Ceponkus (Bowstreet), David Chappell (Sonic Software), Miles Chaston (Epicentric), David Clay (Oracle), David Cleary (Progress Software), Conleth O'Connell (Vignette), Ugo Corda (Xerox), Paul Cotton (Microsoft Corporation), Fransisco Cubera (IBM), Jim d'Augustine (eXcelon), Ron Daniel (Interwoven), Dug Davis (IBM), Ray Denenberg (Library of Congress), Paul Denning (MITRE), Frank DeRose (Tibco), Mike Dierken (DataChannel), Andrew Eisenberg (Progress Software), Brian Eisenberg (DataChannel), Colleen Evans (Sonic Software), John Evdemon (XMLSolutions), David Ezell (Hewlett-Packard), Eric Fedok (Active Data Exchange), Chris Ferris (Sun Microsystems), Daniela Florescu (Propel), Dan Frantz (BEA Systems), Michael Freeman (Engenia Software), Scott Golubock (Epicentric), Rich Greenfield (Library of Congress), Hugo Haas (W3C), Mark Hale (Interwoven), Randy Hall (Intel), Bjoern Heckel (Epicentric), Erin Hoffman (Tradia), Steve Hole (MessagingDirect Ltd.), Mary Holstege (Calico Commerce), Jim Hughes (Fujitsu Software Corporation), Yin-Leng Husband (Hewlett-Packard, formerly of Compaq), Scott Isaacson (Novell), Murali Janakiraman (Rogue Wave), Eric Jenkins (Engenia Software), Jay Kasi (Commerce One), Jeffrey Kay (Engenia Software), Richard Koo (Vitria Technology Inc.), Alan Kropp (Epicentric), Julian Kumar (Epicentric), Peter Lecuyer (Progress Software), Tony Lee (Vitria Technology Inc.), Amy Lewis (TIBCO), Bob Lojek (Intalio), Henry Lowe (OMG), Brad Lund (Intel), Matthew MacKenzie (XMLGlobal Technologies), Murray Maloney (Commerce One), Richard Martin (Active Data Exchange), Highland Mary Mountain (Intel), Alex Milowski (Lexica), Kevin Mitchell (XMLSolutions), Ed Mooney (Sun Microsystems), Dean Moses (Epicentric), Don Mullen (Tibco), Rekha Nagarajan (Calico Commerce), Raj Nair (Cisco), Mark Needleman (Data Research Associates), Art Nevarez (Novell), Henrik Nielsen (Microsoft Corporation), Kevin Perkins (Compaq), Jags Ramnaryan (BEA Systems),

Vilhelm Rosenqvist (NCR), Marwan Sabbouh (MITRE), Waqar Sadiq (Vitria Technology Inc.), Rich Salz (Zolera), Krishna Sankar (Cisco), George Scott (Tradia), Shane Sesta (Active Data Exchange), Lew Shannon (NCR), John-Paul Sicotte (MessagingDirect Ltd.), Simeon Simeonov (Allaire), Simeon Simeonov (Macromedia), Aaron Skonnard (DevelopMentor), Nick Smilonich (Unisys), Soumitro Tagore (Informix Software), James Tauber (Bowstreet), Lynne Thompson (Unisys), Patrick Thompson (Rogue Wave), Jim Trezzo (Oracle), Asir Vedamuthu (WebMethods), Randy Waldrop (WebMethods), Fred Waskiewicz (OMG), David Webber (XMLGlobal Technologies), Ray Whitmer (Netscape), Stuart Williams (Hewlett-Packard), Yan Xu (DataChannel), Amr Yassin (Philips Research), Susan Yee (Active Data Exchange), Jin Yu (Martsoft).

The people who have contributed to discussions on xml-dist-app@w3.org are also gratefully acknowledged.

Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language

W3C Working Draft 26 March 2004

This version:
http://www.w3.org/TR/2004/WD-wsdl20-20040326
Latest version:
http://www.w3.org/TR/wsdl20
Previous versions:
http://www.w3.org/TR/2003/WD-wsdl20-20031110
Editors:
Roberto Chinnici, Sun Microsystems
Martin Gudgin, Microsoft
Jean-Jacques Moreau, Canon
Jeffrey Schlimmer, Microsoft
Sanjiva Weerawarana, IBM Research

This document is also available in these non-normative formats: postscript, PDF, XML, and plain text.

Abstract

This document describes the Web Services Description Language (WSDL) Version 2.0, an XML language for describing Web services. This specification defines the core language which can be used to describe Web services based on an abstract model of what the service offers. It also defines criteria for a conformant processor of this language.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is a W3C Working Draft of the Web Services Description Language (WSDL) 2.0 document.

A diff-marked version against the previous version of this document is available. For a detailed list of changes since the last publication of this document, please refer to appendix **F. Part 1 Change Log**. A list of open issues against this document is also available.

This document has been produced as part of the W3C Web Services Activity. The authors of this document are the Web Services Description Working Group members.

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Comments on this document are invited and are to be sent to the public www-ws-desc@w3.org mailing list (public archive).

This document has been produced under the 24 January 2002 Current Patent Practice as amended by the W3C Patent Policy Transition Procedure. Patent disclosures relevant to this specification may be found on the Working Group's patent disclosure page. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) with respect to this specification should disclose the information in accordance with section 6 of the W3C Patent Policy.

Short Table of Contents

A. The application/wsdl+xml Media Type
B. Acknowledgements (Non-Normative)
C. URI References for WSDL constructs (Non-Normative)
D. Migrating from WSDL 1.1 to WSDL 2.0 (Non-Normative)
E. Examples of Specifications of Extension Elements for Alternative Schema Language Support. (Non-Normative)
F. Part 1 Change Log (Non-Normative)

Table of Contents

Appendices

---

1. Introduction

Web Services Description Language (WSDL) provides a model and an XML format for describing Web services. WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered.

This specification defines a language for describing the abstract functionality of a service as well as a framework for describing the concrete details of a service description. It also defines criteria for a conformant processor of this language. The *WSDL Version 2.0 Part 2: Message Exchange Patterns* specification [*WSDL 2.0 Message Exchange Patterns*] defines the sequence and cardinality of abstract messages sent or received by an operation. The

WSDL Version 2.0 Part 3: Bindings specification [*WSDL 2.0 Bindings*] defines a language for describing such concrete details for SOAP 1.2 [*SOAP 1.2 Part 1: Messaging Framework*], HTTP [*IETF RFC 2616*] and MIME [*IETF RFC 2045*].

1.1 Web Service

WSDL describes a Web service in two fundamental stages: one abstract and one concrete. Within each stage, the description uses a number of constructs to promote reusability of the description and separate independent design concerns.

At an abstract level, WSDL describes a Web service in terms of the messages it sends and receives; messages are described independent of a specific wire format using a type system, typically XML Schema.

An *operation* associates a message exchange pattern with one or more messages. A *message exchange pattern* identifies the sequence and cardinality of messages sent and/or received as well as who they are logically sent to and/or received from. An *interface* groups together operations without any commitment to transport or wire format.

At a concrete level, a *binding* specifies transport and wire format details for one or more interfaces. An *endpoint* associates a network address with a binding. And finally, a *service* groups together endpoints that implement a common interface.

1.2 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [*IETF RFC 2119*].

This specification uses properties from the XML Information Set [*XML Information Set*]. Such properties are denoted by square brackets, e.g. [namespace name].

This specification uses namespace prefixes throughout; they are listed in Table 1-1. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see [*XML Information Set*]).

Table 1-1. Prefixes and Namespaces used in this specification

Prefix	Namespace	Notes

wsdl	"http://www.w3.org/2004/03/wsdl"	A normative XML Schema [*XML Schema: Structures*], [*XML Schema: Datatypes*] document for the "http://www.w3.org/2004/03/wsdl" namespace can be found at http://www.w3.org/2004/03/wsdl. WSDL documents that do NOT conform to this schema are not valid WSDL documents. WSDL documents that DO conform to this schema and also conform to the other constraints defined in this specification are valid WSDL documents.
wsdli	"http://www.w3.org/2004/03/wsdl-instance"	A normative XML Schema [*XML Schema: Structures*], [*XML Schema: Datatypes*] document for the "http://www.w3.org/2004/03/wsdl-instance" namespace can be found at http://www.w3.org/2004/03/wsdl-instance.
wrpc	"http://www.w3.org/2004/03/wsdl/rpc"	A normative XML Schema [*XML Schema: Structures*], [*XML Schema: Datatypes*] document for the "http://www.w3.org/2004/03/wsdl/rpc" namespace can be found at http://www.w3.org/2004/03/wsdl/rpc. WSDL documents that do NOT conform to this schema are not valid WSDL documents. WSDL documents that DO conform to this schema and also conform to the other constraints defined in this specification are valid WSDL documents.
wsoap12	"http://www.w3.org/2003/11/wsdl/soap12"	Defined by WSDL 2.0: Bindings [*WSDL 2.0 Bindings*].

whttp	"http://www.w3.org/2003/11/wsdl/http"	
xs	"http://www.w3.org/2001/XMLSchema"	Defined in the W3C XML Schema specification [*XML Schema: Structures*], [*XML Schema: Datatypes*].
xsi	"http://www.w3.org/2001/XMLSchema-instance"	

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URIs [*IETF RFC 2396*].

All parts of this specification are normative, with the EXCEPTION of notes, pseudo-schemas, examples, and sections explicitly marked as "Non-Normative". Pseudo-schemas are provided for each component, before the description of the component.

2. Component Model

This section describes the conceptual model for WSDL as a set of components with properties, each aspect of a Web service that WSDL can describe having its own property. In addition an XML Infoset representation for these components is provided, along with a mapping from that representation to the various component properties. How the XML Infoset representation of a given set of WSDL components is constructed is outside the scope of this specification.

2.1 Definitions

2.1.1 The Definitions Component

At the abstract level, the Definitions component is just a container for two categories of components; WSDL components and type system components.

WSDL components are interfaces, bindings and services.

Type system components are element declarations drawn from some type system. They define the [local name], [namespace name], [children] and [attributes] properties of an *element information item*.

The properties of the Definitions component are as follows:

- {interfaces} A set of named interface definitions
- {bindings} A set of named binding definitions
- {services} A set of named service definitions
- {element declarations} A set of named element declarations, each one

isomorphic to a global element declaration as defined by XML Schema

The set of interfaces/binding/services/etc. available in the Definitions component include those that are defined within the component itself and those that are imported and/or included. Note that at the component model level, there is no distinction between directly defined components vs. imported/ included components.

The components directly defined within a single Definitions component are said to belong to the same *target namespace*. The target namespace therefore groups a set of related component definitions and represents an unambiguous name for the intended semantics of the components. The target namespace URI SHOULD point to a human or machine processable document that directly or indirectly defines the intended semantics of those components.

Note that it is RECOMMENDED that the value of the `targetNamespace` *attribute information item* SHOULD be a dereferencible URI and that it resolve to a WSDL document which provides service description information for that namespace.

If a service description is split into multiple documents (which may be combined as needed via **4.1 Including Descriptions**), then the `targetNamespace` *attribute information item* SHOULD resolve to a master document which includes all the WSDL documents needed for that service description. This approach enables the WSDL component designators' fragment identifiers to be properly resolvable.

Imported components have different target namespace values from the Definitions component that is importing them. Thus importing is the mechanism to use components from one namespace in another set of definitions.

Each WSDL or type system component MUST be uniquely identified by its qualified name. That is, if two distinct components of the same kind (Interface, Binding etc.) are in the same target namespace, then their QNames MUST be unique. However, different kinds of components (e.g., an Interface component and a Binding component) MAY have the same QName. Thus, QNames of components must be unique within the space of those components in a given target namespace.

In addition to WSDL components and type system components, additional extension components MAY be added via extensibility **6. Language Extensibility**. Further, additional properties to WSDL and type system components MAY also be added via extensibility.

2.1.2 XML Representation of Definitions Component

```
<definitions
```

```
      targetNamespace="xs:anyURI" >
  <documentation />?
  [ <import /> | <include /> ]*
  <types />?
  [ <interface /> | <binding /> | <service /> ]*
</definitions>
```

WSDL definitions are represented in XML by one or more WSDL Information Sets (Infosets), that is one or more `definitions` *element information item*s. A WSDL Infoset contains representations for a collection of WSDL components which share a common target namespace. A WSDL Infoset which contains one or more `import` *element information item*s **4.2 Importing Descriptions** corresponds to a collection with components drawn from multiple target namespaces.

The targetNamespace URI MUST be an absolute URI (see [*IETF RFC 2396*]).

The `definitions` *element information item* has the following Infoset properties:

- A [local name] of `definitions` .
- A [namespace name] of "http://www.w3.org/2004/03/wsdl".
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `targetNamespace` *attribute information item* as described below in **2.1.2.1 targetNamespace attribute information item**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item*s amongst its [children], in order as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `include` *element information item*s (see **4.1 Including Descriptions**)
     - Zero or more `import` *element information item*s (see **4.2 Importing Descriptions**)
     - Zero or more namespace-qualified *element information item*s. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.

org/2004/03/wsdl".

3. An OPTIONAL `types` *element information item* (see **3. Types**).
4. Zero or more *element information items* from among the following, in any order:
   - `interface` *element information items* (see **2.2.2 XML Representation of Interface Component**).
   - `binding` *element information items* (see **2.9.2 XML Representation of Binding Component**).
   - `service` *element information items* (see **2.13.2 XML Representation of Service Component**).
   - Zero or more namespace-qualified *element information items*. The [namespace name] of such *element information items* MUST NOT be "http://www.w3.org/2004/03/wsdl".

2.1.2.1 `targetNamespace` *attribute information item*

The `targetNamespace` *attribute information item* defines the namespace affiliation of top-level components defined in this `definitions` *element information item*. Interfaces, Bindings and Services are top-level components.

The `targetNamespace` *attribute information item* has the following Infoset properties:

- A [local name] of `targetNamespace`
- A [namespace name] which has no value

The type of the `targetNamespace` *attribute information item* is *xs:anyURI*.

2.1.3 Mapping Definitions' XML Representation to Component Properties

The mapping between the properties of the Definitions component (see **2.1.1 The Definitions Component**) and the XML Representation of the `definitions` *element information item* (see **2.1.2 XML Representation of Definitions Component**) is described in Table 2-1.

Table 2-1. Mapping between Definitions Component Properties and XML Representation

Property	Mapping

{interfaces}	The interface definitions corresponding to all the `interface` *element information item*s in the [children] of the `definitions` *element information item*, if any, plus any included or imported interface definitions (see **4. Modularizing WSDL descriptions**).
{bindings}	The binding definitions corresponding to all the `binding` *element information item*s in the [children] of the `definitions` *element information item*, if any, plus any included or imported binding definitions (see **4. Modularizing WSDL descriptions**).
{services}	The service definitions corresponding to all the `service` *element information item*s in the [children] of the `definitions` *element information item*, if any, plus any included or imported service definitions (see **4. Modularizing WSDL descriptions**).
{element declarations}	The element declaration components corresponding to all the element declarations defined as descendants of the `types` *element information item*, if any, plus any imported element definitions. At a minimum this will include all the global element declarations defined by XML Schema `element` *element information item*s. It MAY also include any definition from some other type system which describes the [local name], [namespace name], [attributes] and [children] properties of an *element information item*.

2.2 Interface

2.2.1 The Interface Component

An Interface component describes sequences of messages that a service sends and/or receives. It does this by grouping related messages into operations. An operation is a sequence of input and output messages, an interface is a set of operations.

An interface can optionally extend one or more other interfaces. In such cases the interface contains the operations of the interfaces it extends, along with any operations it defines. The interfaces a given interface extends MUST NOT themselves extend that interface either directly or indirectly.

Interfaces are named constructs and can be referred to by QName (see **2.17**

QName resolution). For instance, Binding components refer to interfaces in this way.

The properties of the Interface component are as follows:

- {name} An NCName as defined by [*XML Namespaces*].
- {target namespace} A namespace name, as defined in [*XML Namespaces*].
- {extended interfaces} A set of named interface definitions which this interface extends.
- {faults} A set of named interface fault definitions.
- {operations} A set of named interface operation definitions.
- {features} A set of named feature definitions.
- {properties} A set of named property definitions.

For each Interface component in the {interfaces} property of a definitions container, the combination of {name} and {target namespace} properties MUST be unique.

2.2.2 XML Representation of Interface Component

```
<definitions>
  <interface
        name="xs:NCName"
        extends="list of xs:QName"?
        styleDefault="xs:anyURI"? >
    <documentation />?
    [ <fault /> | <operation /> | <feature /> |
<property /> ]*
  </interface>
</definitions>
```

The XML representation for an Interface component is an *element information item* with the following Infoset properties:

- A [local name] of `interface`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.2.2.1 name attribute information item with interface [owner]**.

    - An OPTIONAL `extends` *attribute information item* as described below in **2.2.2.2 extends attribute information item**.
    - An OPTIONAL `styleDefault` *attribute information item* as described below in **2.2.2.3 styleDefault attribute information item**.
    - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item*s amongst its [children], in order, as follows:
    1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
    2. Zero or more *element information item*s from among the following, in any order:
        - Zero or more `fault` *element information item*s **2.3.2 XML Representation of Interface Fault Component**.
        - Zero or more `operation` *element information item*s **2.4.2 XML Representation of Interface Operation Component**.
        - Zero or more `feature` *element information item*s **2.7.2 XML Representation of Feature Component**.
        - Zero or more `property` *element information item*s **2.8.2 XML Representation of Property Component**.
        - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".

2.2.2.1 `name` ***attribute information item with*** `interface` ***[owner]***

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item* forms the QName of the interface.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name`
- A [namespace name] which has no value

The type of the `name` *attribute information item* is *xs:NCName*.

2.2.2.2 `extends` *attribute information item*

The `extends` *attribute information item* lists the interfaces that this interface derives from.

The `extends` *attribute information item* has the following Infoset properties:

- A [local name] of `extends`
- A [namespace name] which has no value

The type of the `extends` *attribute information item* is a list of *xs:QName*.

2.2.2.3 `styleDefault` *attribute information item*

The `styleDefault` *attribute information item* indicates the default style used to construct the {element} properties of {message references} of all operations contained within the [owner] `interface` .

The `styleDefault` *attribute information item* has the following Infoset properties:

- A [local name] of `styleDefault.`
- A [namespace name] which has no value.

The type of the `styleDefault` *attribute information item* is *xs:anyURI*. Moreover, the value of the `styleDefault` *attribute information item*, if present, MUST be an absolute URI (see [*IETF RFC 2396*]).

2.2.3 Mapping Interface's XML Representation to Component Properties

The mapping between the properties of the Interface component (see **2.2.1 The Interface Component**) and the XML Representation of the `interface` *element information item* (see **2.2.2 XML Representation of Interface Component**) is as described in Table 2-2.

Table 2-2. Mapping between Interface Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*
{target namespace}	The actual value of the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item*

{extended interfaces}	The set of interface definitions resolved to by the values in the `extends` *attribute information item* if any, plus the set of interface definitions in the {extended interfaces} property of those interface definitions, otherwise empty.
{faults}	The set of interface fault definitions corresponding to the `fault` *element information item*s in [children], if any, plus the set of interface fault definitions in the {faults} property of the interface definitions in {extended interfaces}, if any.
{operations}	The set of interface operation definitions corresponding to the `operation` *element information item*s in [children], if any, plus the set of interface operation definitions in the {operations} property of the interface definitions in {extended interfaces}, if any.
{features}	The set of feature definitions corresponding to the `feature` *element information item*s in [children], if any, plus the set of feature definitions in the {features} property of the feature definitions in {extended interfaces}, if any.
{properties}	The set of property definitions corresponding to the `property` *element information item*s in [children], if any, plus the set of property definitions in the {properties} property of the property definitions in {extended interfaces}, if any.

Note that, per **2.2.1 The Interface Component**, the Interface components in the {extended interfaces} property of a given Interface component MUST NOT contain that Interface component in any of their {extended interfaces} properties, that is to say, recursive extension of interfaces is disallowed.

2.3 Interface Fault

2.3.1 The Interface Fault Component

An Interface Fault component describes a fault that MAY be occur during execution of an operation of the interface. The Interface Fault component declares a fault by naming it and indicating the content or payload of the fault message. When and how the fault message flows is indicated by the Interface Operation component **2.4 Interface Operation**.

The reason the Interface Fault component is a property of the Interface component is because that provides a convenient mechanism to declare a set of fault message types and then indicate which operations use those types,

thus allowing one to easily indicate that the same fault message type can occur in multiple operations.

The properties of the Interface Fault component are as follows:

- {name} An NCName as defined by [*XML Namespaces*].
- {element} A reference to an XML element declaration in the {element declarations} property of **2.1.1 The Definitions Component**. This element represents the content or "payload" of the fault.

If a non-XML type system is in use (as considered in **3.2 Using Other Schema Languages**) then additional properties would need to be added to the Fault Component (along with extensibility attributes to its XML representation) to allow associating such message types with the message reference.

For each Interface Fault component in the {faults} property of an Interface component, the combination of {name} and {target namespace} properties must be unique.

Interface Fault components are local to Interface components; they cannot be referred to by QName, despite having both {name} and {target namespace} properties. That is, two Interface components sharing the same {target namespace} property but with different {name} properties MAY contain Interface Fault components which share the same {name} property. Thus, the {name} and {target namespace} properties of the Interface Fault components are not sufficient to form the unique identity of an Interface Fault component. To uniquely identify an Interface Fault component one must first identify the Interface component (by QName) and then identify the Interface Fault within that Interface component (by a further QName).

In cases where, due to an interface extending one or more other interfaces, two or more Interface Faults components have the same value for their {name} and {target namespace} properties, then the component models of those Interface Fault components MUST be equivalent (see **2.15 Equivalence of Components**). If the Interface Fault components are equivalent then they are considered to collapse into a single component. It is an error if two Interface Fault components have the same value for their {name} and {target namespace} properties but are not equivalent.

Note that, due to the above rules, if two interfaces that have the same value for their {target namespace} property also have one or more faults that have the same value for their {name} property then those two interfaces cannot both form part of the derivation chain of a derived interface unless those faults are the same fault.

Note:

For the above reason, it is considered good practice to ensure, where necessary, that the {name} property of Interface Fault components

within a namespace are unique, thus allowing such derivation to occur without inadvertent error.

2.3.2 XML Representation of Interface Fault Component

```
<definitions>
  <interface>
    <fault
          name="xs:NCName"
          element="xs:QName"? >
      <documentation />?
    </fault>
  </interface>
</definitions>
```

The XML representation for an Interface Fault component is an *element information item* with the following Infoset properties:

- A [local name] of `fault`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- Two or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.3.2.1 name attribute information item with fault [owner]**.
  - An OPTIONAL `element` *attribute information item* as described below in **2.3.2.2 element attribute information item with fault [owner]**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
  2. Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".

2.3.2.1 `name` *attribute information item with* `fault` *[owner]*

The `name` *attribute information item* identifies a given `fault` *element information item* inside a given `interface` *element information item*.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name`
- A [namespace name] which has no value

The type of the `name` *attribute information item* is *xs:NCName*.

2.3.2.2 `element` *attribute information item with* `fault` *[owner]*

The `element` *attribute information item* refers, by QName, to an element declaration component.

The `element` *attribute information item* has the following Infoset properties:

- A [local name] of `element` .
- A [namespace name] which has no value.

The type of the `element` *attribute information item* is *xs:QName*.

2.3.3 Mapping Interface Fault's XML Representation to Component Properties

The mapping between the properties of the Interface Fault component (see **2.3.1 The Interface Fault Component**) and the XML Representation of the `fault` *element information item* (see **2.3.2 XML Representation of Interface Fault Component**) is as described in Table 2-3.

Table 2-3. Mapping between Interface Fault Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*.
{target namespace}	The actual value of the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item* of the [parent] `interface` *element information item*.

{element}	The element declaration from the {element declarations} property of **2.1.1 The Definitions Component** resolved to by the value of the `element` *attribute information item* if present, otherwise empty. It is an error for the `element` *attribute information item* to have a value and for it to not resolve to a global element declaration from the {element declarations} property of **2.1.1 The Definitions Component**.

2.4 Interface Operation

2.4.1 The Interface Operation Component

An Interface Operation component describes an operation that a given interface supports. An operation is an interaction with the service consisting of a set (ordinary and fault) messages exchanged between the service and the other roles involved in the interaction, in particular the service requestor. The sequencing and cardinality of the messages involved in a particular interaction is governed by the *message exchange pattern* used by the operation (see {message exchange pattern} property).

A message exchange pattern defines placeholders for messages, the participants in the pattern (i.e., the sources and sinks of the messages), and the cardinality and sequencing of messages exchanged by the participants. The message placeholders are associated with specific message types by the operation that uses the pattern by means of message and fault references (see {message references} and {fault references} properties). The service whose operation is using the pattern becomes one of the participants of the pattern. This specification does not define a machine understandable language for defining message exchange patterns, nor does it define any specific patterns. The companion specification, [*WSDL 2.0 Message Exchange Patterns*] defines a set of such patterns and defines identifying URIs any of which MAY be used as the value of the {message exchange pattern} property.

The properties of the Interface Operation component are as follows:

- {name} An NCName as defined by [*XML Namespaces*].
- {target namespace} A namespace name, as defined in [*XML Namespaces*].
- {message exchange pattern} A URI identifying the message exchange pattern used by the operation. This URI MUST be an absolute URI (see [*IETF RFC 2396*]).
- {message references} A set of Message Reference components for the

ordinary messages the operation accepts or sends. (See **2.5 Message Reference**.)

- {fault references} A set of Fault Reference components for the fault messages the operation accepts or sends. (See **2.6 Fault Reference**.)
- {style} A URI identifying the rules that were used to construct the {element} properties of {message references}. (See **2.4.1.1 Operation Style**.) This URI MUST be an absolute URI (see [*IETF RFC 2396*]).
- {safety} A boolean indicating whether the operation is asserted to be safe (as defined in Section 3.5 of [*Web Architecture*]) for users of the described service to invoke. If this property is false or is not set, then no assertion has been made about the safety of the operation, thus the operation MAY or MAY NOT be safe. However, an operation SHOULD be marked safe if it meets the criteria for a safe interaction defined in Section 3.5 of [*Web Architecture*]. The default value of this property is false.
- {features} A set of named feature definitions used by the operation
- {properties} A set of named property definitions used by the operation

For each Interface Operation component in the {operations} property of an Interface component, the combination of {name} and {target namespace} properties MUST be unique.

Interface Operation components are local to Interface components; they cannot be referred to by QName, despite having both {name} and {target namespace} properties. That is, two Interface components sharing the same {target namespace} property but with different {name} properties MAY contain Interface Operation components which share the same {name} property. Thus, the {name} and {target namespace} properties of the Interface Operation components are not sufficient to uniquely identify an Interface Operation component. In order to uniquely identify an Interface Operation component, one must first identify the Interface component (by QName) and then identify the Interface Operation within that Interface component (by a further QName).

In cases where, due to an interface extending one or more other interfaces, two or more Interface Operation components have the same value for their {name} and {target namespace} properties, then the component models of those Interface Operation components MUST be equivalent (see **2.15 Equivalence of Components**). If the Interface Operation components are equivalent then they are considered to collapse into a single component. It is an error if two Interface Operation components have the same value for their {name} and {target namespace} properties but are not equivalent.

Note that, due to the above rules, if two interfaces that have the same value for their {target namespace} property also have one or more operations that

have the same value for their {name} property then those two interfaces cannot both form part of the derivation chain of a derived interface unless those operations are the same operation.

Note:

For the above reason, it is considered good practice to ensure, where necessary, that the {name} property of Interface Operation components within a namespace are unique, thus allowing such derivation to occur without inadvertent error.

2.4.1.1 Operation Style

If the {style} property of an Interface Operation component has a value then that value (a URI) implies the rules that were used to define the {element} properties (or other property which defines the content of the message properties; see **3.2 Using Other Schema Languages**) of *all* the Message Reference components which are members of the {message references} property of that component. Note that the property MAY not have any value. If this property has a given value, then the rules implied by that value (such as rules that govern the schemas) MUST be followed or it is an error.

This specification defines the following pre-defined operation style:

- RPC Style (see **2.4.4 RPC Style**)

2.4.2 XML Representation of Interface Operation Component

```
<definitions>
  <interface>
    <operation
          name="xs:NCName"
          pattern="xs:anyURI"
          style="xs:anyURI"?
          safe="xs:boolean"? >
      <documentation />?
      [ <feature /> | <property /> |
        [ <input /> | <output /> | <infault /> |
<outfault /> ]+
      ]*
    </operation>
  </interface>
</definitions>
```

The XML representation for an Interface Operation component is an *element information item* with the following Infoset properties:

- A [local name] of `operation`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- Two or more *attribute information item*s amongst its [attributes] as follows:
    - A REQUIRED `name` *attribute information item* as described below in **2.4.2.1 name attribute information item with operation [owner]**.
    - A REQUIRED `pattern` *attribute information item* as described below in **2.4.2.2 pattern attribute information item with operation [owner]**.
    - An OPTIONAL `style` *attribute information item* as described below in **2.4.2.3 style attribute information item with operation [owner]**.
    - An OPTIONAL `safe` *attribute information item* as described below in **2.4.2.4 safe attribute information item with operation [owner]**.
    - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
    1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
    2. Zero or more *element information item*s from among the following, in any order:
        - Zero or more `input` *element information item*s (see **2.5.2 XML Representation of Message Reference Component**).
        - Zero or more `output` *element information item*s (see **2.5.2 XML Representation of Message Reference Component**).
        - Zero or more `infault` *element information item*s (see **2.6.2 XML Representation of Fault Reference Component**).
        - Zero or more `outfault` *element information item*s (see **2.6.2 XML Representation of Fault Reference Component**).
        - A `feature` *element information item* (see **2.7.2 XML Representation of Feature Component**).

- A `property` *element information item* (see **2.8.2 XML Representation of Property Component**).
- Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".

- At least one of the [children] MUST be an `input` , `output` , `infault` , or `outfault` *element information item*.

2.4.2.1 `name` *attribute information item with* `operation` *[owner]*

The `name` *attribute information item* identifies a given `operation` *element information item* inside a given `interface` *element information item*.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name`
- A [namespace name] which has no value

The type of the `name` *attribute information item* is *xs:NCName*.

2.4.2.2 `pattern` *attribute information item with* `operation` *[owner]*

The `pattern` *attribute information item* identifies the message exchange pattern a given operation uses.

The `pattern` *attribute information item* has the following Infoset properties:

- A [local name] of `pattern`
- A [namespace name] which has no value

The type of the `pattern` *attribute information item* is *xs:anyURI*.

2.4.2.3 `style` *attribute information item with* `operation` *[owner]*

The `style` *attribute information item* indicates the rules that were used to construct the {element} properties of the Message Reference components which are members of the {message references} property of the [owner] operation.

The `style` *attribute information item* has the following Infoset properties:

- A [local name] of `style`
- A [namespace name] which has no value

The type of the `style` *attribute information item* is *xs:anyURI*.

2.4.2.4 `safe` *attribute information item with* `operation` *[owner]*

The `safe` *attribute information item* indicates whether the operation is *safe* or not.

The `safe` *attribute information item* has the following Infoset properties:

- A [local name] of `safe`
- A [namespace name] which has no value

The type of the `safe` *attribute information item* is *xs:boolean* and does not have a default value.

2.4.3 Mapping Interface Operation's XML Representation to Component Properties

The mapping between the properties of the Interface Operation component (see **2.4.1 The Interface Operation Component**) and the XML Representation of the `operation` *element information item* (see **2.4.2 XML Representation of Interface Operation Component**) is as described in Table 2-4.

Table 2-4. Mapping between Interface Operation Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*
{target namespace}	The actual value of the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item* of the [parent] `interface` *element information item*.
{message exchange pattern}	The actual value of the `pattern` *attribute information item*
{message references}	The set of message references corresponding to the `input` and `output` *element information items* in [children], if any.
{fault references}	The set of fault references corresponding to the `infault` and `outfault` *element information items* in [children], if any.

{style}	The actual value of the `style` *attribute information item* if present, otherwise the actual value of the `styleDefault` *attribute information item* of the [parent] `interface` *element information item* if present, otherwise none.
{safety}	The actual value of the `safe` *attribute information item* if present, otherwise the value *false*.
{features}	The set of features corresponding to the `feature` *element information item*s in [children], if any.
{properties}	The set of properties corresponding to the `property` *element information item*s in [children], if any.

2.4.4 RPC Style

The RPC style is selected by assigning to an Interface Operation component's {style} property the value *http://www.w3.org/2004/03/wsdl/style/rpc*.

The RPC style MUST NOT be used for Interface Operation components whose {message exchange pattern} property has a value other than 'http://www.w3.org/2004/03/wsdl/in-only' or 'http://www.w3.org/2004/03/wsdl/in-out'.

Use of this value indicates that XML Schema [*XML Schema: Structures*] was used to define the schemas of the {element} properties of all {message reference} components of the Interface Operation component. Those schemas MUST adhere to the rules below.

Note that if the Interface Operation component uses the {message exchange pattern} 'http://www.w3.org/2004/03/wsdl/in-only' then there is no output element and hence the rules which refer to the output element do not apply.

- The content model of input and output {element} elements are defined using a complex type that contains a sequence from XML Schema.
- The sequence MUST only contain elements. It MUST NOT contain other structures such as xs:choice.
- The sequence MUST contain only local element children. Note that these child elements MAY contain the following attributes: nillable, minOccurs and maxOccurs.
- The LocalPart of input element's QName MUST be the same as the Interface operation component's name.
- The LocalPart of the output element's QName is obtained by concatenating the name of the operation and the string value

"Response", i.e. concat(operation/@name,"Response").

- Input and output elements MUST both be in the same namespace.
- The complex type that defines the body of an input or an output element MUST NOT contain any attributes.
- If elements with the same qualified name appear as children of both the input and output elements, then they MUST both be declared using the same type.
- The input or output sequence MUST NOT contain multiple children elements declared with the same name.

2.4.4.1 `wrpc:signature` ***Extension***

The `wrpc:signature` extension AII MAY be be used in conjunction with the RPC style to describe the exact signature of the function represented by an operation that uses the RPC style.

When present, the `wrpc:signature` extension contributes the following property to the interface operation component it is applied to:

- {rpc-signature} A (possibly empty) list of pairs *(q, t)* whose first component is of type *xs:QName* (as defined by [*XML Namespaces*]) and whose second component is of type *xs:Token* (as defined by [*XML Namespaces*]). Values for the second component MUST be chosen among the following four: "#in", "#out", "#inout" "#return".

The value of the {rpc-signature} property MUST satisfy the following conditions:

- The value of the first component of each pair *(q, t)* MUST be unique within the list.
- For each child element of the input and output messages of the operation, a pair *(q, t)* whose first component *q* is equal to the qualified name of that element MUST be present in the list, with the caveat that elements that appear with cardinality greater than one MUST be treated as as a single element.
- For each pair *(q, #in)*, there MUST be a child element of the input element with a name of *q* and there MUST NOT be a child element of the output element with the same name.
- For each pair *(q, #out)*, there MUST be a child element of the output element with a name of *q* and there MUST NOT be a child element of the input element with the same name.
- For each pair *(q, #inout)*, there MUST be a child element of the input element with a name of *q* and there MUST be a child element of the

output element with the same name. Furthermore, those two elements MUST have the same type.

- For each pair *(q, #return)*, there MUST be a child element of the output element with a name of *q* and there MUST NOT be a child element of the input element with the same name.

The function signature defined by a `wrpc:signature` extension is determined as follows:

1. Start with the value of the {rpc-signature} property, a (possibly empty) list of pairs of this form:

   [(q0, t0), (q1, t1), ...]
2. Filter the elements of this list into two lists, the first one *(L1)* comprising pairs whose *t* component is one of *{#in, #out, #inout}*, the second *(L2)* pairs whose *t* component is *#return*.

   For ease of visualization, let's denote the two lists as

   (L1) *[(a0, u0), (a1, u1),...]*

   and

   (L2) *[(r0, #return), (r1, #return),...]*

   respectively.
3. Then the formal signature of the function is

   f([d0] a0, [d1] a1, ...) => (r0, r1, ...)

   i.e.

   - the list of formal arguments to the function is *[a0, a1, ...]*;
   - the direction of each formal argument *a* is one of *[in]*, *[out]*, *[inout]*, determined according to the value of its corresponding *u* token;
   - the list of formal return parameters of the function is *[r0, r1, ...]*;
   - each formal argument and formal return parameter is typed according to the type of the child element identified by it (unique per the conditions given above).

2.4.4.2 XML Representation of the `wrpc:signature` Extension

The XML representation for the RPC signature extension is an *attribute information item* with the following Infoset properties:

- A [local name] of `signature`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl/rpc"

The type of the `name` *attribute information item* is a list type whose item type is the union of the *xs:QName* type and the subtype of the *xs:Token* type restricted to the following four values: "#in", "#out", "#inout", "#return". See Example 2-1 for a definition of this type.

Additionally, each even-numbered item (0, 2, 4, ...) in the list MUST be of type *xs:QName* and each odd-numbered item (1, 3, 5, ...) in the list MUST be of type *xs:Token*.

Example 2-1. Definition of the wrpc:signature extension

```
<xs:attribute name="signature" type="wrpc:
signatureType"/>

<xs:simpleType name="signatureType">
  <xs:list itemType="wrpc:signatureItemType"/>
</xs:simpleType>

<xs:simpleType name="signatureItemType">
  <xs:union memberTypes="wrpc:directionToken xsd:
QName"/>
</xs:simpleType>

<xs:simpleType name="directionToken">
  <xs:restriction base="xs:token">
    <xs:enumeration value="#in"/>
    <xs:enumeration value="#out"/>
    <xs:enumeration value="#inout"/>
    <xs:enumeration value="#return"/>
  </xs:restriction>
</xs:simpleType>
```

2.4.4.3 `wrpc:signature` *Extension Mapping To Properties of an Interface Operation Component*

A `wrpc:signature` extension *attribute information item* is mapped to the following property of the Interface Operation component (see **2.4.1 The Interface Operation Component**) defined by its [owner].

Table 2-5. Mapping of a `wrpc:signature` *Extension to Interface Operation Component Properties*

Property	Mapping

{rpc-signature}	A list of *(xs:QName, xs:Token)* pairs formed by grouping the items present in the actual value of the `wrpc:signature` *attribute information item* in the order in which they appear there.

2.5 Message Reference

2.5.1 The Message Reference Component

A Message Reference component associates to a message exchanged in an operation an XML element declaration that specifies its message content.

Message Reference components are identified by the role the message plays in the {message exchange pattern} that the operation is using. That is, a message exchange pattern defines a set /meof placeholder messages that participate in the pattern and assigns them unique names within the pattern. The purpose of a Message Reference component is to associate an actual message type (XML element declaration or some other declaration (see **3.2 Using Other Schema Languages**) for message content) with the message that will perform a specific role in the message exchange pattern.

The properties of the Message Reference component are as follows:

- {message label} An NCName as defined by [*XML Namespaces*]. This property identifies the role this message plays in the {message exchange pattern} of the Interface Operation component this is contained within. The value of this property MUST match the name of a placeholder message defined by the message exchange pattern.
- {direction} One of *in* or *out* indicating whether the message is coming to the service or going from the service, respectively. The direction MUST be the same as the direction of the message identified by the {message label} property in the {message exchange pattern} of the Interface Operation component this is contained within.
- {message content model} A token with one of the values *#any*, *#none*, or *#element*. A value of *#any* indicates that the message content is any single element. A value of *#none* indicates there is no message content. A value of *#element* indicates that the message consists of a single element described by the global element declaration reference by the {element} property.
- {element} A reference to an XML element declaration in the {element declarations} property of **2.1.1 The Definitions Component**. This element represents the content or "payload" of the message. When the {message content model} property has the value *#any* or *#none* the {element} property has no value.

If a non-XML type system is in use (as considered in **3.2 Using Other Schema Languages**) then additional properties would need to be added to the Message Reference Component (along with extensibility attributes to its XML representation) to allow associating such message types with the message reference.

For each Message Reference component in the {message references} property of an Interface Operation component, its {message label} property MUST be unique.

2.5.2 XML Representation of Message Reference Component

```
<definitions>
  <interface>
    <operation>
      <input
            messageLabel="xs:NCName"?
            element="union of xs:QName, xs:Token"? >
        <documentation />?
      </input>
      <output
            messageLabel="xs:NCName"?
            element="union of xs:QName, xs:Token"? >
        <documentation />?
      </output>
    </operation>
  </interface>
</definitions>
```

The XML representation for a Message Reference component is an *element information item* with the following Infoset properties:

- A [local name] of `input` or `output`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- Zero or more *attribute information item*s amongst its [attributes] as follows:
  - An OPTIONAL `messageLabel` *attribute information item* as described below in **2.5.2.1 messageLabel attribute information item with input, or output [owner]**.

    If the {message exchange pattern} of the Interface Operation component has only one message with a given value for {direction}, then the `messageLabel` *attribute information item* is optional for the XML representation of the Message Reference component with that {direction}.

    - An OPTIONAL `element` *attribute information item* as described below in **2.5.2.2 element attribute information item with input, or output [owner]**.
    - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item*s amongst its [children], in order, as follows:
    1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
    2. Zero or more namespace-qualified *element information item*s. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".

2.5.2.1 `messageLabel` attribute information item with `input` , or `output` [owner]

The `messageLabel` *attribute information item* identifies the role of this message in the message exchange pattern of the given `operation` *element information item*.

The `messageLabel` *attribute information item* has the following Infoset properties:

- A [local name] of `messageLabel`
- A [namespace name] which has no value

The type of the `messageLabel` *attribute information item* is *xs:NCName*.

2.5.2.2 `element` attribute information item with `input` , or `output` [owner]

The `element` *attribute information item* has the following Infoset properties:

- A [local name] of `element` .
- A [namespace name] which has no value.

The type of the `element` *attribute information item* is a union of *xs:QName* and *xs:Token* where the allowed token values are *#any* or *#none*.

2.5.3 Mapping Message Reference's XML Representation to Component Properties

The mapping between the properties of the Message Reference component (see **2.5.1 The Message Reference Component**) and the XML Representation of the message reference *element information item* (see **2.5.2 XML Representation of Message Reference Component**) is as described in Table 2-6.

Table 2-6. Mapping between Message Reference Component Properties and XML Representation

Property	Mapping
{message label}	The actual value of the `messageLabel` *attribute information item* if any; otherwise the {message label} property of the message with same {direction} from the {message exchange pattern} of the Interface Operation component, provided there is exactly one such message; otherwise empty.
{direction}	If the [local name] of the *element information item* is `input` then "in", else if the [local name] of the *element information item* is `output` then "out".
{message content model}	If the `element` *attribute information item* is present and its value is a QName, then *#element*. Otherwise the actual value of the `element` *attribute information item*, if any.
{element}	If the `element` *attribute information item* is present and its value is a QName, then the element declaration from the {element declarations} property of **2.1.1 The Definitions Component** resolved to by the value of the `element` *attribute information item*, otherwise empty. It is an error for the `element` *attribute information item* to have a value and for it to not resolve to a global element declaration from the {element declarations} property of **2.1.1 The Definitions Component**.

2.6 Fault Reference

2.6.1 The Fault Reference Component

A Fault Reference component associates a Fault component that defines the fault message type for a fault that occurs related to a message participating in an operation.

Fault Reference components are identified by the role the related message plays in the {message exchange pattern} that the operation is using. That is, a message exchange pattern defines a set of placeholder messages that participate in the pattern and assigns them unique labels within the pattern. The purpose of a Fault Reference component is to associate an actual Fault component for the fault that will occur with a specific message in the message exchange pattern.

The companion specification [*WSDL 2.0 Message Exchange Patterns*] defines two *fault patterns* that a given message exchange pattern may use. For the pattern *fault-replaces-message*, the message that the fault relates to identifies the message *in place of which* the declared fault message will occur. Thus, the fault message will travel in the *same* direction as the message it replaces in the pattern. For the pattern *message-triggers-fault*, the message that the fault relates to identifies the message after which the indicated fault may occur, in the opposite direction of the referred to message. That is, the fault message will travel in the *opposite* direction of the message it comes after in the pattern.

More than one Fault Reference component may refer to the same message label. This allows one to indicate that there is more than one type of fault that is related to that message.

The properties of the Fault Reference component are as follows:

- {message label} An NCName as defined by [*XML Namespaces*]. This property identifies the message this fault relates to among those defined in the {message exchange pattern} property of the Interface Operation component it is contained within. The value of this property MUST match the name of a placeholder message defined by the message exchange pattern.
- {direction} One of *in* or *out* indicating whether the fault is coming to the service or going from the service, respectively. The direction MUST be consistent with the direction implied by the fault rule used in the message exchange pattern of the operation. For example, if the fault rule *fault-replaces-message* is used, then a fault which refers to an outgoing message would have a {direction} property value of *out*. On the other hand, if the fault rule *message-triggers-fault* is used, then a fault which refers to an outgoing message would have a {direction} property value of *in* as the fault travels in the opposite direction of the message.
- {fault reference} A reference to a Fault component in the {faults} property of the parent Interface Operation component's parent Interface component. Identifying the Fault component therefore indirectly defines the actual content or payload of the fault message.

2.6.2 XML Representation of Fault Reference Component

```
<definitions>
  <interface>
    <operation>
      <infault
            ref="xs:QName"
            messageLabel="xs:NCName"? >
        <documentation />?
      </infault>*
      <outfault
            ref="xs:QName"
            messageLabel="xs:NCName"? >
        <documentation />?
      </outfault>*
    </operation>
  </interface>
</definitions>
```

The XML representation for a Fault Reference component is an *element information item* with the following Infoset properties:

- A [local name] of `infault` or `outfault`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `ref` *attribute information item* as described below in **2.6.2.1 ref attribute information item with infault, or outfault [owner]**.
  - An OPTIONAL `messageLabel` *attribute information item* as described below in **2.6.2.2 messageLabel attribute information item with infault, or outfault [owner]**.

    If the {message exchange pattern} of the Interface Operation component has only one message with a given value for {direction}, the `messageLabel` *attribute information item* is optional for the XML representation of any Fault Reference component with the same value for {direction} (if the *fault pattern* of the {message exchange pattern} is *fault-replaces-message*) or of any Fault Reference component with the opposite value for {direction} (if the *fault pattern* is *message-triggers-fault*).
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item*s amongst its [children], in order,

as follows:

1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
2. Zero or more namespace-qualified *element information item*s. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".

2.6.2.1 `ref` *attribute information item with* `infault` *, or* `outfault` *[owner]*

The `ref` *attribute information item* refers to a fault component.

The `ref` *attribute information item* has the following Infoset properties:

- A [local name] of `ref`
- A [namespace name] which has no value

The type of the `fault` *attribute information item* is *xs:QName*.

2.6.2.2 `messageLabel` *attribute information item with* `infault` *, or* `outfault` *[owner]*

The `messageLabel` *attribute information item* identifies the message in the message exchange pattern of the given `operation` *element information item* to which this fault is related to.

The `messageLabel` *attribute information item* has the following Infoset properties:

- A [local name] of `messageLabel`
- A [namespace name] which has no value

The type of the `messageLabel` *attribute information item* is *xs:NCName*.

2.6.3 Mapping Fault Reference's XML Representation to Component Properties

The mapping between the properties of the Fault Reference component (see **2.6.1 The Fault Reference Component**) and the XML Representation of the message reference *element information item* (see **2.6.2 XML Representation of Fault Reference Component**) is as described in Table 2-7.

Table 2-7. Mapping between Fault Reference Component Properties and XML Representation

Property	Mapping
{fault reference}	The actual value of the `ref` *attribute information item*
{message label}	The actual value of the `messageLabel` *attribute information item* if any; otherwise the {message label} property of the message with the same {direction} from the {message exchange pattern} of the Interface Operation component, provided there is exactly one such message and the *fault pattern* of the {message exchange pattern} is *fault-replaces-message*; otherwise the {message reference} property of the message with the opposite {direction}, provided there is exactly one such message and the *fault pattern* is *message-triggers-fault*; otherwise empty.
{direction}	If the [local name] of the *element information item* is `infault` then "in", else if the [local name] of the *element information item* is `outfault` then "out".

2.7 Feature

2.7.1 The Feature Component

A feature component describes an abstract piece of functionality typically associated with the exchange of messages between communicating parties. Although WSDL poses no constraints on the potential scope of such features, examples might include "reliability", "security", "correlation", and "routing". The presence of a feature component in a WSDL description indicates that the service supports the feature and may require a requester agent that interacts with the service to use that feature. Each Feature is identified by its URI.

The properties of the Feature component are as follows:

- {name} An absolute URI as defined by [*IETF RFC 2396*]. This URI SHOULD be dereferenceable to a document that directly or indirectly defines the meaning and use of the Feature that it identifies.
- {required} A boolean value. If the {require} property is true, then the requester agent MUST use the Feature that is identified by the {name} URI. Otherwise, the requester agent MAY use the Feature that is identified by the {name} URI. In either case, if the requester agent does use the Feature that is identified by the {name} URI, then the requester agent MUST obey all semantics implied by the definition of that Feature.

2.7.1.1 Feature Composition Model

The set of features which are required or available for a given service and a

particular interaction consists of the combined set of ALL feature declarations in the following scope. The list is in order of increasing specificity.

- The interface component.
- The specific interface operation component.
- The specific message reference component.
- The binding component.
- The specific binding operation component.
- The specific binding message or fault reference component.

Note that multiple declarations of the same feature have no effect on the combined set of active features, since features are either in use or not, with no multiplicity. If multiple declarations of the same feature are in scope for a given interaction, the feature is required if ANY of the in scope declarations have the `required` attribute set to "true".

2.7.1.1.1 Example of Feature Composition Model

In the following example, the `depositFunds` operation on the `BankService` has to be used with the `ISO9001` , the `notarization` and the `secure-channel` features; they are all in scope. The fact that the `notarization` feature is declared both in the operation and in the service has no effect.

```
<definitions targetNamespace="http://example.com/bank"
     xmlns:ns1="http://example.com/bank">
  <interface name="ns1:Bank">
    <!-- All uses of this interface must be secure -->
    <feature uri="http://example.com/secure-channel"
             required="true"/>
    <operation name="withdrawFunds">
      <!-- This operation must have ACID properties -->
      <feature uri="http://example.com/transaction"
               required="true"/>
      ...
    </operation>
    <operation name="depositFunds">
      <!-- This operation requires notarization -->
      <feature uri="http://example.com/notarization"
               required="true"/>
      ...
    </operation>
  </interface>
  <binding name="ns1:BankSOAPBinding">
  </binding>
```

```
  <service name="ns1:BankService"
           interface="tns:Bank">
   <!-- This particular service requires ISO9001
        compliance to be verifiable -->
   <feature uri="http://example.com/ISO9001"
            required="true"/>
   <!-- This service also requires notarization -->
   <feature uri="http://example.com/notarization"
            required="true"/>
   <endpoint>
     ...
   </endpoint>
  </service>
</definitions>
```

2.7.2 XML Representation of Feature Component

```
<feature
      uri="xs:anyURI"
      required="xs:boolean"? >
  <documentation />?
</feature>
```

The XML representation for a Feature component is an *element information item* with the following Infoset properties:

- A [local name] of `feature`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `uri` *attribute information item* as described below in **2.7.2.1 uri attribute information item with feature [owner]**.
  - An OPTIONAL `required` *attribute information item* as described below in **2.7.2.2 required attribute information item with feature [owner]**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item*s amongst its [children], in order as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).

2. Zero or more namespace-qualified *element information items*. The [namespace name] of such *element information items* MUST NOT be "http://www.w3.org/2004/03/wsdl".

2.7.2.1 `uri` *attribute information item with* `feature` *[owner]*

The `uri` *attribute information item* specifies the URI of the feature.

The `uri` *attribute information item* has the following Infoset properties:

- A [local name] of `uri`
- A [namespace name] which has no value

The type of the `uri` *attribute information item* is `xs:anyURI` .

2.7.2.2 `required` *attribute information item with* `feature` *[owner]*

The `required` *attribute information item* specifies whether the use of the feature is mandatory or optional.

The `required` *attribute information item* has the following Infoset properties:

- A [local name] of `required`
- A [namespace name] which has no value

The type of the `required` *attribute information item* is `xs:boolean` .

2.7.3 Mapping Feature's XML Representation to Component Properties

The mapping between the properties of the Feature component (see **2.7.1 The Feature Component**) and the XML Representation of the `feature` *element information item* (see **2.7.2 XML Representation of Feature Component**) is as described in Table 2-8.

Table 2-8. Mapping between Feature Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `uri` *attribute information item*
{required}	If the value of the `required` *attribute information item* is "true" or "1", then "true", otherwise "false".

2.8 Property

2.8.1 The Property Component

A Property component describes the set of possible values for a particular property. The permissible values are specified by references to a Schema description. A property is typically used to control a feature's behavior. Properties, and hence property values, can be shared amongst features.

The properties of the Property component are as follows:

- {name} An absolute URI as defined by [*IETF RFC 2396*]. This URI SHOULD be dereferenceable to a document that directly or indirectly defines the meaning and use of the Property that it identifies.
- {required} A boolean value. If the {required} property is true, then the requester agent MUST use the Property that is identified by the {name} URI. Otherwise, the requester agent MAY use the Property that is identified by the {name} URI. In either case, if the requester agent does use the Property that is identified by the {name} URI, then the requester agent MUST obey all semantics implied by the definition of that Property.
- {value constraint} A type definition constraining the value of the property.
- {value} The value of the property.

2.8.1.1 Property Composition Model

At runtime, the behaviour of features, (SOAP) modules and bindings may be affected by the values of in-scope properties. Properties combine into a virtual "execution context" which maps property names (URIs) to constraints. Each property URI MAY therefore be associated with AT MOST one property constraint for a given interaction.

The particular set of constraints for a given service and a particular interaction consists of the combined set of ALL constraints in the following scope. The list is in order of increasing specificity, and if a given property URI is constrained in a later scope, it overrides the earlier constraint.

- The interface component.
- The specific interface operation component.
- The specific message reference component.
- The binding component.
- The specific binding operation component.
- The specific binding message or fault reference component.

Note that, in the text above, "property constraint" (or, simply, "constraint") is

used to mean EITHER a `constraint` inside a property component OR a `value` , since `value` may be considered a special case of `constraint` .

2.8.2 XML Representation of Property Component

```
<property
      uri="xs:anyURI"
      required="xs:boolean"? >
  <documentation />?
  [ <value /> | <constraint /> ]
</property>
```

The XML representation for a Property component is an *element information item* with the following Infoset properties:

- A [local name] of `property`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `uri` *attribute information item* as described below in **2.8.2.1 uri attribute information item with property [owner]**.
  - An OPTIONAL `required` *attribute information item* as described below in **2.8.2.2 required attribute information item with feature [owner]**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- One or more *element information item*s amongst its [children], in order as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
  2. One REQUIRED *element information item* from among the following:
     - A `value` *element information item* as described in **2.8.2.3 value element information item with property [parent]**
     - A `constraint` *element information item* as described in **2.8.2.4 constraint element information item with property [parent]**
  3. Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/

wsdl".

2.8.2.1 `uri` *attribute information item with* `property` *[owner]*

The `uri` *attribute information item* specifies the URI of the property. It has the following Infoset properties:

- A [local name] of `uri`
- A [namespace name] which has no value

The type of the `uri` *attribute information item* is `xs:anyURI` .

2.8.2.2 `required` *attribute information item with* `feature` *[owner]*

The `required` *attribute information item* specifies whether use of the property is mandatory or optional.

The `required` *attribute information item* has the following Infoset properties:

- A [local name] of `required`
- A [namespace name] which has no value

The type of the `required` *attribute information item* is `xs:boolean` .

2.8.2.3 `value` *element information item with* `property` *[parent]*

```
<property>
  <value>
    xs:anyType
  </value>
</property>
```

The `value` *element information item* specifies the value of the property. It has the following Infoset properties:

- A [local name] of `value`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"

The type of the `value` *element information item* is `xs:anyType` .

2.8.2.4 `constraint` *element information item with* `property` *[parent]*

```
<property>
```

```
    <constraint>
      xs:QName
    </constraint>
  </property>
```

The `constraint` *element information item* specifies a constraint on the value of the property. It has the following Infoset properties:

- A [local name] of `constraint`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"

The type of the `constraint` *attribute information item* is `xs:QName` .

2.8.3 Mapping Property's XML Representation to Component Properties

The mapping between the properties of the Property component (see **2.8.1 The Property Component**) and the XML Representation of the `property` *element information item* (see **2.8.2 XML Representation of Property Component**) is as described in Table 2-9.

Table 2-9. Mapping between Property Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `uri` *attribute information item*
{value constraint}	If the `constraint` *element information item* is present, the type referred to by the value of this *element information item*. Otherwise, an anonymous type, whose base type is "xs:anyType", with a single "enumeration" facet whose value is the type of the value of the `value` *element information item*. Otherwise, "xs:anyType".
{value}	The actual value of the `value` *element information item*, if any.

2.9 Binding

2.9.1 The Binding Component

A Binding component describes a concrete message format and transmission protocol which may be used to define an endpoint (see **2.14 Endpoint**). Binding components can be used to describe such information in a re-usable manner for any interface or specifically for a given interface. Furthermore, binding information MAY be specified on a per-operation basis (see **2.11.1**

The Binding Operation Component) within an interface in addition to across all operations of an interface.

If a Binding component specifies any operation-specific binding details (by including Binding Operation components) or any fault binding details (by including Binding Fault components) then it MUST specify an interface the Binding component applies to, so as to indicate which interface the operations come from.

Conversely, a Binding component which omits any operation-specific binding details and any fault binding details MAY omit specifying an interface. Binding components that do not specify an interface MAY be used to specify operation-independent binding details for Service components with different interfaces. That is, such Binding components are reusable across one or more interfaces.

No concrete binding details are given in this specification. The companion specification, *WSDL (Version 2.0): Bindings* [*WSDL 2.0 Bindings*] defines such bindings for SOAP 1.2 [*SOAP 1.2 Part 1: Messaging Framework*] and HTTP [*IETF RFC 2616*]. Other specifications MAY define additional binding details. Such specifications are expected to annotate the Binding component (and its sub-components) with additional properties and specify the mapping between those properties and the XML representation.

A Binding component which defines bindings for an Interface component MUST define bindings for all the operations of that Interface component. The bindings may occur via defaulting rules which allow one to specify default bindings for all operations (see, for example [*WSDL 2.0 Bindings*]) or by directly listing each Operation component of the Interface component and defining bindings for them. Thus, it is an error for a Binding component to not define bindings for all the Operation components of the Interface component for which the Binding component purportedly defines bindings for.

Bindings are named constructs and can be referred to by QName (see **2.17 QName resolution**). For instance, Endpoint components refer to bindings in this way.

The properties of the Binding component are as follows:

- {name} An NCName as defined by [*XML Namespaces*].
- {target namespace} A namespace name, as defined in [*XML Namespaces*].
- {interface} An named interface definition indicating the interface for which binding information is being specified.
- {faults} A set of named binding fault definitions.
- {operations} A set of named binding operation definitions.
- {features} A set of named feature definitions.

- {properties} A set of named property definitions.

For each Binding component in the {bindings} property of a definitions container, the combination of {name} and {target namespace} properties must be unique.

2.9.2 XML Representation of Binding Component

```
<definitions>
  <binding
        name="xs:NCName"
        interface="xs:QName"? >
    <documentation />?
    [ <fault /> | <operation /> | <feature /> |
<property /> ]*
  </binding>
</definitions>
```

The XML representation for a Binding component is an *element information item* with the following Infoset properties:

- A [local name] of `binding`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.9.2.1 name attribute information item with binding [owner]**.
  - An OPTIONAL `interface` *attribute information item* as described below in **2.9.2.2 interface attribute information item with binding [owner]**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item*s amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `fault` *element information item*s (see **2.10.2 XML Representation of Binding Fault**

Component).

- Zero or more `operation` *element information item*s (see **2.11.2 XML Representation of Binding Operation Component**).
- Zero or more `feature` *element information item*s (see **2.7.2 XML Representation of Feature Component**).
- Zero or more `property` *element information item*s (see **2.8.2 XML Representation of Property Component**).
- Zero or more namespace-qualified *element information item*s. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl". Such *element information item*s are considered to be binding extension elements(see **2.9.2.3 Binding extension elements**).

2.9.2.1 `name` *attribute information item with* `binding` *[owner]*

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the `definitions` *element information item* forms the QName of the binding.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name`
- A [namespace name] which has no value

The type of the `name` *attribute information item* is *xs:NCName*.

2.9.2.2 `interface` *attribute information item with* `binding` *[owner]*

The `interface` *attribute information item* refers, by QName, to an Interface component.

The `interface` *attribute information item* has the following Infoset properties:

- A [local name] of `interface`
- A [namespace name] which has no value

The type of the `interface` *attribute information item* is *xs:QName*.

2.9.2.3 Binding extension elements

Binding extension elements are used to provide information specific to a

particular binding. The semantics of such *element information items* are defined by the specification for those *element information items*. Such specifications are expected to annotate the Binding component with additional properties and specify the mapping between those properties and the XML representation.

2.9.3 Mapping Binding's XML Representation to Component Properties

The mapping between the properties of the Binding component (see **2.9.1 The Binding Component**) and the XML Representation of the `binding` *element information item* (see **2.9.2 XML Representation of Binding Component**) is as described in Table 2-10.

Table 2-10. Mapping between Binding Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*
{target namespace}	The actual value of the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item*.
{interface}	The Interface component resolved to by the actual value of the `interface` *attribute information item*, if any.
{faults}	The set of Binding Fault components corresponding to the `fault` *element information items* in [children], if any.
{operations}	The set of Binding Operation components corresponding to the `operation` *element information items* in [children], if any.
{features}	The set of Feature components corresponding to the `feature` *element information items* in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information items* in [children], if any.

2.10 Binding Fault

2.10.1 The Binding Fault Component

A Binding Fault component describes a concrete binding of a particular fault within an interface to a particular concrete message format. A particular fault

of an interface is uniquely identified by the target namespace of the interface and the name of the fault within that interface.

Note that the fault does not occur by itself - it occurs as part of a message exchange as defined by an Interface Operation component (and its binding counterpart the Binding Operation component). Thus, the fault binding information specified in a Binding Fault component describes how faults that occur within a message exchange of an operation will be formatted.

The properties of the Binding Fault component are as follows:

- {fault reference} A QName as defined by [*XML Namespaces*] which refers to an Interface Fault component in the {faults} property of the Interface component identified by the {interface} property of the parent Binding component. This is the Interface Fault component for which binding information is being specified.

For each Binding Fault component in the {faults} property of a Binding component, the {fault reference} property MUST be unique. That is, one cannot define multiple bindings for the same fault within a given Binding component.

2.10.2 XML Representation of Binding Fault Component

```
<definitions>
  <binding>
    <fault
          ref="xs:QName" >
      <documentation />?
    </fault>
  </binding>
</definitions>
```

The XML representation for a Binding Fault component is an *element information item* with the following Infoset properties:

- A [local name] of `fault`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `ref` *attribute information item* as described below in **2.10.2.1 ref attribute information item with fault [owner]**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".

- Zero or more *element information item*s amongst its [children], in order, as follows:
    1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
    2. Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl". Such *element information item*s are considered to be binding fault extension elements as described below (see **2.10.2.2 Binding Fault extension elements**).

2.10.2.1 `ref` attribute information item with `fault` [owner]

The `ref` *attribute information item* has the following Infoset properties:

- A [local name] of `ref`
- A [namespace name] which has no value

The type of the `ref` *attribute information item* is *xs:QName*.

2.10.2.2 Binding Fault extension elements

Binding Fault extension elements are used to provide information specific to a particular fault in a binding. The semantics of such *element information item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Binding Fault component with additional properties and specify the mapping between those properties and the XML representation.

2.10.3 Mapping Binding Fault's XML Representation to Component Properties

The mapping between the properties of the Binding Fault component (see **2.10.1 The Binding Fault Component**) and the XML Representation of the `fault` *element information item* (see **2.10.2 XML Representation of Binding Fault Component**) is as described in Table 2-11.

Table 2-11. Mapping between Binding Fault Component Properties and XML Representation

Property	Mapping
{fault reference}	The actual value of the `ref` *attribute information item*.

2.11 Binding Operation

2.11.1 The Binding Operation Component

A Binding Operation component describes a concrete binding of a particular operation of an interface to a particular concrete message format. A particular operation of an interface is uniquely identified by the target namespace of the interface and the name of the operation within that interface.

The properties of the Binding Operation component are as follows:

- {operation reference} A QName as defined by [*XML Namespaces*] which refers to an Interface Operation component in the {operations} property of the Interface component identified by the {interface} property of the parent Binding component. This is the Interface Operation component for which binding information is being specified.
- {message references} A set of Binding Message Reference components

For each Binding Operation component in the {operations} property of a Binding component, the {operation reference} property MUST be unique. That is, one cannot define multiple bindings for the same operation within a given Binding component.

Interface Operation components are local to Interface components; they cannot be referred to by QName, despite having both {name} and {target namespace} properties. That is, two Interface components sharing the same {target namespace} property but with different {name} properties MAY contain Interface Operation components which share the same {name} property. Thus, the {name} and {target namespace} properties of the Interface Operation components are not sufficient to form the unique identity of an Interface Operation component. To uniquely identify an Interface Operation component one must first identify the Interface component (by QName) and then identify the Interface Operation within that Interface component (by a further QName).

2.11.2 XML Representation of Binding Operation Component

```
<definitions>
  <binding>
    <operation
          ref="xs:QName" >
      <documentation />?
      [ <input /> | <output /> | <feature /> |
<property /> ]*
    </operation>
  </binding>
```

```
</definitions>
```

The XML representation for a Binding Operation component is an *element information item* with the following Infoset properties:

- A [local name] of `operation`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `ref` *attribute information item* as described below in **2.11.2.1 ref attribute information item with operation [owner]**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item*s amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
  2. Zero or more *element information item*s from among the following, in any order:
     - Zero or more `input` *element information item*s (see **2.12 Binding Message Reference**)
     - Zero or more `output` *element information item*s (see **2.12 Binding Message Reference**)
     - Zero or more `feature` *element information item*s
     - Zero or more `property` *element information item*s
     - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl". Such *element information item*s are considered to be binding operation extension elements as described below (see **2.11.2.2 Binding Operation extension elements**).

2.11.2.1 `ref` *attribute information item with* `operation` *[owner]*

The `ref` *attribute information item* has the following Infoset properties:

- A [local name] of `ref`

- A [namespace name] which has no value

The type of the `ref` *attribute information item* is *xs:QName*.

2.11.2.2 Binding Operation extension elements

Binding Operation extension elements are used to provide information specific to a particular operation in a binding. The semantics of such *element information item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Binding Operation component with additional properties and specify the mapping between those properties and the XML representation.

2.11.3 Mapping Binding Operation's XML Representation to Component Properties

The mapping between the properties of the Binding Operation component (see **2.11.1 The Binding Operation Component**) and the XML Representation of the `operation` *element information item* (see **2.11.2 XML Representation of Binding Operation Component**) is as described in Table 2-12.

Table 2-12. Mapping between Binding Operation Component Properties and XML Representation

Property	Mapping
{operation reference}	The actual value of the `ref` *attribute information item*.
{messages references}	The set of Binding Message Reference components corresponding to the `input` and `output` *element information item*s in [children], if any.
{features}	The set of Feature components corresponding to the `feature` *element information item* in [children], if any.
{properties}	The set of Property components corresponding to the `property` *element information item* in [children], if any.

2.12 Binding Message Reference

2.12.1 The Binding Message Reference Component

A Binding Message Reference component describes a concrete binding of a

particular message participating in an operation to a particular concrete message format.

The properties of the Binding Message Reference component are as follows:

- {message label} An NCName as defined by [*XML Namespaces*]. The value of this property identifies the role that the message for which binding details are being specified plays in the {message exchange pattern} of the Interface Operation component being bound by the containing Binding Operation component.
- {direction} One of *in* or *out* indicating whether the message is coming to the service or going from the service, respectively. The direction MUST be the same as the direction of the message identified by the {message label} property in the {message exchange pattern} of the Interface Operation component being bound by the containing Binding Operation component.

For each Binding Message Reference component in the {message references} property of a Binding Operation component, the {message label} property MUST be unique. That is, the same message cannot be bound twice within the same operation.

2.12.2 XML Representation of Binding Message Reference Component

```
<definitions>
  <binding>
    <operation>
      <input
            messageLabel="xs:NCName"? >
        <documentation />?
      </input>
      <output
            messageLabel="xs:NCName"? >
        <documentation />?
      </output>
    </operation>
  </binding>
</definitions>
```

The XML representation for a Binding Message Reference component is an *element information item* with the following Infoset properties:

- A [local name] of `input` or `output` .
- A [namespace name] of "http://www.w3.org/2004/03/wsdl".
- One or more *attribute information item*s amongst its [attributes] as

follows:

- An OPTIONAL `messageLabel` *attribute information item* as described below in **2.12.2.1 messageLabel attribute information item with input or output [owner]**.

  If the {message exchange pattern} of the Interface Operation component being bound has only one message with a given value for {direction}, then the `messageLabel` *attribute information item* is optional for the XML representation of the Binding Message Reference component with that {direction}.
- Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".

- Zero or more *element information item*s amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
  2. Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl". Such *element information item*s are considered to be binding message reference extension elements, as described below (see **2.12.2.2 Binding Message Reference extension elements**).

2.12.2.1 `messageLabel` *attribute information item with* `input` *or* `output` *[owner]*

The `messageLabel` *attribute information item* has the following Infoset properties:

- A [local name] of `messageLabel` .
- A [namespace name] which has no value.

The type of the `messageLabel` *attribute information item* is *xs:NCName*.

2.12.2.2 Binding Message Reference extension elements

Binding Message Reference extension elements are used to provide information specific to a particular message in an operation. The semantics of such *element information item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Binding Message Reference component with additional properties and specify

the mapping between those properties and the XML representation.

2.12.3 Mapping Binding Message Reference's XML Representation to Component Properties

The mapping between the properties of the Binding Message Reference component (see **2.12.1 The Binding Message Reference Component**) and the XML Representation of the `binding` *element information item* (see **2.12.2 XML Representation of Binding Message Reference Component**) is as described in Table 2-13.

Table 2-13. Mapping between Binding Message Reference Component Properties and XML Representation

Property	Mapping
{message label}	The actual value of the `messageLabel` *attribute information item* if any; otherwise the {message label} property of the message with same {direction} from the {message exchange pattern} of the Interface Operation component being bound, provided there is exactly one such message; otherwise empty.
{direction}	If the [local name] of the *element information item* is `input` then "in", else if the [local name] of the *element information item* is `output` then "out".

2.13 Service

2.13.1 The Service Component

A Service component describes a set of endpoints (see **2.14 Endpoint**) at which the single interface of the service is provided. The endpoints thus are in effect alternate places at which the service is provided.

Services are named constructs and can be referred to by QName (see **2.17 QName resolution**).

The properties of the Service component are as follows:

- {name} An NCName as defined by [*XML Namespaces*].
- {target namespace} A namespace name, as defined in [*XML Namespaces*].
- {interface} An Interface component.
- {endpoints} A set of Endpoint components.

For each Service component in the {services} property of a definitions container, the combination of {name} and {target namespace} properties MUST be unique.

2.13.2 XML Representation of Service Component

```
<definitions>
  <service
        name="xs:NCName"
        interface="xs:QName" >
    <documentation />?
    <endpoint />+
  </service>
</definitions>
```

The XML representation for a Service component is an *element information item* with the following Infoset properties:

- A [local name] of `service`
- A [namespace name] of "http://www.w3.org/2004/03/wsdl"
- Two or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.13.2.1 name attribute information item with service [owner]**.
  - A REQUIRED `interface` *attribute information item* as described below in **2.13.2.2 interface attribute information item with service [owner]**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- One or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
  2. One or more *element information item*s from among the following, in any order:
     - One or more `endpoint` *element information item*s (see **2.14.2 XML Representation of Endpoint Component**
     - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of

such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".

Note that the XML Schema [*XML Schema: Structures*] type of the *element information item* `service` as defined in the WSDL schema MAY be used as the basis for defining new elements which can be used as service references in message exchanges. To enable such reuse, the WSDL schema defines the *attribute information item* `name` as optional in the type of the *element information item* `service` , while it is REQUIRED for the *element information item* `service` as indicated above.

Note:

See the primer [*WSDL 2.0 Primer*] for more information and examples.

2.13.2.1 `name` *attribute information item with* `service` *[owner]*

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the `definitions` *element information item* forms the QName of the service.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name`
- A [namespace name] which has no value

The type of the `name` *attribute information item* is *xs:NCName*.

2.13.2.2 `interface` *attribute information item with* `service` *[owner]*

The `interface` *attribute information item* identifies the interface that the service is an instance of.

The `interface` *attribute information item* has the following Infoset properties:

- A [local name] of `interface`
- A [namespace name] which has no value

The type of the `interface` *attribute information item* is *xs:QName.*.

2.13.3 Mapping Service's XML Representation to Component Properties

The mapping between the properties of the Service component (see **2.13.1 The Service Component**) and the XML Representation of the `service` *element information item* (see **2.13.2 XML Representation of Service Component**) is as described in Table 2-14.

Table 2-14. Mapping between Service Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*
{target namespace}	The actual value of the `targetNamespace` *attribute information item* of the [parent] `definitions` *element information item*
{interface}	The Interface component resolved to by the actual value of the `interface` *attribute information item*.
{endpoints}	The Endpoint components corresponding to the `endpoint` *element information item*s in [children] if any.

2.14 Endpoint

2.14.1 The Endpoint Component

An Endpoint component defines the particulars of a specific endpoint at which a given service is available.

Endpoint components are local to a given Service component; they cannot be referred to by QName.

The properties of the Endpoint component are as follows:

- {name} An NCName as defined by [*XML Namespaces*].
- {binding} A named Binding component.

For each Endpoint component in the {endpoints} property of a Service component, the {binding} property (see **2.14.1 The Endpoint Component**) MUST either be a Binding component with an unspecified {interface} property (see **2.9.1 The Binding Component** or a Binding component with an {interface} property equal to the {interface} property of the Service component.

For each Endpoint component in the {endpoints} property of a Service component, the {name} property MUST be unique.

2.14.2 XML Representation of Endpoint Component

```
<definitions>
  <service>
    <endpoint
          name="xs:NCName"
```

```
            binding="xs:QName" >
          <documentation />?
        </endpoint>
      </service>+
    </definitions>
```

The XML representation for a Endpoint component is an *element information item* with the following Infoset properties:

- A [local name] of `endpoint` .
- A [namespace name] of "http://www.w3.org/2004/03/wsdl".
- Two or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `name` *attribute information item* as described below in **2.14.2.1 name attribute information item with endpoint [owner]**.
  - A REQUIRED `binding` *attribute information item* as described below in **2.14.2.2 binding attribute information item with endpoint [owner]**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item* amongst its [children], in order, as follows:
  1. An OPTIONAL `documentation` *element information item* (see **5. Documentation**).
  2. Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl". Such *element information item*s are considered to be endpoint extension elements (see **2.14.2.3 Endpoint extension elements**).

2.14.2.1 `name` ***attribute information item with*** `endpoint` ***[owner]***

The `name` *attribute information item* together with the `targetNamespace` *attribute information item* of the `definitions` *element information item* forms the QName of the endpoint.

The `name` *attribute information item* has the following Infoset properties:

- A [local name] of `name` .
- A [namespace name] which has no value.

The type of the `name` *attribute information item* is *xs:NCName*.

2.14.2.2 `binding` *attribute information item with* `endpoint` *[owner]*

The `binding` *attribute information item* refers, by QName, to a Binding component

The `binding` *attribute information item* has the following Infoset properties:

- A [local name] of `binding`
- A [namespace name] which has no value

The type of the `binding` *attribute information item* is *xs:QName*.

2.14.2.3 Endpoint extension elements

Endpoint extension elements are used to provide information specific to a particular endpoint in a server. The semantics of such *element information item*s are defined by the specification for those *element information item*s. Such specifications are expected to annotate the Endpoint component with additional properties and specify the mapping between those properties and the XML representation.

2.14.3 Mapping Endpoint's XML Representation to Component Properties

The mapping between the properties of the Endpoint component (see **2.14.1 The Endpoint Component**) and the XML Representation of the `endpoint` *element information item* (see **2.14.2 XML Representation of Endpoint Component**) is as described in Table 2-15.

Table 2-15. Mapping between Endpoint Component Properties and XML Representation

Property	Mapping
{name}	The actual value of the `name` *attribute information item*
{binding}	The Binding component resolved to by the actual value of the `binding` *attribute information item*.

2.15 Equivalence of Components

Two components of the same type are considered equivalent if, for each property, the value in the first component is the same as the value in the

second component.

With respect to top-level components (Interfaces, Bindings and Services) this effectively translates to name-based equivalence given the constraints on names. That is, given two top-level components of the same type, if their {name} properties have the same value and their {target namespace} properties have the same values then the two components are in fact, the same component.

2.16 Symbol Spaces

This specification defines three symbol spaces, one for each top-level component type (Interface, Binding and Service).

Within a symbol space, all qualified names (that is, the combination of {name} and {target namespace} properties) are unique. Between symbol spaces, the combination of these two properties need not be unique. Thus it is perfectly coherent to have, for example, a binding and an interface that have the same name.

When XML Schema is being used as one of the type systems for a WSDL description, then six other symbol spaces also exist, one for each of: global element declarations, global attribute declarations, named model groups, named attribute groups, type definitions and key constraints, as defined by [*XML Schema: Structures*]. Other type systems may define additional symbol spaces.

2.17 QName resolution

In its serialized form WSDL makes significant use of references between components. Such references are made using the Qualified Name, or QName, of the component being referred to. QNames are a tuple, consisting of two parts; a namespace name and a local name. For example, in the case of an Interface component, the namespace name is represented by the {namespace name} property and the local name is represented by the {name} property.

QName references are resolved by looking in the appropriate property of the Definitions component. For example, to resolve a QName of an interface (as referred to by the `interface` *attribute information item* on a binding), the {interfaces} property of the Definitions component would be inspected.

If the appropriate property of the Definitions component does not contain a component with the required QName then the reference is a broken reference. It is an error for a Definitions component to have such broken references.

2.18 Comparing URIs

This specification uses absolute URIs to identify several components (for example, features and properties) and components characteristics (for example, operation message exchange patterns and styles). When such absolute URIs are being compared to determine equivalency (see **2.15 Equivalence of Components**) the URIs MUST be compared character-by-character as indicated in [*TAG URI FINDING*].

3. Types

```
<definitions>
  <types>
    <documentation />?
    [extension elements]*
  </types>
</definitions>
```

At the abstract level, the {element declarations} property of **2.1.1 The Definitions Component** is a collection of imported and embedded schema components. By design, WSDL supports any schema language for which the syntax and semantics of import (i.e., the ability to import some schema by reference) or embed (i.e., the ability to embed a schema directly into another document) have been defined. However, only the XML Schema implementation is defined in this specification. Instances of WSDL (i.e., WSDL documents) MAY require support for an alternative schema language by using the standard `wsdl:required` *attribute information item* (any imported or embedded XML Schema *element information item*s may be regarded as having this *attribute information item* set).

> **Note:**
>
> Support for the W3C XML Schema Description Language [*XML Schema: Structures*],[*XML Schema: Datatypes*] is required of all processors.

The schema components contained in the {element declarations} properties of **2.1.1 The Definitions Component** provide the type system used for Message Reference and Interface Fault components. Message Reference components indicate their structure and content by using the standard *attribute information item*s `element` , or for alternate schema languages in which these concepts do not map well, by using alternative *attribute information item* extensions. Interface Fault components behave similarly. Such extensions should define how they reference type system components. Such type system components MAY appear in additional collection properties on **2.1.1 The Definitions Component**.

The `types` *element information item* encloses data type definitions used to

define messages and has the following Infoset properties:

- A [local name] of `types` .
- A [namespace name] of "http://www.w3.org/2004/03/wsdl".
- Zero or more namespace qualified *attribute information item*s in its [attributes] property. The [namespace name] property of such *attribute information item*s MUST NOT be http://www.w3.org/2004/03/wsdl
- Zero or more *element information item*s amongst its [children] as follows:
  - An OPTIONAL `documentation` *element information item* (see **5. Documentation**) in its [children] property.
  - Zero or more *element information item*s from among the following, in any order:
    - `xs:import` *element information item*s
    - `xs:schema` *element information item*s
    - Other namespace qualified *element information item*s whose namespace is NOT http://www.w3.org/2004/03/wsdl

3.1 Using W3C XML Schema Description Language

XML Schema MAY be used as the schema language via import or embedding. Each method defines a different *element information item* for use within a `types` *element information item*. All processors MUST support XML Schema type definitions.

A WSDL description MUST NOT refer to XML Schema components in a given namespace unless an `xs:import` and/or `xs:schema` statement for that namespace is present. That is, using the `xs:import` and/or `xs:schema` constructs is a necessary condition for making XML Schema components available to a WSDL description.

3.1.1 Importing XML Schema

Importing an XML Schema uses the syntax and semantics of the `xs:import` mechanism defined by XML Schema [*XML Schema: Structures*],[*XML Schema: Datatypes*], with some additional restrictions. The schema components defined in the imported schema are available for reference by QName (see **2.17 QName resolution**). Note that only components defined in the schema itself and components included by it via `xs:include` are available to WSDL. Specifically, components that the schema imports via `xs:import` are NOT available to WSDL.

A child *element information item* of the `types` *element information item* is defined with the Infoset properties as follows:

- A [local name] of "import".
- A [namespace name] of ""http://www.w3.org/2001/XMLSchema"".
- One or two *attribute information item*s as follows:
    - A REQUIRED `namespace` *attribute information item* as described below.
    - An OPTIONAL `schemaLocation` *attribute information item* as described below.

3.1.1.1 `namespace` *attribute information item*

The `namespace` *attribute information item* defines the namespace of the element declarations imported from the referenced schema. The referenced schema MUST contain a `targetNamespace` *attribute information item* on its `xs:schema` *element information item* and the values of these two *attribute information item*s MUST be identical. It is an error to import a schema that does not have a `targetNamespace` *attribute information item* on its `xs:schema` *element information item*. Such schemas must first be included (using `xs:include`) in a schema that contains a `targetNamespace` *attribute information item* on its `xs:schema` *element information item*, which can then be either imported or inlined in the WSDL document.

The `namespace` *attribute information item* has the following Infoset properties:

- A [local name] of namespace
- A [namespace name] which has no value.

The type of the `namespace` *attribute information item* is *xs:anyURI*.

3.1.1.2 `schemaLocation` *attribute information item*

The `schemaLocation` *attribute information item*, if present, provides a hint to the processor as to where the schema may be located. Caching and cataloging technologies may provide better information than this hint. The `schemaLocation` *attribute information item* has the following infoset properties:

- A [local name] of schemaLocation.
- A [namespace name] which has no value.

The type of the `schemaLocation` *attribute information item* is *xs:anyURI.*

3.1.2 Embedding XML Schema

Embedding an XML schema uses the existing top-level `xs:schema` *element information item* defined by XML Schema [*XML Schema: Structures*]. It may be viewed as simply cutting and pasting an existing, stand-alone schema, to a location inside the types *element information item*.

The schema components defined in the embedded schema are available to WSDL for reference by QName (see **2.17 QName resolution**). Note that only components defined in the schema itself and components included by it via `xs:include` are available to WSDL. Specifically components that the schema imports via `xs:import` are NOT available to WSDL.

Similarly, components defined in an embedded XML schema are NOT automatically made available to a WSDL description that imported (using `wsdl:import`) the description that embeds the schema (see **4.2 Importing Descriptions** for more details). For this reason, it is recommended that XML schema documents intended to be shared across several WSDL descriptions be placed in separate documents and imported using `xs:import` , rather than embedded inside a WSDL document.

Inside an embedded XML schema, the `xs:import` and `xs:include` *element information item*s MAY be used to refer to other XML schemas embedded in the same WSDL description, provided that an appropriate value is specified for their `schemaLocation` *attribute information item*s. The semantics of such *element information item*s are governed solely by the XML Schema specification [*XML Schema: Structures*].

The `xs:schema` *element information item* has the following Infoset properties:

- A [local name] of schema.
- A [namespace name] of "http://www.w3.org/2001/XMLSchema".
- A REQUIRED `targetNamespace` *attribute information item*, amongst its [attributes] as described below.
- Additional OPTIONAL *attribute information item*s as specified for the `xs:schema` *element information item* by the XML Schema specification.
- Zero or more child *element information item*s as specified for the `xs:schema` *element information item* by the XML Schema specification.

3.1.2.1 `targetNamespace` *attribute information item*

The `targetNamespace` *attribute information item* defines the namespace of the element declarations embedded in its [owner] `xs:schema` *element information item*. WSDL modifies the XML Schema definition of the `xs:schema` *element information item* to make this *attribute information item* required. The `targetNamespace` *attribute information item* has the following infoset properties:

- A [local name] of targetNamespace.
- A [namespace name] which has no value.

The type of the `targetNamespace` *attribute information item* is *xs:anyURI*.

3.1.3 References to Element Declarations

Whether embedded or imported, the element declarations present in a schema may be referenced from a Message Reference or Interface Fault component.

A named, global `xs:element` declaration may be referenced from the `element` *attribute information item* of an `input` , `output` or `fault` *element information item*. The QName is constructed from the `targetNamespace` of the schema and the value of the `name` *attribute information item* of the `xs:element` *element information item*. An `element` *attribute information item* MUST NOT refer to a global `xs:simpleType` or `xs:complexType` definition.

3.2 Using Other Schema Languages

Since it is unreasonable to expect that a single schema language can be used to describe all possible Message Reference and Fault component contents and their constraints, WSDL allows alternate schema languages to be specified via extensibility elements. An extensibility *element information item* MAY appear under the `types` *element information item* to identify the schema language employed, and to locate the schema instance defining the grammar for Message Reference and Interface Fault components. Depending upon the schema language used, an *element information item* MAY be defined to allow embedding, if and only if the schema language can be expressed in XML.

A specification of extension syntax for an alternative schema language MUST include the declaration of an *element information item*, intended to appear as a child of the `wsdl:types` *element information item*, which references, names, and locates the schema instance (an "import" *element information item*). The extension specification SHOULD, if necessary, define additional properties of **2.1.1 The Definitions Component** (and extensibility attributes) to hold the components of the referenced type system. It is expected that additional extensibility attributes for Message Reference and Interface Fault components will also be defined, along with a mechanism for resolving the values of those attributes to a particular imported type system component.

See **E. Examples of Specifications of Extension Elements for Alternative Schema Language Support.** for examples of using other schema languages. These examples reuse the {element declarations} property of **2.1.1 The**

Definitions Component and the `element` *attribute information item*s of the `wsdl:input` , `wsdl:output` and `wsdl:fault` *element information item*s.

4. Modularizing WSDL descriptions

This specification provides two mechanisms, described in this section, for modularizing WSDL descriptions. These mechanisms help to make WSDL descriptions clearer by allowing separation of the various components of a description. Such separation could be performed according to the level of abstraction of a given set of components, or according to the namespace affiliation required of a given set of components or according to some other grouping such as application applicability.

Both mechanisms work at the level of WSDL components and NOT at the level of XML Information Sets or XML 1.0 serializations.

4.1 Including Descriptions

```
<definitions>
  <include
        location="xs:anyURI" >
    <documentation />?
  </include>
</definitions>
```

The WSDL `include` *element information item* allows for the separation of different components of a service definition, belonging the same target namespace, into independent WSDL documents which can be merged as needed.

The WSDL `include` *element information item* is modeled after the XML Schema `include` *element information item* (see [*XML Schema: Structures*], section 4.2.3 "References to schema components in the same namespace"). Specifically, it can be used to include components from WSDL descriptions that share a target namespace with the including description. Components in *directly* included descriptions become part of the component model of the including description. Directly included means that component inclusion is not transitive; components included by one of the included documents are *not* available to the original including document unless the are included directly by that document. The included components can be referenced by QName. Note that because all WSDL descriptions have a target namespace, no-namespace includes (sometimes known as "chameleon includes") never occur in WSDL.

A mutual include is direct inclusion by one WSDL document of another WSDL document which includes the first. A circular include achieves the same effect with greater indirection (WSDL A includes WSDL B includes WSDL A, for

instance). Multiple inclusion of a single WSDL document resolves to a single set of components. Mutual, multiple, and circular includes are explicitly permitted, and do not represent multiple redefinitions of the same components. Multiple inclusion of a single WSDL document has the same meaning as including it only once. Processors are encouraged to keep track of the source of component definitions, so that multiple, mutual, and circular includes do not require establishing identity on a component-by-component basis.

The `include` *element information item* has:

- A [local name] of `include` .
- A [namespace name] of "http://www.w3.org/2004/03/wsdl".
- One or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `location` *attribute information item* as described below in **4.1.1 location attribute information item with include [owner]**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item* amongst its [children], as follows:
  - An optional `documentation` *element information item* (see **5. Documentation**).
  - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".

4.1.1 `location` *attribute information item* with `include` [owner]

The `location` *attribute information item* has the following Infoset properties:

- A [local name] of `location` .
- A [namespace name] which has no value.

A `location` *attribute information item* is of type `xs:anyURI` . Its actual value is the location of some information about the namespace identified by the `targetNamespace` *attribute information item* of the containing `definitions` *element information item*.

If the URI indicated by `location` is not dereferenceable or does not resolve to a WSDL document then the processor MUST fail immediately. That is, `include` elements MUST be processed immediately by WSDL processors.

The actual value of the `targetNamespace` *attribute information item* of the included WSDL document MUST match the actual value of the `targetNamespace` *attribute information item* of the `definitions` *element information item* which is the [parent] of the `include` *element information item*.

4.2 Importing Descriptions

```
<definitions>
  <import
        namespace="xs:anyURI"
        location="xs:anyURI"? >
    <documentation />?
  </import>
</definitions>
```

The WSDL `import` *element information item*, like the `include` *element information item* (see **4.1 Including Descriptions**) also allows for the separation of the different components of a WSDL description into independent descriptions, but in this case with different target namespaces, which can be imported as needed. This technique helps writing clearer WSDL descriptions by separating the definitions according to their level of abstraction, and maximizes reusability.

The WSDL `import` *element information item* is modeled after the XML Schema `import` *element information item* (see [*XML Schema: Structures*], section 4.2.3 "References to schema components across namespaces"). Specifically, it can be used to import components from WSDL descriptions that do not share a target namespace with the importing document. Components in *directly* imported descriptions are part of the component model of the importing description. Directly imported means that component importation is not transitive; components imported by one of the imported documents are *not* available to the original importing document unless the are imported directly by that document. The imported components can be referenced by QName.

Using the `import` construct is a necessary condition for making components from another namespace available to a WSDL description. That is, a WSDL description MUST NOT refer to components in a namespace other than the target namespace unless an import statement for that namespace is present. The same considerations apply to schemas embedded in an imported WSDL description (see **3.1.2 Embedding XML Schema**). More explicitely, components defined by an XML schema document embedded inside an imported WSDL description are NOT made available to the importer unless the latter contains an explicit `xs:import` statement to that purpose.

This specification DOES NOT preclude repeating the `import` *element*

information item for the same value of the `namespace` *attribute information item* as long as they provide different values for the `location` *attribute information item*. Repeating the `import` *element information item* for the same `namespace` value MAY be used as a way to provide alternate locations to find information about a given namespace.

Furthermore, this specification DOES NOT require the `location` *attribute information item* to be dereferenceable. If it is not dereferenceable then no information about the imported namespace is provided by that `import` *element information item*. It is possible that such lack of information results in QNames in other parts of a WSDL Definitions component to become broken references (see **2.17 QName resolution**). Such broken references are not errors of the `imports` *element information item* but rather QName resolution errors which must be detected as described in **2.17 QName resolution**.

The `import` *element information item* has the following Infoset properties:

- A [local name] of `import` .
- A [namespace name] of "http://www.w3.org/2004/03/wsdl".
- Two or more *attribute information item*s amongst its [attributes] as follows:
  - A REQUIRED `namespace` *attribute information item* as described below in **4.2.1 namespace attribute information item**.
  - An OPTIONAL `location` *attribute information item* as described below in **4.2.2 location attribute information item with import [owner]**.
  - Zero or more namespace qualified *attribute information item*s. The [namespace name] of such *attribute information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".
- Zero or more *element information item* amongst its [children], as follows:
  - An optional `documentation` *element information item* (see **5. Documentation**).
  - Zero or more namespace-qualified *element information item*s amongst its [children]. The [namespace name] of such *element information item*s MUST NOT be "http://www.w3.org/2004/03/wsdl".

4.2.1 `namespace` *attribute information item*

The `namespace` *attribute information item* has the following Infoset properties:

- A [local name] of `namespace` .

- A [namespace name] which has no value.

The `namespace` *attribute information item* is of type `xs:anyURI` . Its actual value indicates that the containing WSDL document MAY contain qualified references to WSDL definitions in that namespace (via one or more prefixes declared with namespace declarations in the normal way). This value MUST NOT match the actual value of the enclosing WSDL document `targetNamespace` *attribute information item*. If the import statement results in the import of a WSDL document then the actual value of the `namespace` *attribute information item* MUST be identical to the actual value of the imported WSDL document's `targetNamespace` *attribute information item*.

4.2.2 `location` *attribute information item* with `import` [owner]

The `location` *attribute information item* has the following Infoset properties:

- A [local name] of `location` .
- A [namespace name] which has no value.

The `location` *attribute information item* is of type `xs:anyURI` . Its actual value is the location of some information about the namespace identified by the `namespace` *attribute information item*.

The `location` *attribute information item* is optional. This allows WSDL components to be constructed from information other than serialized XML 1.0. It also allows the development of WSDL processors that have *a priori* (i.e., built-in) knowledge of certain namespaces.

5. Documentation

```
<documentation>
  [extension elements]*
</documentation>
```

WSDL uses the optional `documentation` *element information item* as a container for human readable and/or machine processable documentation. The content of the *element information item* is arbitrary *character information items* and *element information items* ("mixed" content in XML Schema[*XML Schema: Structures*]). The `documentation` *element information item* is allowed inside any WSDL *element information item*.

The `documentation` *element information item* has:

- A [local name] of `documentation` .
- A [namespace name] of "http://www.w3.org/2004/03/wsdl".

- Zero or more *attribute information items* in its [attributes] property.
- Zero or more child *element information items* in its [children] property.
- Zero or more *character information items* in its [children] property.

6. Language Extensibility

The schema for WSDL has a two-part extensibility model based on namespace-qualified elements and attributes. The extension is identified by the qname consisting of its namespace URI and its element name. The meaning of the extension SHOULD be defined (directly or indirectly) in a document that is available at its namespace URI.

6.1 Element based Extensibility

WSDL allows extensions to be defined in terms of *element information items*. Where indicated herein, WSDL allows namespace-qualified *element information items* whose [namespace name] is NOT "http://www.w3.org/2004/03/wsdl" to appear among the [children] of specific *element information items* whose [namespace name] is "http://www.w3.org/2004/03/wsdl". Such *element information items* MAY be used to annotate WSDL constructs such as interface, operation, etc.

It is expected that extensions will want to add to the existing properties of components in the component model. The specification for an extension *element information item* should include definitions of any such properties and the mapping between the XML representation of the extension and the properties in the component model.

The WSDL schema also defines a base type for use by extensibility elements. Example 6-1 shows the type definition. The use of this type as a base type is optional. The element declarations which serve as the heads of the defined substitution groups are all of type "xs:anyType".

Extensibility elements are commonly used to specify some technology-specific binding. They allow innovation in the area of network and message protocols without having to revise the base WSDL specification. WSDL recommends that specifications defining such protocols also define any necessary WSDL extensions used to describe those protocols or formats.

Example 6-1. Base type for extensibility elements

```
<xs:complexType name='ExtensibilityElement'
abstract='true' >
  <xs:attribute ref='wsdl:required' use='optional' />
</xs:complexType>
```

6.1.1 Mandatory extensions

Extension elements can be marked as mandatory by annotating them with a `wsdl:required` *attribute information item* (see **6.1.2 required attribute information item**) with a value of "true". A mandatory extension is an extension that MAY change the meaning of the element to which it is attached, such that the meaning of that element is no longer governed by this specification. Instead, the meaning of an element containing a mandatory extension is governed by the meaning of that extension. Thus, the definition of the element's meaning is *delegated* to the specification that defines the extension.

An extension that is NOT marked as mandatory MUST NOT invalidate the meaning of any part of the WSDL document. Thus, a NON-mandatory extension merely provides additional description of capabilities of the service. Furthermore, any extension that is NOT marked as mandatory and which is NOT understood, MUST be ignored. Any NOT understood extension attributes MUST be ignored as this specification does not provide a mechanism to mark extension attributes as being required.

> **Note:**
>
> A mandatory extension is considered mandatory because it has the ability to change the meaning of the element to which it is attached. Thus, the meaning of the element may not be fully understood without understanding the attached extension. A NON-mandatory extension, on the other hand, can be safely ignored without danger of misunderstanding the rest of the WSDL document.

6.1.2 `required` *attribute information item*

WSDL provides a global *attribute information item* with the following Infoset properties:

- A [local name] of `required` .
- A [namespace name] of "http://www.w3.org/2004/03/wsdl".
- A [specified] property with a value of "true".

The type of the `required` *attribute information item* is *xs:boolean*.

6.2 Attribute-based Extensibility

WSDL allows qualified *attribute information item*s whose [namespace name] is NOT "http://www.w3.org/2004/03/wsdl" to appear on any *element*

information item whose namespace name IS "http://www.w3.org/2004/03/wsdl". Such *attribute information item*s can be used to annotate WSDL constructs such as interfaces, bindings, etc.

WSDL does not provide a mechanism for marking extension *attribute information item*s as mandatory.

6.3 Extensibility Semantics

As indicated above, it is expected that the presence of extensibility elements and attributes will result in additional properties appearing in the component model.

The presence of an optional extensibility element or attribute MAY therefore augment the semantics of a WSDL document in ways that do not invalidate the existing semantics. However, the presence of a mandatory extensibility element MAY alter the semantics of a WSDL document in ways that invalidate the existing semantics.

> **Note:**
>
> Authors of extensibility elements should avoid altering the existing semantics in ways that are likely to confuse users.

7. Locating WSDL Documents

As an XML vocabulary, WSDL documents or fragments or references to WSDL components (via QNames) MAY appear within other XML documents. In such scenarios it could be necessary to provide some hints on where additional WSDL information for a given namespace can be found in order to help with QName resolution **2.17 QName resolution**.

This specification defines a global attribute, `wsdlLocation` in the namespace "http://www.w3.org/2004/03/wsdl-instance" for this purpose (hereafter referred to as "wsdli:wsdlLocation"). This global attribute MAY appear on any XML element which allows attributes from other namespaces to occur. It MUST NOT appear on a `wsdl:definitions` element or any of its children/descendants.

7.1 `wsdli:wsdlLocation` *attribute information item*

WSDL provides a global *attribute information item* with the following Infoset properties:

- A [local name] of `wsdlLocation` .
- A [namespace name] of "http://www.w3.org/2004/03/wsdl-instance".

The type of the `wsdlLocation` *attribute information item* is a list *xs:anyURI*. Its actual value MUST be a list of pairs of URIs; where the first URI of a pair, which MUST be an absolute URI as defined in [*IETF RFC 2396*], indicates a WSDL namespace name, and, the second a hint as to the location of a WSDL document defining WSDL components for that namespace name. The second URI of a pair MAY be abolute or relative.

8. Conformance

8.1 Document Conformance

An *element information item* whose namespace name is "http://www.w3.org/2004/03/wsdl" and whose local part is `definitions` conforms to this specification if conforms to the XML Schema for that element as defined by this specification family and additionally adheres to all the constraints contained in this specification.

8.2 XML Information Set Conformance

This specification conforms to the [*XML Information Set*]. The following information items MUST be present in the input infosets to enable correct processing of WSDL documents:

- *Document Information Items* with *children* and *base URI* properties.
- *Element Information Items* with *namespace name*, *local name*, *children*, *attributes*, *base URI* and *parent* properties.
- *Attribute Information Items* with *namespace name*, *local name* and *normalized value* properties.
- *Character Information Items* with *character code*, *element content whitespace* and *parent* properties.

8.3 Processor Conformance

This section defines a class of conformant WSDL processors that are intended to act on behalf of a party that wishes to make use of a Web service (i.e., the requester entity or requester agent), rather than the party that implements the Web service (i.e., the provider entity or provider agent).

An extension element is said to be *processed* if the WSDL processor decides (through whatever means) that its parent (an *element information item* in the "http://www.w3.org/2004/03/wsdl" namespace) will be processed. Note that it is possible for WSDL processors to process only a subset of a given WSDL document. For instance, a tool may wish to focus on interfaces and operations only, and ignore bindings.

A conformant WSDL processor MUST adhere to the following rules:

- Except as noted below for mandatory extensions, a conformant WSDL processor MUST accept any legal WSDL document as defined by this specification.
- A conformant WSDL processor MUST fault if a portion of a WSDL document is illegal according to this specification and the WSDL processor attempts to process that portion.
- A conformant WSDL processor MUST support at least XML Schema as a type system language.
- A conformant WSDL processor MUST fail if it processes an element containing a wsdl:include statement having a URI that is not dereferenceable to a legal WSDL document.
- If a mandatory extension (i.e., a mandatory element, feature or property) is processed, a conformant WSDL processor MUST either agree to fully abide by all the rules and semantics signaled by that extension, or immediately cease processing (fault). In particular, if the WSDL processor does not recognize the extension, it MUST fault. If the WSDL processor recognizes the extension, and determines that the extension in question is incompatible with any other aspect of the document (including other required extensions), it MUST fault.
- A conformant WSDL processor MAY safely ignore a NON-mandatory extension that it does not recognize or that it does not choose to implement.

9. XML Syntax Summary (Non-Normative)

```
<definitions targetNamespace="xs:anyURI" >
  <documentation />?

  <import namespace="xs:anyURI" location="xs:anyURI"? >
    <documentation />?
  </import>*

  <include location="xs:anyURI" >
    <documentation />?
  </include>*

  <types>
    <documentation />?
  </types>

  <interface name="xs:NCName" extends="list of xs:
```

```
QName"? styleDefault="xs:anyURI"? >
    <documentation />?

    <fault name="xs:NCName" element="xs:QName"? >
      <documentation />?
    </fault>*

    <operation name="xs:NCName" pattern="xs:anyURI"
style="xs:anyURI"? safe="xs:boolean"? >
      <documentation />?

      <input messageLabel="xs:NCName"? element="union
of xs:QName, xs:Token"? >
        <documentation />?
      </input>*

      <output messageLabel="xs:NCName"? element="union
of xs:QName, xs:Token"? >
        <documentation />?
      </output>*

      <infault ref="xs:QName" messageLabel="xs:
NCName"? >
        <documentation />?
      </infault>*

      <outfault ref="xs:QName" messageLabel="xs:
NCName"? >
        <documentation />?
      </outfault>*

      <feature ... />*

      <property ... />*
    </operation>*

    <feature uri="xs:anyURI" required="xs:boolean"? >
      <documentation />?
    </feature>*

    <property uri="xs:anyURI" required="xs:boolean"? >
      <documentation />?

      <value> xs:anyType </value>?

      <constraint> xs:QName </constraint>?
```

```
      </property>*
    </interface>*

    <binding name="xs:NCName" interface="xs:QName"? >
      <documentation />?

      <fault ref="xs:QName" >
        <documentation />?
      </fault>*

      <operation ref="xs:QName" >
        <documentation />?

        <input messageLabel="xs:NCName"? >
          <documentation />?
        </input>*

        <output messageLabel="xs:NCName"? >
          <documentation />?
        </output>*

        <feature ... />*

        <property ... />*
      </operation>*

      <feature ... />*

      <property ... />*
    </binding>*

    <service name="xs:NCName" interface="xs:QName"
      <documentation />?

      <endpoint name="xs:NCName" binding="xs:QName" >
        <documentation />?
      </endpoint>*
    </service>*
  </definitions>
```

10. References

10.1 Normative References

[IETF RFC 2119]

Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, Author. Internet Engineering Task Force, June 1999. Available at http://www.ietf.org/rfc/rfc2119.txt.

[IETF RFC 2396]

Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, Authors. Internet Engineering Task Force, August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt.

[XML 1.0]

Extensible Markup Language (XML) 1.0 (Second Edition), T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, Editors. World Wide Web Consortium, 10 February 1998, revised 6 October 2000. This version of the XML 1.0 Recommendation is http://www.w3.org/TR/2000/REC-xml-20001006. The latest version of XML 1.0 is available at http://www.w3.org/TR/REC-xml.

[XML Information Set]

XML Information Set, J. Cowan and R. Tobin, Editors. World Wide Web Consortium, 24 October 2001. This version of the XML Information Set Recommendation is http://www.w3.org/TR/2001/REC-xml-infoset-20011024. The latest version of XML Information Set is available at http://www.w3.org/TR/xml-infoset.

[XML Namespaces]

Namespaces in XML, T. Bray, D. Hollander, and A. Layman, Editors. World Wide Web Consortium, 14 January 1999. This version of the XML Information Set Recommendation is http://www.w3.org/TR/1999/REC-xml-names-19990114. The latest version of Namespaces in XML is available at http://www.w3.org/TR/REC-xml-names.

[XML Schema: Structures]

XML Schema Part 1: Structures, H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, Editors. World Wide Web Consortium, 2 May 2001. This version of the XML Schema Part 1 Recommendation is http://www.w3.org/TR/2001/REC-xmlschema-1-20010502. The latest version of XML Schema Part 1 is available at http://www.w3.org/TR/xmlschema-1.

[XML Schema: Datatypes]

XML Schema Part 2: Datatypes, P. Byron and A. Malhotra, Editors. World Wide Web Consortium, 2 May 2001. This version of the XML Schema Part 2 Recommendation is http://www.w3.org/TR/2001/REC-xmlschema-2-20010502. The latest version of XML Schema Part 2 is available at http://www.w3.org/TR/xmlschema-2.

[RFC 3023]

IETF "RFC 3023: XML Media Types", M. Murata, S. St. Laurent, D. Kohn, July 1998. (See *http://www.ietf.org/rfc/rfc3023.txt*.)

[WSDL MediaType]

IETF Internet Draft "The 'application/wsdl+xml' media type", @@@.

(Work to be done once we have consensus on the media type).

[WSDL 2.0 Bindings]

Web Services Description (WSDL) Version 1.2: Bindings, J-J. Moreau, J. Schlimmer, Editors. World Wide Web Consortium, 11 June 2003. This version of the "Web Services Description Version 2.0: Bindings" Specification is available at http://www.w3.org/TR/2003/WD-wsdl12-bindings-20030611. The latest version of "Web Services Description Version 1.2: Bindings" is available at http://www.w3.org/TR/wsdl12-bindings.

[WSDL 2.0 Message Exchange Patterns]

Web Services Description Language (WSDL) Version 2.0 Part 2: Message Exchange Patterns, M. Gudgin, A. Lewis, and J. Schlimmer, Editors. World Wide Web Consortium, 26 March 2004. This version of the "Web Services Description Version 2.0: Message Exchange Patterns" Specification is available at http://www.w3.org/TR/2004/WD-wsdl20-patterns-20040326. The latest version of "Web Services Description Language (WSDL) Version 2.0 Part 2: Message Exchange Patterns" is available at http://www.w3.org/TR/wsdl20-patterns.

[WSDL 2.0 RDF Mapping]

To be written.

[TAG URI FINDING]

TAG Finding on URI Comparisn, X. Foo, Y. Bar, Authors. W3C Technical Architecture Group, Month, Year. Available at http://www.w3.org/2001/tag/findings/ZZZZ.

[Web Architecture]

Architecture of the World Wide Web, First Edition, Ian Jacobs, Editor. W3C Technical Architecture Group, December, 2003. Available at http://www.w3.org/TR/2003/WD-webarch-20031209/.

10.2 Informative References

[IETF RFC 2045]

Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N. Freed, N. Borenstein, Authors. Internet Engineering Task Force, November 1996. Available at http://www.ietf.org/rfc/rfc2045.txt.

[IETF RFC 2616]

Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Authors. Internet Engineering Task Force, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt.

[SOAP 1.1]

Simple Object Access Protocol (SOAP) 1.1, D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, D.

Winer, Editors. World Wide Web Consortium, 8 May 2000. This version of the Simple Object Access Protocol 1.1 Note is http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[SOAP 1.2 Part 1: Messaging Framework]

SOAP Version 1.2 Part 1: Messaging Framework, M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Frystyk Nielsen, Editors. World Wide Web Consortium, 24 June 2003. This version of the "SOAP Version 1.2 Part 1: Messaging Framework" Recommendation is http://www.w3.org/TR/2003/REC-soap12-part1-20030624/. The latest version of "SOAP Version 1.2 Part 1: Messaging Framework" is available at http://www.w3.org/TR/soap12-part1/.

[XML Linking]

XML Linking Language (XLink) Version 1.0, S. DeRose, E. Maler, D. Orchard, Editors. World Wide Web Consortium, 27 June 2001. This version of the XML Linking Language 1.0 Recommendation is http://www.w3.org/TR/2001/REC-xlink-20010627. The latest version of XML Linking Language 1.0 is available at http://www.w3.org/TR/xlink.

[WSDL 1.1]

Web Services Description Language (WSDL) 1.1, E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, Authors. World Wide Web Consortium, 15 March 2002. This version of the Web Services Description Language 1.1 Note is http://www.w3.org/TR/2001/NOTE-wsdl-20010315. The latest version of Web Services Description Language 1.1 is available at http://www.w3.org/TR/wsdl.

[WSDL 2.0 Primer]

Web Services Description (WSDL) Version 2.0: Primer, K. Sankar, K. Liu, D. Booth, Editors. World Wide Web Consortium, 26 March 2004. The editors' version of the Web Services Description Version 2.0: Primer document is available from http://www.w3.org/2002/ws/desc/.

[WSD Requirements]

Web Services Description Requirements, J. Schlimmer, Editor. World Wide Web Consortium, 28 October 2002. This version of the Web Services Description Requirements document is http://www.w3.org/TR/2002/WD-ws-desc-reqs-20021028. The latest version of Web Services Description Requirements is available at http://www.w3.org/TR/ws-desc-reqs.

[XPointer Framework]

XPointer Framework,Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh, Editors. World Wide Web Consortium, 22 November 2002. This version of the XPointer Framework Proposed Recommendation is http://www.w3.org/TR/2003/REC-xptr-framework-20030325/ The latest version of XPointer Framework is available at http://www.w3.org/TR/xptr-framework/.

A. The "application/wsdl+xml" Media Type

Editorial note: JJM	20021107
This was lifted from the SOAP 1.2 specification, and needs to be edited to reflect WSDL's own requirements. For example, the WG has not reached consensus on whether to use "text/xml", "text/wsdl+xml" or "application/wsdl +xml".	

This appendix defines the "application/wsdl+xml" media type which can be used to describe WSDL 2.0 documents serialized as XML. It is referenced by the corresponding IANA registration document [*WSDL MediaType*].

A.1 Registration

MIME media type name:

application

MIME subtype name:

wsdl+xml

Required parameters:

none

Optional parameters:

charset

This parameter has identical semantics to the charset parameter of the "application/xml" media type as specified in [*RFC 3023*].

Encoding considerations:

Identical to those of "application/xml" as described in [*RFC 3023*], section 3.2, as applied to the WSDL document infoset.

Security considerations:

See section **A.2 Security considerations**.

Interoperability considerations:

There are no known interoperability issues.

Published specification:

This document and [*WSDL 2.0 Bindings*].

Applications which use this media type:

No known applications currently use this media type.

Additional information:

File extension:

WSDL documents are not required or expected to be stored as

files.

Fragment identifiers:

Either a syntax identical to that of "application/xml" as described in [*RFC 3023*], section 5 or the syntax defined in [*WSDL 2.0 RDF Mapping*].

Base URI:

As specified in [*RFC 3023*], section 6.

Macintosh File Type code:

TEXT

Person and email address to contact for further information:

@@@ <@@@@>

Intended usage:

COMMON

Author/Change controller:

The WSDL 2.0 specification set is a work product of the World Wide Web Consortium's Web Service Description Working Group. The W3C has change control over these specifications.

A.2 Security considerations

Editorial note: JJM	20021107
Are there any security considerations other than the standard ones.	

This media type uses the "+xml" convention, it shares the same security considerations as described in [*RFC 3023*], section 10.

B. Acknowledgements (Non-Normative)

This document is the work of the W3C Web Service Description Working Group.

Members of the Working Group are (at the time of writing, and by alphabetical order): Mike Ballantyne (Electronic Data Systems), David Booth (W3C), Allen Brookes (Rogue Wave Softwave), Roberto Chinnici (Sun Microsystems), Glen Daniels (Sonic Software), Alan Davies (SeeBeyond), Mike Davoren (W. W. Grainger), Paul Downey (British Telecommunications), Youenn Fablet (Canon), Yaron Goland (BEA), Martin Gudgin (Microsoft Corporation), Hugo Haas (W3C), Hao He (The Thomson Corporation), Tom Jordahl (Macromedia), Jacek Kopecky (Systinet), Dan Kulp (IONA Technologies), Sandeep Kumar (Cisco Systems), Amelia Lewis (TIBCO Software, Inc.), Kevin Canyang Liu (SAP), Michael Mahan (Nokia), Jonathan Marsh

(Microsoft Corporation), Mike McHugh (W. W. Grainger), Michael Mealling (Verisign), Ingor Melzer (DaimlerChrysler Research and Technology), Jeff Mischkinsky (Oracle Corporation), Dale Moberg (Cyclone Commerce), Jean-Jacques Moreau (Canon), David Orchard (BEA), Bijan Parsia (University of Maryland), Arthur Ryman (IBM), Waqar Sadiq (Electronic Data Systems), Adi Sakala (IONA Technologies), Jeffrey Schlimmer (Microsoft Corporation), Igor Sedukhin (Computer Associates), Sandra Swearingen (U.S. Department of Defense, U.S. Air Force), Bryan Thompson (Hicks & Associates), Jerry Thrasher (Lexmark), William Vambenepe (Hewlett-Packard Company), Asir Vedamuthu (webMethods, Inc.), Sanjiva Weerawarana (IBM), Ümit YalçÄ ±nalp (Oracle Corporation), Prasad Yendluri (webMethods, Inc.).

Previous members were: Lily Liu (webMethods, Inc.), Don Wright (Lexmark), Joyce Yang (Oracle Corporation), Daniel Schutzer (Citigroup), Dave Solo (Citigroup), Stefano Pogliani (Sun Microsystems), William Stumbo (Xerox), Stephen White (SeeBeyond), Barbara Zengler (DaimlerChrysler Research and Technology), Tim Finin (University of Maryland), Laurent De Teneuille (L'Echangeur), Johan Pauhlsson (L'Echangeur), Mark Jones (AT&T), Steve Lind (AT&T), Philippe Le Hégaret (W3C), Jim Hendler (University of Maryland), Dietmar Gaertner (Software AG), Michael Champion (Software AG), Don Mullen (TIBCO Software, Inc.), Steve Graham (Global Grid Forum), Steve Tuecke (Global Grid Forum).

The people who have contributed to discussions on www-ws-desc@w3.org are also gratefully acknowledged.

C. URI References for WSDL constructs (Non-Normative)

This appendix provides a syntax for URI references for named components found in a WSDL document. This includes the top level components: interface, binding and service and the subordinate components: operation, fault, and endpoint. The URI references are easy to understand and compare, while imposing no burden on the WSDL author.

C.1 WSDL URIs

There are two main cases for WSDL URIs:

- the URI of a WSDL document
- the URI of a WSDL namespace

The URI of a WSDL document can be dereferenced to give a resource representation that contributes component definitions to a single WSDL namespace. If the media type is set to the WSDL media type, then the

fragment identifiers can be used to identify the main components that are defined in the document.

However, in keeping with the recommendation in **2.1.1 The Definitions Component** that the namespace URI be dereferencible to a WSDL document, this appendix specifies the use of the namespace URI with the WSDL fragment identifiers to form a URI-reference.

C.2 Fragment Identifiers

The following fragment identifier syntax is compliant with the [*XPointer Framework*].

The URI in a URI-reference for a WSDL component is the {target namespace} property of either the component itself, in the case of interfaces, bindings, and services, or the {target namespace} property of an ancestor component. The URI provided by the {target namespace} property is combined with a fragment identifier, where the fragment identifier is constructed from the {name} property of the component and the {name} properties of its ancestors as a path according to Table C-1. In that table the first column gives the name of the WSDL component as the [local name] of the *element information item* that represents that construct in a WSDL document. Columns two and three populate the variables x and y respectively. These variables are then used to construct the fragment in column four.

Table C-1. Rules for determining fragments for WSDL constructs

Construct	x	y	Fragment
interface	{name} property of interface	n/a	`interface(x)`
operation	{name} property of operation	{name} property of parent interface	`operation(y/x)`
fault	{name} property of fault	{name} property of parent interface	`fault(y/x)`
binding	{name} property of binding	n/a	`binding(x)`
service	{name} property of service	n/a	`service(x)`
endpoint	{name} property of endpoint	{name} property of parent service	`endpoint(y/x)`

Note that the above rules are defined in terms of component properties rather the XML Infoset representation of the component model.

C.3 Extension Elements

WSDL has an open content model. It is therefore possible for an extension to define new components. The XPointer Framework scheme for components added by extensions is:

`extension(extension-namespace, extension-specific-syntax)`

where extension-namespace is the namespace that identifies the extension, e. g. for SOAP the namespace is http://www.w3.org/2003/06/wsdl/soap12, and extension-specific-syntax is defined by the extension. The owner of the extension must define any components contributed by the extension and a syntax for identifying them.

C.4 Example

Consider the following WSDL located at http://example.org/TicketAgent.wsdl:

Example C-1. URI References - Example WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
    targetNamespace="http://example.org/TicketAgent.
wsdl20"
    xmlns:xsTicketAgent="http://example.org/
TicketAgent.xsd"
    xmlns:wsdl="http://www.w3.org/2004/03/wsdl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation="http://www.w3.org/2004/03/
wsdl wsdl20.xsd">

    <wsdl:types>
        <xs:import schemaLocation="TicketAgent.xsd"
                   namespace="http://example.org/
TicketAgent.xsd" />
    </wsdl:types>

    <wsdl:interface name="TicketAgent">
        <wsdl:operation name="listFlights"
pattern="http://www.w3.org/@@@@/@@/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:
listFlightsRequest"/>
            <wsdl:output element="xsTicketAgent:
listFlightsResponse"/>
```

```
        </wsdl:operation>

        <wsdl:operation name="reserveFlight"
pattern="http://www.w3.org/@@@@/@@/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:
reserveFlightRequest"/>
            <wsdl:output element="xsTicketAgent:
reserveFlightResponse"/>
        </wsdl:operation>
    </wsdl:interface>
</wsdl:definitions>
```

Its conceptual elements have the following URI-references:

Example C-2. URI References - Example URIs

```
http://example.org/TicketAgent.wsdl20#interface
(TicketAgent)
http://example.org/TicketAgent.wsdl20#operation
(TicketAgent/listFlights)
http://example.org/TicketAgent.wsdl20#operation
(TicketAgent/reserveFlight)
```

D. Migrating from WSDL 1.1 to WSDL 2.0 (Non-Normative)

This section will attempt to document some of the migration concerns of going from WSDL 1.1 to WSDL 2.0. We do not claim that all migration problems will be addressed here.

D.1 Operation Overloading

WSDL 1.1 supported operation overloading and WSDL 2.0 removes it. This section will provide some rationale for it and provide hints on how to work around some scenarios.

D.2 PortTypes

Port types have been renamed to interfaces. We now have interface inheritance.

D.3 Ports

Ports have been renamed to endpoints.

E. Examples of Specifications of Extension Elements for Alternative Schema Language Support. (Non-Normative)

E.1 DTD

A DTD may be used as the schema language for WSDL. It may not be embedded; it must be imported. A namespace must be assigned. DTD types appear in the {element declarations} property of **2.1.1 The Definitions Component** and may be referenced from the `wsdl:input` , `wsdl:output` and `wsdl:fault` elements using the `element` *attribute information item*.

The prefix, dtd, used throughout the following is mapped to the namespace URI "http://www.example.org/dtd/".

The `dtd:import` *element information item* references an external Document Type Definition, and has the following infoset properties:

- A [local name] of import.
- A [namespace name] of "http://www.example.org/dtd".
- One or two *attribute information item*s, as follows:
  - A REQUIRED `namespace` *attribute information item* as described below.
  - An OPTIONAL `location` *attribute information item* as described below.

E.1.1 `namespace` *attribute information item*

The `namespace` *attribute information item* sets the namespace to be used with all imported element definitions described in the DTD. It has the following infoset properties:

- A [local name] of namespace.
- A [namespace name] which has no value.

The type of the `namespace` *attribute information item* is *xs:anyURI*.

The WSDL author should ensure that a prefix is associated with the namespace at the proper scope (probably document scope).

E.1.2 `location` *attribute information item*

The `location` *attribute information item*, if present, provides a hint to the processor as to where the DTD may be located. Caching and cataloging technologies may provide better information than this hint. The `location` *attribute information item* has the following infoset properties:

- A [local name] of location.
- A [namespace name] which has no value.

The type of the `location` *attribute information item* is *xs:anyURI*.

E.1.3 References to Element Definitions

The `element` *attribute information item* MUST be used when referring to an element definition (<!ELEMENT>) from a Message Reference component; referring to an element definition from a Interface Fault component is similar. The value of the element definition MUST correspond to the content of the `namespace` *attribute information item* of the `dtd:import` *element information item*. The local name part must correspond to an element defined in the DTD.

Note that this pattern does not attempt to make DTDs namespace-aware. It applies namespaces externally, in the import phase.

E.2 RELAX NG

A RELAX NG schema may be used as the schema language for WSDL. It may be embedded or imported; import is preferred. A namespace must be specified; if an imported schema specifies one, then the [actual value] of the `namespace` *attribute information item* in the `import` *element information item* must match the specified namespace. RELAX NG provides both type definitions and element declarations, the latter appears in the {element declarations} property of **2.1.1 The Definitions Component** respectively. The following discussion supplies the prefix rng which is mapped to the URI "http://www.relaxng.org/ns/structure/1.0".

E.2.1 Importing RELAX NG

Importing a RELAX NG schema uses the rng:include mechanism defined by RNG, with restrictions on its syntax and semantics. A child *element information item* of the `types` *element information item* is defined with the Infoset properties as follows:

- A [local name] of include.
- A [namespace name] of "http://www.relaxng.org/ns/structure/1.0".

- Two *attribute information item*s as follows:
  - A REQUIRED `ns` *attribute information item* as described below.
  - An OPTIONAL `href` *attribute information item* as described below.
  - Additional *attribute information item*s as defined by the RNG specification.

Note that WSDL restricts the `rng:include` *element information item* to be empty. That is, it cannot redefine `rng:start` and `rng:define` *element information item*s; it may be used solely to import a schema.

E.2.1.1 `ns` *attribute information item*

The `ns` *attribute information item* defines the namespace of the type and element definitions imported from the referenced schema. If the referenced schema contains an `ns` *attribute information item* on its `grammar` *element information item*, then the values of these two *attribute information item*s must be identical. If the imported grammar does not have an `ns` *attribute information item* then the namespace specified here is applied to all components of the schema as if it did contain such an *attribute information item*. The `ns` *attribute information item* contains the following Infoset properties:

- A [local name] of ns.
- A [namespace name] which has no value.

The type of the `ns` *attribute information item* is *xs:anyURI*.

E.2.1.2 `href` *attribute information item*

The `href` *attribute information item* must be present, according to the rules of the RNG specification. However, WSDL allows it to be empty, and considers it only a hint. Caching and cataloging technologies may provide better information that this hint. The `href` *attribute information item* has the following Infoset properties:

- A [local name] of href.
- A [namespace name] which has no value.

The type of the `href` *attribute information item* is *xs:anyURI*.

E.2.2 Embedding RELAX NG

Embedding an RNG schema uses the existing top-level `rng:grammar` *element information item*. It may be viewed as simply cutting and pasting an existing, stand-alone schema to a location inside the `wsdl:types` *element information item*. The `rng:grammar` *element information item* has the following Infoset properties:

- A [local name] of grammar.
- A [namespace name] of "http://www.relaxng.org/ns/structure/1.0".
- A REQUIRED `ns` *attribute information item*s as described below.
- Additional *attribute information item*s as specified for the `rng:grammar` *element information item* in the RNG specification.
- Child *element information item*s as specified for the `rng:grammar` *element information item* in the RNG specification.

E.2.2.1 `ns` *attribute information item*

The `ns` *attribute information item* defines the namespace of the type and element definitions embedded in this schema. WSDL modifies the RNG definition of the `rng:grammar` *element information item* to make this *attribute information item* required. The `ns` *attribute information item* has the following infoset properties:

- A [local name] of ns.
- A [namespace name] which has no value.

The type of the `ns` *attribute information item* is *xs:anyURI*.

E.2.3 References to Element Declarations

Whether embedded or imported, the element definitions present in a schema may be referenced from a Message Reference or Interface Fault component.

A named rng:define definition MUST NOT be referenced from the Message Reference or Interface Fault components.

A named Relax NG element declaration MAY be referenced from a Message Reference or Interface Fault component. The QName is constructed from the namespace (`ns` *attribute information item*) of the schema and the content of the `name` *attribute information item* of the `element` *element information item* An `element` *attribute information item* MUST NOT be used to refer to an `rng:define` *element information item*.

F. Part 1 Change Log (Non-Normative)

F.1 WSDL Specification Changes

Date	Author	Description
20040323	JJM	Commented out the (missing) property example.
20040322	RRC	Added definition of wsdli:wsdlLocation attribute.
20040322	JJM	Added faults to properties and features.
20040319	JJM	Use lowercase "should" in notes.
20040319	JJM	Comment out features at service level. Uniformize scope between features and properties.
20040318	JJM	Moved normative notes into the main body of the document.
20040318	JJM	Incorporated the property text from Glen.
20040318	JJM	Addressed comments from Yuxiao Zhao.
20040318	JJM	Updated the feature description, as per Glen and David Booth's suggestions.
20040317	RRC	Removed redundant {styleDefault} property of the interface component.
20040317	JJM	Include comments from Kevin.
20040315	RRC	Added clarification on embedded XML schemas that refer to siblings.
20040315	RRC	Updated RPC signature extension to use #in/#out/#inout/ #return tokens.
20040315	RRC	Added explanatory text to types and modularization sections per resolution of issue #102.
20040315	SW	Change binding/{fault,operation}/@name to @ref
20040312	RRC	Fixed appendix D to take the removal of wsdl:message into account.
20040312	RRC	Added definition of wrpc:signature extension attribute.
20040311	SW	Change fault stuff per decision to make faults first class in interfaces.
20040308	SW	Renamed {message} property to {element} and @message to @element
20040305	SW	Added {safety} property
20040227	MJG	Merged in branch Issue143 containing resolution of issue 143

20040227	SW	Dropped {type definitions} property from definitions; leftover from <message> days.
20040226	SW	Working thru various edtodo items.
20040106	JS	Per 18 Dec 2003 telecon decision, added text re: circular includes.
20031204	JS	Per 4 Dec 2003 telecon decision, removed redundant binding/operation/{infault, outfault}/@messageReference.
20031105	JS	Added point to attributes task force recommendation accepted by the working group.
20031104	JS	Mapping to component model for {message} of Fault Reference component indicated that `message` *attribute information item* was optional, but the pseudo syntax and XML reprsentation indicated it was required. Made uniformly optional to allow other type systems as was previously done for {message} of Message Reference component.
20031104	JS	Renamed interface /operation /{input,output} /@body to ./ @message and interface /operation /{infault,outfault} / @details to ./@message per 4 Nov face-to-face decision.
20031104	JS	Made interface /operation /{input,output,infault,outfault} / @messageReference optional per 4 Nov face-to-face decision.
20031104	JS	Removed interface/operation/{input,output}/@header per 4 Nov face-to-face decision.
20031102	SW	Updated fault reference components to indicate that if operation's MEP uses MTF then the fault is in the opposite direction as the referenced message and if it use FRM then its in the same direction. Per 10/30 telecon decision.
20031102	SW	Updated operation styles terminology per message #57 of Oct. and the RPC style rules per message #58 of Oct. per decision on 10/30 telecon to consider those status quo.
20031102	SW	Clarified wording in operation styles discussion to better explain the use of the {style} attribute.
20031102	SW	Clarified wording in XML <-> component model mapping section for message reference components to say that {body} and {headers} may not have a value.
20031102	SW	Made interface/operation/(input|output)/ @messageReference REQUIRED per 10/30 telecon decision.

20031028	SW	Renamed to wsdl20.xml and updated contents.
20031028	SW	Updated bindings.
20031025	SW	Updated faults.
20031013	JJM	Moved appendix C to a separate document, as per 24 Sep 2003 meeting in Palo Alto, CA.
20031003	SW	Softened <documentation> wording to allow machine processable documentation.
20031002	SW	Changed binding/operation/@name to QName per edtodo.
20030930	SW	Added placeholders for set-attr/get-attr operation styles.
20030929	SW	Inserted Glen Daniels' feature text.
20030919	RRC	Removed import facility for chameleon schemas and added a description of a workaround.
20030918	JJM	Changed message pattern to message exchange pattern, as per WG resolution on 18 Sep. 2003
20030916	RRC	Added editorial note for the missing RPC encoding style.
20030915	RRC	Yet more updates for REQUIRED, OPTIONAL; updated section 3 to reflect the removal of "wsdl:message".
20030911	RRC	More updates for REQUIRED, OPTIONAL; removed diff markup; fixed example C.4.
20030911	RRC	Renamed message reference "name" attribute and property to "messageReference"; fixed incorrect reference to "fault" element in the binding operation section.
20030910	SW	Fixed message references and added proper use of REQUIRED etc. for the part I've gone through so far.
20030910	SW	Updating spec; fixed up interface operation component more.
20030808	JCS	Fixed errors found by IBM\Arthur.
20030804	JCS	Removed Message component per 30 July-1 Aug meeting.
20030803	JCS	Replaced substitution groups with xs:any namespace='##other' per 3 July, 17 July, and 24 July telecons.
20030801	JCS	Made binding/@interface optional per 31 July meeting.
20030724	JCS	Remove @targetResource per 17 July 2003 telecon.
20030612	JJM	Incorporate revised targetResource definition, as per 12 June 2003 telcon.

20030606	JJM	Refer to the two graphics by ID. Indicate pseudo-schemas are not normative.
20030604	JJM	Fixed figures so they don't appear as tables. Fixed markup so it validates.
20030603	JCS	Plugged in jmarsh auto-generated schema outlines
20030529	MJG	Fixed various issues with the XmlRep portions of the spec
20030527	MJG	Added text to **2.2.1 The Interface Component** and **2.2.3 Mapping Interface's XML Representation to Component Properties** indicating that recursive interface extension is not allowed.
20030523	JJM	Added pseudo-syntax to all but Type and Modularizing sections.
20030523	JJM	Added the "interface" and "targetResource" attribute on <service>.
20030523	JJM	Fixed miscellaneous typos (semi-colon instead of colon, space after parenthesis, etc.).
20030523	JJM	Rewrote the service-resource text and merge it with the introduction.
20030522	JCS	s/set of parts/list of parts/.
20030514	JJM	Updated the service-resource figure, and split the diagram into two.
20030512	JJM	Added service-resource drawing and description.
20030512	JJM	Added syntax summary for the Interface component.
20030428	MJG	Various edits to **3. Types**, **E. Examples of Specifications of Extension Elements for Alternative Schema Language Support.** to accomadate other type systems and spell out how extensibility elements/ attributes play out in such scenarios.
20030428	MJG	Added text to **1.2 Notational Conventions** regarding normative nature of schema and validity of WSDL documents
20030411	JJM	Allowed features and properties at the interface, interface operation, binding and binding operation levels, as agreed at the Boston f2f http://lists.w3.org/Archives/ Public/www-ws-desc/2003Mar/0019.html.
20030411	JJM	Incorporate features and properties' text from separate document and merged change logs
20030313	MJG	Changed title to include 'part 1'

20030313	MJG	Changed port to endpoint
20030313	MJG	Changed type to interface in binding
20030313	MJG	Changed mep to pattern and message exchange pattern to message pattern
20030313	MJG	Added text to **D.2 PortTypes**
20030313	MJG	Changed portType to interface
20030407	JJM	Refined and corrected the definitions for features and properties.
20030304	JJM	Filled in blank description of Feature and Property component.
20030303	MJG	Skeleton Feature and Property components
20030305	MJG	Merged ComponentModelForMEPs branch (1.46.2.5) into main branch (1.54). Below is change log from the branch: **Date** / **Author** / **Description** 20030220 / MJG / Minor wording change at suggestion of JJM 20030212 / MJG / Updated component model to include Fault Reference component. Associated changes to Port Type Operation component 20030211 / MJG / Changes to component model to support MEPs
20030228	MJG	Updated **4.2 Importing Descriptions** to be consistent in layout with other XML rep sections. Detailed that documentation and extensibility attributes are allowed, per schema
20030228	MJG	Updated **4.1 Including Descriptions** to be consistent in layout with other XML rep sections. Detailed that documentation and extensibility attributes are allowed, per schema
20030228	MJG	Updated **2.9.2 XML Representation of Binding Component** to list type attribute
20030217	MJG	Minor edits to wording in **2.4.1 The Interface Operation Component**
20030213	MJG	Added xlink nsdecl to spec element

<table>
<tr><td>20030213</td><td>MJG</td><td>Incorporated text from dbooths proposal on semantics, per decision 20021031</td></tr>
<tr><td>20030213</td><td>MJG</td><td>Merged operationnames branch (1.37.2.3) into main branch (1.46). Below is the change log from the branch.
<table>
<tr><th>Date</th><th>Author</th><th>Description</th></tr>
<tr><td>20030130</td><td>MJG</td><td>Updated binding section to match changes to port type section WRT operation names</td></tr>
<tr><td>20030130</td><td>MJG</td><td>Added best practice note on operation names and target namespaces to 2.4.1 The Interface Operation Component</td></tr>
<tr><td>20030122</td><td>MJG</td><td>Started work on making operations have unique names</td></tr>
</table></td></tr>
<tr><td>20030213</td><td>MJG</td><td>Change name of {message exchange pattern} back to {variety} to consolidate changes due to MEP proposal</td></tr>
<tr><td>20030206</td><td>MJG</td><td>Updated Appendix A to refer to Appendix C</td></tr>
<tr><td>20030204</td><td>MJG</td><td>Tidied up appendix C</td></tr>
<tr><td>20030203</td><td>MJG</td><td>Incorporated resolution to R120</td></tr>
<tr><td>20030124</td><td>MJG</td><td>Fixed error in 2.5.2 XML Representation of Message Reference Component which had name attribute information item on input, output and fault element information item being mandatory. Made it optional.</td></tr>
<tr><td>20030123</td><td>JJM</td><td>Change name of {variety} property to {message exchange pattern}</td></tr>
<tr><td>20030130</td><td>MJG</td><td>Updated binding section to match changes to port type section WRT operation names</td></tr>
<tr><td>20030130</td><td>MJG</td><td>Added best practice note on operation names and target namespaces to 2.4.1 The Interface Operation Component</td></tr>
<tr><td>20030122</td><td>MJG</td><td>Started work on making operations have unique names</td></tr>
<tr><td>20030122</td><td>MJG</td><td>Added some <emph>, <el>, <att>, &AII;, &EII;, <el> markup</td></tr>
<tr><td>20030120</td><td>MJG</td><td>Incorporated Relax NG section from Amy's types proposal</td></tr>
<tr><td>20030120</td><td>MJG</td><td>Incorporated DTD section from Amy's types proposal</td></tr>
<tr><td>2003020</td><td>MJG</td><td>Incorporated Amy's types proposal except annexes</td></tr>
</table>

20030118	MJG	Made some changes related to extensibility
20030118	MJG	Amended content model for operation to disallow fault element children in the input-only and output-only cases
20030118	MJG	Removed {extension} properties from Binding components and Port components. Added text relating to how extension elements are expected to annotate the component model.
20030117	MJG	Made further edits related to extensibility model now using substitution groups
20030117	MJG	Added initial draft of section on QName resolution
20030117	MJG	Reworked section on extensibility
20030116	MJG	Added text regarding multiple operations with the same {name} in a single port type
20030116	MJG	Added section on symbol spaces
20030116	MJG	Removed various ednotes
20030116	MJG	Added section on component equivalence
20030116	MJG	More work on include and import
20021201	MJG	Did some work on wsdl:include
20021127	MJG	Added placeholder for wsdl:include
20021127	MJG	Cleaned up language concerning `targetNamespace` *attribute information item* **2.1.2.1 targetNamespace attribute information item**
20021127	MJG	changed the language regarding extensibility elements in **2.1.2 XML Representation of Definitions Component**.
20021127	MJG	Moved all issues into issues document (../issues/wsd-issues.xml)
20021127	MJG	Removed name attribute from definitions element
20021127	MJG	Removed 'pseudo-schema'
20021121	JJM	Updated media type draft appendix ednote to match minutes.
20021111	SW	Added appendix to record migration issues.
20021107	JJM	Incorporated and started adapting SOAP's media type draft appendix.
20021010	MJG	Added port type extensions, removed service type.
20020910	MJG	Removed parameterOrder from spec, as decided at September 2002 FTF

20020908	MJG	Updated parameterOrder description, fixed some spelling errors and other types. Added ednote to discussion of message parts
20020715	MJG	AM Rewrite
20020627	JJM	Changed a few remaining <emph> to either <att> or <el>, depending on context.
20020627	SW	Converted portType stuff to be infoset based and improved doc structure more.
20020627	SW	Converted message stuff to be infoset based and improved doc structure more.
20020625	SW	Mods to take into account JJM comments.
20020624	JJM	Fixed spec so markup validates.
20020624	JJM	Upgraded the stylesheet and DTD
20020624	JJM	Added sections for references and change log.
20020624	JJM	Removed Jeffrey from authors :-(Added Gudge :-)
20020620	SW	Started adding abstract model
20020406	SW	Created document from WSDL 1.1

TECHNICAL PROCESS

Overview
TC Process
IPR Policy
TC Guidelines
Spec Templates
Approved Work

TECHNICAL COMMITTEES

Current TC List
Join a TC
Propose a New TC
Mailing List Archive

Access Control
Application Vulnerability
Asynch Service
Biometrics
Business-Centric Methodology
Business Transactions
CGM Open WebCGM
Content Assembly
Customer Information
Digital Signature Svcs
DITA
DocBook
eb Serv Oriented Arch
ebXML Business Process
ebXML CPPA
ebXML Implement
ebXML Messaging
ebXML Registry
e-Government
Election Services
Electronic Procurement
Emergency Mgmt
Entity Resolution
Framework for WS Impl
HumanMarkup
LegalXML Court Filing
LegalXML eContracts
LegalXML eNotary
LegalXML IntJustice
LegalXML Legislative
LegalXML ODR
Localization
Materials
Open Building Info Exchange
Open Office Format
PKI
Product Life Cycle
Prod Plan Sched
Provisioning

OASIS UDDI Specification TC

Join This TC | TC Members Page | Send A Comment

The purpose of the UDDI Specification TC is to evolve work on the Web services registry foundations. The UDDI specifications form the necessary technical foundation for publication and discovery of Web services implementations both within and between enterprises. The UDDI Specification TC is a Technical Committee of the UDDI.org Member Section.

Chairs: Mr Tom Bellwood bellwood@us.ibm.com, Mr Tony Rogers Tony.Rogers@ca.com, Mr Luc Clement luc@iclement.net.

Charter
IPR Statement
FAQ
Membership
List Archives
Comments Archive
Documents
Schedule
Minutes
Press

Announcements

The UDDI v2 specifications have been promoted as OASIS standards. The UDDI v3, now at revision v3.0.1, and Schema Centric Canonicalization specifications are UDDI Spec TC Committee Specifications.

UDDI v3.0.1 Committee Specification. The UDDI Spec TC has approved the UDDI v3.0.1 specification as a UDDI Spec TC Committee Specification. A marked-up version of the specification is available

Technical Notes

- ***New*** Value Set Overview Documents
- ***Updated*** UDDI as the registry for ebXML Components
- Providing a Value Set For Use in UDDI Version 3
- Versioning Value Sets in a UDDI Registry, Version 1.12
- Using WSDL in a UDDI Registry, Version 2.0

Meeting Schedule

The UDDI Specification TC meets every third-Tuesday via conference call and holds quarterly face-to-face meetings on a schedule determined by the UDDI Specification TC members. See conference call dates and details.

Conference call times are as follows by region: **UTC**: Tue-19:30, **Seattle**: Tue-12:30, **New York**: Tue-15:30, **London**: Tue-20:30, **Frankfurt**: Tue-21:30, **Moscow**: Tue-23:30, **Tokyo**: Wed-04:30,

TC PARTICIPANTS:

Computer Associates

IBM

Microsoft Corporation

Oracle

SAP

Organizations listed above are OASIS Sponsor-level members who have representatives serving on this TC.

RELAX NG
Rights Language
Security Services
Tax XML
TM Pub Sbj Geo Lang
TM Vocab for XML Stds
Topic Maps Published Subjects
Translation WS
UDDI Spec
Universal Business Language
User Interface
Web Application Security
Web Services Business Process
Web Services Comp Appl Framework
Web Services Distrib Mgmt
Web Services Notification
Web Services Reliable Messaging
Web Services Resource Framework
Web Services for Remote Portlets
Web Services Security
XDI
XRI
XSLT Conformance

JOINT COMMITTEES
ebXML Joint Committee
Security Joint Committee

OASIS NETWORK
CGM Open
ebXML
LegalXML
PKI
UDDI

OASIS INFO CHANNELS
Cover Pages
XML.org
Sponsorship

NEWSLETTERS
OASIS News
Using our RSS Feed
Cover Pages Weekly
XML.org Daily Newslink
Subscribe

Sydney: Wed-05:30, **Auckland**: Wed-07:30

Subscribe to uddi-dev, an unmoderated, public mail list that provides an open forum for developers to exchange ideas and information on implementing the UDDI OASIS Standard.

To send a comment to this TC, click the "Send A Comment" button above.

Quick Links

Committee Specifications:
- UDDI Version 2 Specifications
- UDDI Version 3 Specification
- Schema Centric XML Canonicalization Specification

Specification Change Requests
UDDI Spec TC Committee Best Practices
UDDI Spec TC Committee Technical Notes
UDDI Spec TC Committee Drafts
Contributed Material:
- UDDI Version 1 Specifications
- UDDI WG UDDI v2 Specification Errata

UDDI Mailing Lists
UDDI.org Member Section

Process

The TC as adopted processes to manage its deliverables, namely Committee Specification, Best Practices, Technical Notes, and Change Requests. The TC has posted its "UDDI Spec TC Process" document.

This OASIS TC web page is maintained by webmaster@oasis-open.org

TOP

WebServices - Axis

WebServices - Axis - Introduction

NEWS (May 07, 2004) : Axis C++1.1.1 is now available!

NEWS (April 16, 2004) : Axis C++1.1 is now available!

NEWS (March 31, 2004) : Axis 1.2 Beta is now available.

NEWS (December 1, 2003) : Axis 1.2 Alpha is now available.

(June 16, 2003) : Axis 1.1 Final is still the most recent stable release (read the release notes)!

Apache Axis is an implementation of the SOAP ("Simple Object Access Protocol") submission to W3C.

From the draft W3C specification:

> SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

This project is a follow-on to the Apache SOAP project.

Please see the Reference Library for a list of technical resources that should prove useful.

Axis 1.2 and beyond

Axis 1.1 has proven itself to be a reliable and stable base on which to implement Java Web Services. There is a very active user community and there a many companies who use Axis for Web Service support in their products.

For Axis 1.2, we are focusing on our document/ literal support to better address the WS-I Basic Profile 1.0 and JAX-RPC 1.1 specifications. And we are fixing as many bug as possible.

We can always use **your** help. Here are some links to help you help us:

- How do I report bugs?
- How do I submit patches to Axis?
- Where can i get snapshots of latest CVS?

Credits

The Axis Development Team

PDF

BEA Announces Open Source Application Framework Based on WebLogic Workshop

(20.05.2004 18:32)

"WebLogic Workshop consists of two major technologies: a powerful integrated development environment and an application framework to abstract many of the tedious tasks associated with Java development," said Scott Dietzen, CTO, BEA Systems. "By open sourcing the application framework, we can help provide a way for all Java developers, as well as our ISV partners, to build fully portable applications more productively, which creates immense business opportunities and future growth for the Java ecosystem. Time will prove these same technologies critical to the standardization of inter-application orchestration via work-flows and Web-flows."

Read more | Open Source

Sun's Quarterly Web Event, 04-Q2: Tune in to Network Computing

(20.05.2004 18:13)

Taking place on June 1st, three Web events cover the latest news and product information and includes videocasts with Sun executives and technology experts.

Read more | Web Seminars

Why SOA Now: A Q&A With BEA CIO Rhonda Hocker

(20.05.2004 11:12)

According to industry analysts, the basic concepts of service-oriented architecture (SOA) - reuse and interoperability -- have been around for 20 years. So what's new about SOA? Why is it likely to succeed where other technologies and standards have failed? BEA CIO Rhonda Hocker answers these questions about SOA's potential for transforming IT.

Read more | Industry News

IDC Reports on Competitive Market for Web Services Based Professional Services

(20.05.2004 10:48)

According to a new study from IDC in which they analyse the Web services-based professional services market, they report that "services firms report significant increases in the number of projects, in project complexity, and in the penetration of Web services in their every day services activities".

Read more | Analysts/Opinion

- Business Process Management
- Enterprise Web Services

Poll

Would you use open-source Web services solutions?

- Yes
- Possibly
- No

Vote

- Results
- Polls

Members

- Interested in doing a product review?
- Report from a conference
- Member's opinion articles
- Suggest a Poll
- Give us feedback

Members

- Interested in doing a product review?
- Report from a conference
- Member's opinion articles
- Suggest a Poll
- Give us feedback

PolarLake Announces Product Roadmap for PolarLake Integration Suite v4.0: Adds BPEL and BAM Support

(20.05.2004 10:35)

Ronan Bradley, CEO, PolarLake. "Specifically, this release addresses key requirements for organizations solving the most complex integration problems with XML and Web Services - the need to orchestrate business processes within and across departments and to track and monitor the flow of process and data through the network". Support will be included for business activity management (BAM) and business process orchestration based on the BPEL4WS (Business Process Execution Language for Web Services) standard.

Read more | Industry News

Analysis: Can You Spare a Cycle or Two? Grid Computing Moves Closer to Web Services

(15.03.2004 20:28)

Joe McKendrick provides an analysis and overview of recent movements in Grid computing towards more harmonization with Web services.

Read more | Analysts/Opinion

A Design Center for Web Services

(15.08.2002 12:07)

A review of CapeClear's flagship products, CapeStudio, a development environment for Web services and CapeConnect, a hosting environment designed for exposing enterprise Web services.

Read review | Vendor Reports

WS-I Publishes Basic Security Profile Working Group Draft

(20.05.2004 10:02)

WS-I announces the availability of the WS-I Basic Security Profile Working Group Draft for public comment. The Basic Security Profile follows the earlier publication of WS-I Security Scenarios Working Group Draft which defined the requirements and scope for the WS-I Basic Security Profile.

Read more | WS-I

Web services = or != distributed objects?

(02.03.2004 06:57)

David Orchard of BEA Systems writes about how Web services and distributed object technology differ, and what the critical success factors are.

Read more | Web Services Papers

A New Architecture Standard for Java Business Integration

(10.08.2003 17:30)

Sun recently announced Java Business Integration (JBI), a vision for a new Java based architecture for business integration products and solutions. Sun have now published a new white paper on JBI, describing its key concepts, scope and architecture.

Read more | Java

All under one roof with BEA's WebLogic Enterprise Platform

(11.01.2003 09:05)

An overview of WebLogic Enterprise Platform which comprises BEA WebLogic Portal, BEA WebLogic Integration, BEA WebLogic Workshop, and WebLogic Server.

Read more | Vendor Reports

Four Ways to Know Your WSDL

(19.09.2003 10:04)

SOAP has received considerable attention as a flagship standard for interoperable services; however, as Web services move to maturity, it is becoming clear that in fact WSDL is more central to Web service design and operation. This shift of focus requires architects and developers to thoroughly understand WSDL and its implications when they deploy services. This paper presents four specific techniques to scrutinize and master the WSDL language, taking advantage of Mindreef's Web services diagnostic system, SOAPscope 2.0.

Read more | WSDL

Actional Introduces Free Developer Version of SOAPstation Web Service Broker

(20.05.2004 10:02)

Actional SOAPstation is a Web Services Broker, "a proxy - brokering the interactions between applications providing Web services and the systems that build on them". The free developer version of SOAPstation is for development purposes only. Deploying an application requires a runtime license.

Read more | Web Services Tools

Roman Stanek of Systinet Discusses Web Services, SOA and Standards

(18.05.2004 17:21)

Roman briefly discusses how he sees Web services being used, and his views on future technology for SOA. " SOA is an architectural style of computing and Web services are an implementation of SOA. SOA is not a new term but Web services make this "dream" achievable. "

Read more | Industry News

Head lines

W3C Publish First Working Draft of Web Services Choreography

(27.04.2004)

Web Services Security (WSS) Officially Ratified as OASIS Standard

(20.04.2004)

Systinet Support WS-ReliableMessaging in New Beta Version of WASP Server for Java 5.0

(15.04.2004)

Gartner Advise Enterprises to Adopt Web Services Security (WS-Security)

(14.04.2004)

OASIS To Update ebXML Technical Architecture in Line with Web Services and SOA

(09.04.2004)

Systinet Announces New Family of Products to Enable Service-Oriented Architectures

(18.05.2004 14:52)

Systinet announce the immediate availability of Systinet Gateway 1.0, Systinet UDDI Registry 5.0 and Systinet Server for Java 5.0 (formerly WASP Server for Java) and C++ 5.0. Systinet Gateway 1.0 "extends and bridges proprietary message-oriented middleware (MOM) products, like TIBCO Rendezous, IBM WebSphereMQ (formerly MQ Series), SonicMQ and any other JMS-based solutions". The new 5.0 release of Systinet Server for Java fully supports reliable messaging using the WS-ReliableMessaging standard.

Read more | Industry News

Grand Central Promotes Loosely-Coupled, But Level Playing Field

(10.08.2003 19:24)

Web services may be the glue that will bring organizations together, but there are still many sticking points that keep them apart. Joe McKendrick reports on Grand Central Communications, a company committed to solving integration and communication issues between organizations by acting as a Web services interoperability network intermediary.

Read article | Vendor Reports

Integrating C++ Web Services with Rogue Wave's Lightweight Enterprise Integration Framework

(17.11.2003 21:32)

Neil Roodyn discusses his experiences of using the Rogue Wave Lightweight Enterprise Integration Framework (LEIF) to expose existing C++ components as Web services and also to consume Web services into C++ applications.

Read more | C++

The JavaOne Conference, June 28 - July 1st, San Francisco, CA

(15.05.2004 08:47)

JavaOne conference Attendees will receive an added benefit to their conference package: free membership to the JavaOne Online program, provides in-depth Java technology training delivered through content-rich sessions, Webinars, and featured articles. Attendees will have access to over 500 multimedia sessions from past Conferences.

Read more | Events

Optimyz WebServiceTester 3.0 EA Adds Support for WS-Security and BPEL4WS

(18.02.2004 21:47)

WebServiceTester 3.0 EA (Early Access) provides support for WS-Orchestration based on BPEL4WS specification. The new version of the tool "will graphically represent all the dependencies and drive the entire Workflow execution and notify the users about the defects". Also included is support for HTTP Authentication, SOAP Authentication, Binary Security Tokens, XML Signatures and XML Encryption.

Read more | Testing

Jonathan Schwartz of Sun Talks about Interoperability between Visual Studio and Java Studio Creator

(11.05.2004 18:20)

In a brief interview Jonathan Schwartz, the recently appointed COO and President of Sun, says "Cooperation with .NET simply means answering the needs of our customers. For example, we are looking at building interoperability between Visual Studio and Java Studio Creator, and having a common philosophy on how you build service-oriented applications delivered through both of those technologies".

Read more | Industry News

Gartner Think Cape Clear Has Chance to Lead ESB Integration Market

(10.05.2004 13:23)

A recently published profile of Cape Clear Software by Gartner Group has declared that Cape Clear can become a leader in the emerging market for Enterprise Service Bus (ESB) integration but needs to sustain differentiation of its products in the face of challenges from large, established middleware vendors.

Read more | Analysts/Opinion

Datapower Break Gigabit Barrier For XML Processing with its XG4 XML Chip

(10.05.2004 11:19)

"The patent-pending XG4 architecture combines DataPower's XML ASIC, off-the-shelf components and proprietary driver software to perform XPath and XML Schema validation directly in hardware"

Read more | Industry News

PayPal Announces PayPal Web Services

(05.05.2004 10:22)

PayPal Web Services, currently in beta release, is comprised of four new informational and transactional APIs: TransactionSearch, GetTransactionDetails, RefundTransaction and MassPay. Access is available through a newly launched PayPal Developer Central website. Code examples are given in .Net and Java Axis. A test enviornment (sandbox) is also provided.

BPEL and BPELJ: The Role of Standards in Building Process-Driven Applications, Webinar

(05.05.2004 10:21)

In this Webinar held 28th April, Stephen Hood, Product Manager for WebLogic Integration at BEA, outlines the functionality of outline the functionality of both BPEL and BPELJ and explores the advantages that process standards provide. Windows Media version of the presentation is available.

Read more | Events

Read more | Development

IONA Presents Half Day Seminar: Put Your Company on the SOA Fast Track, Boston, June 9, 2004

(05.05.2004 09:31)

IONA, along with Forrester Research, and Microsoft are holding a free seminar on the return on investment and the potential cost savings that enterprise customers can expect by adopting Web services based.

Read more | Events

Microsoft Publish Devices Profile for Web Services

(05.05.2004 09:14)

The Devices Profile for Web Services, co-authored by Intel, Lexmark and Ricoh, describes a core subset of Web services specifications that devices can implement to enable a base level of interoperability. Working within the UPnP 2.0 architecture, "network-connected devices can automatically be discovered, installed and utilized using the Windows Plug and Play subsystem".

Read more | .Net

Designing J2EE 1.4 Web Applications: An Excerpt from Designing Web Services With J2EE 1.4

(04.05.2004 15:33)

Larry Freeman, Yutaka Yoshida, and Beth Stearns of Sun provide an overview of the J2EE 1.4 "The Adventure Builder Reference" application, a demonstration application analogous to Pet Store. This article is taken from the forthcoming Java BluePrints book, "Designing Web Services with the J2EE 1.4 Platform".

Read more | Development

Using Mindreef's SOAPscope with Systinet's WASP

(04.05.2004 15:20)

Jiri Fabian of Systinet illustrates how Mindreef's SOAPscope can radically speedup the development cycle by providing handy tools to trace and debug Web service invocations.

Read more | Development

Sun Release J2EE 1.4 Java Application Verification Kit (30.04.2004 10:05)

W3C Publish First Working Draft of Web Services Choreography (27.04.2004 17:47)

Web Services and Grid Computing: Convergence to Utility Computing, 24-25th May, London	(27.04.2004 16:43)
Systinet And Arjuna Partner To Deliver Transaction Support For Web Services	(27.04.2004 15:35)
Sun and Microsoft Forge Broad Cooperative Agreement	(27.04.2004 15:23)
IBM Announces WebSphere Business Integration Server Foundation with Support for BPEL	(22.04.2004 09:39)

NTT DATA, CSN and Sun Purchase Optimyz's WebServiceTester Product

(22.04.2004 08:07)

Optimyz's WebServiceTester, currently in version 2.0, provides support for "Functional, Regression, Load/Stress, Performance, and Interoperability testing of Web Services". It was chosen by NTT since it offers "Testing without Programming and Code Maintenance", by Sun for automatically generating tests and performing load/stress testing for the SunONE Application Server.

press release | Industry News

Apache Axis C++ 1.1 Released

(20.04.2004 17:38)

Axis C++ is a SOAP implementation done in C++ which can be used as a SOAP messaging server or client. This release includes enhancements over the 1.0 release such as document/literal support.

Read more | C++

Web Services Security (WSS) Officially Ratified as OASIS Standard

(20.04.2004 17:05)

OASIS have announced that its members have approved the Web Services Security (WSS) version 1.0 (WS-Security 2004) as an OASIS Standard. Participant members give closing remarks on specification.

Read more | Security / Identity

Principles of SOA Design - Webinar, April 27th

(19.04.2004 17:22)

Hosted by Cape Clear Software, this free webinar discusses how architects and developers can leverage Web Services design principles to support a Service Oriented Architecture (SOA).

Read more | Web Seminars

Reactivity and Grand Central Partner to Enhance Secure Enterprise Integration	(19.04.2004 17:11)
The Economics of Integration - Webinar, 29th April	(17.04.2004 17:32)
Systinet Support WS-ReliableMessaging in New Beta Version of WASP Server for Java 5.0	(15.04.2004 18:20)
Gartner Advise Enterprises to Adopt Web Services Security (WS-Security)	(14.04.2004 16:37)
Early Access Release of Sun's Java Studio Creator Now Available	(14.04.2004 16:25)
Oracle Release Updated Version of JDeveloper with WS-I Basic Profile and J2EE 1.4 Support	(14.04.2004 16:11)
Web Services Security Startup Ping Identity Completes First Round of Financing	(14.04.2004 15:03)
XML Europe 2004 - Amsterdam, 18-21st April	(13.04.2004 16:14)
Systinet Web Services Software Chosen for BMC Software's New Developer's Kit	(13.04.2004 16:04)
BEA eWorld 2004: Deploy SOA Now, May 24-27th, San Francisco	(12.04.2004 09:31)

OASIS To Update ebXML Technical Architecture in Line with Web Services and SOA (09.04.2004 08:52)

<%@page import="java.util.Date"%>

Hello World!

Current date: <%= (new Date()).toString() %>

FAUST: EINE TRAGÖDIE

Zueignung.

<%@ include file="Zueignung.html" %>

Vorspiel auf dem Theater.

<%@ include file="Vorspiel.html" %>

Prolog im Himmel.

<%@ include file="Prolog.html" %>

```
import java.io.IOException;
import java.util.Date;

import javax.servlet.jsp.JspTagException;
import javax.servlet.jsp.tagext.TagSupport;

public class HelloTag extends TagSupport {
	public int doStartTag() throws JspTagException {
		return EVAL_BODY_INCLUDE;
	}

	public int doEndTag() throws JspTagException {
		try {
			pageContext.getOut().write("Hello World!");
			pageContext.getOut().write(
				"current date: " + new Date().toString());
		} catch (IOException ioe) {
			ioe.printStackTrace();
		}
		return EVAL_PAGE;
	}

}
```

Hello World!

<%@ taglib uri="hello" prefix="example" %>

Statische Ausgabe ...

XSL Transformations (XSLT) Version 1.0

W3C Recommendation 16 November 1999

This version:

http://www.w3.org/TR/1999/REC-xslt-19991116

(available in XML or HTML)

Latest version:

http://www.w3.org/TR/xslt

Previous versions:

http://www.w3.org/TR/1999/PR-xslt-19991008
http://www.w3.org/1999/08/WD-xslt-19990813
http://www.w3.org/1999/07/WD-xslt-19990709
http://www.w3.org/TR/1999/WD-xslt-19990421
http://www.w3.org/TR/1998/WD-xsl-19981216
http://www.w3.org/TR/1998/WD-xsl-19980818

Editor:

James Clark <jjc@jclark.com>

Abstract

This specification defines the syntax and semantics of XSLT, which is a language for transforming XML documents into other XML documents.

XSLT is designed for use as part of XSL, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that

uses the formatting vocabulary.

XSLT is also designed to be used independently of XSL. However, XSLT is not intended as a completely general-purpose XML transformation language. Rather it is designed primarily for the kinds of transformations that are needed when XSLT is used as part of XSL.

Status of this document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from other documents. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

The list of known errors in this specification is available at http://www.w3.org/1999/11/REC-xslt-19991116-errata.

Comments on this specification may be sent to xsl-editors@w3.org; archives of the comments are available. Public discussion of XSL, including XSL Transformations, takes place on the XSL-List mailing list.

The English version of this specification is the only normative version. However, for translations of this document, see http://www.w3.org/Style/XSL/translations.html.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

This specification has been produced as part of the W3C Style activity.

Table of contents

Appendices

---

1 Introduction

This specification defines the syntax and semantics of the XSLT language. A transformation in the XSLT language is expressed as a well-formed XML document [XML] conforming to the Namespaces in XML Recommendation [XML Names], which may include both elements that are defined by XSLT and elements that are not defined by XSLT. XSLT-defined elements are distinguished by belonging to a specific XML namespace (see [**2.1 XSLT Namespace**]), which is referred to in this specification as the **XSLT namespace**. Thus this specification is a definition of the syntax and semantics of the XSLT namespace.

A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added.

A transformation expressed in XSLT is called a stylesheet. This is because, in the case when XSLT is transforming into the XSL formatting vocabulary, the transformation functions as a stylesheet.

This document does not specify how an XSLT stylesheet is associated with an XML document. It is recommended that XSL processors support the mechanism described in [XML Stylesheet]. When this or any other mechanism yields a sequence of more than one XSLT stylesheet to be applied simultaneously to a XML document, then the effect should be the same as applying a single stylesheet that imports each member of the sequence in order (see [**2.6.2 Stylesheet Import**]).

A stylesheet contains a set of template rules. A template rule has two parts: a pattern which is matched against nodes in the source tree and a template which can be instantiated to form part of the result tree. This allows a stylesheet to be applicable to a wide class of documents that have similar source tree structures.

A template is instantiated for a particular source element to create part of the result tree. A template can contain elements that specify literal result element structure. A template can also contain elements from the XSLT namespace that are instructions for creating result tree fragments. When a template is instantiated, each instruction is executed and replaced by the result tree fragment that it creates. Instructions can select and process descendant source elements. Processing a descendant element creates a result tree fragment by finding the applicable template rule and instantiating its template.

Note that elements are only processed when they have been selected by the execution of an instruction. The result tree is constructed by finding the template rule for the root node and instantiating its template.

In the process of finding the applicable template rule, more than one template rule may have a pattern that matches a given element. However, only one template rule will be applied. The method for deciding which template rule to apply is described in [**5.5 Conflict Resolution for Template Rules**].

A single template by itself has considerable power: it can create structures of arbitrary complexity; it can pull string values out of arbitrary locations in the source tree; it can generate structures that are repeated according to the occurrence of elements in the source tree. For simple transformations where the structure of the result tree is independent of the structure of the source tree, a stylesheet can often consist of only a single template, which functions as a template for the complete result tree. Transformations on XML documents that represent data are often of this kind (see [**D.2 Data Example**]). XSLT allows a simplified syntax for such stylesheets (see [**2.3 Literal Result Element as Stylesheet**]).

When a template is instantiated, it is always instantiated with respect to a **current node** and a **current node list**. The current node is always a member of the current node list. Many operations in XSLT are relative to the current node. Only a few instructions change the current node list or the current node (see [**5 Template Rules**] and [**8 Repetition**]); during the instantiation of one of these instructions, the current node list changes to a new list of nodes and each member of this new list becomes the current node in turn; after the instantiation of the instruction is complete, the current node and current node list revert to what they were before the instruction was instantiated.

XSLT makes use of the expression language defined by [XPath] for selecting elements for processing, for conditional processing and for generating text.

XSLT provides two "hooks" for extending the language, one hook for extending the set of instruction elements used in templates and one hook for extending the set of functions used in XPath expressions. These hooks are both based on XML namespaces. This version of XSLT does not define a mechanism for implementing the hooks. See [**14 Extensions**].

> **NOTE:**The XSL WG intends to define such a mechanism in a future version of this specification or in a separate specification.

The element syntax summary notation used to describe the syntax of XSLT-defined elements is described in [**18 Notation**].

The MIME media types `text/xml` and `application/xml` [RFC2376] should be used for XSLT stylesheets. It is possible that a media type will be registered specifically for XSLT stylesheets; if and when it is, that media type may also be used.

2 Stylesheet Structure

2.1 XSLT Namespace

The XSLT namespace has the URI `http://www.w3.org/1999/XSL/Transform`.

> **NOTE:**The `1999` in the URI indicates the year in which the URI was allocated by the W3C. It does not indicate the version of XSLT being used, which is specified by attributes (see [**2.2 Stylesheet Element**] and [**2.3 Literal Result Element as Stylesheet**]).

XSLT processors must use the XML namespaces mechanism [XML Names] to recognize elements and attributes from this namespace. Elements from the XSLT namespace are recognized only in the stylesheet not in the source document. The complete list of XSLT-defined elements is specified in [**B Element Syntax Summary**]. Vendors must not extend the XSLT namespace with additional elements or attributes. Instead, any extension must be in a separate namespace. Any namespace that is used for additional instruction elements must be identified by means of the extension element mechanism specified in [**14.1 Extension Elements**].

This specification uses a prefix of `xsl:` for referring to elements in the XSLT namespace. However, XSLT stylesheets are free to use any prefix, provided that there is a namespace declaration that binds the prefix to the URI of the XSLT namespace.

An element from the XSLT namespace may have any attribute not from the XSLT namespace, provided that the expanded-name of the attribute has a non-null namespace URI. The presence of such attributes must not change the behavior of XSLT elements and functions defined in this document. Thus, an XSLT processor is always free to ignore such attributes, and must ignore such attributes without giving an error if it does not recognize the namespace URI. Such attributes can provide, for example, unique identifiers, optimization hints, or documentation.

It is an error for an element from the XSLT namespace to have attributes with

expanded-names that have null namespace URIs (i.e. attributes with unprefixed names) other than attributes defined for the element in this document.

> **NOTE:**The conventions used for the names of XSLT elements, attributes and functions are that names are all lower-case, use hyphens to separate words, and use abbreviations only if they already appear in the syntax of a related language such as XML or HTML.

2.2 Stylesheet Element

```
<xsl:stylesheet
  id = id
  extension-element-prefixes = tokens
  exclude-result-prefixes = tokens
  version = number>
  <!-- Content: (xsl:import*, top-level-elements) -->
</xsl:stylesheet>
```

```
<xsl:transform
  id = id
  extension-element-prefixes = tokens
  exclude-result-prefixes = tokens
  version = number>
  <!-- Content: (xsl:import*, top-level-elements) -->
</xsl:transform>
```

A stylesheet is represented by an `xsl:stylesheet` element in an XML document. `xsl:transform` is allowed as a synonym for `xsl:stylesheet`.

An `xsl:stylesheet` element must have a `version` attribute, indicating the version of XSLT that the stylesheet requires. For this version of XSLT, the value should be `1.0`. When the value is not equal to `1.0`, forwards-compatible processing mode is enabled (see [**2.5 Forwards-Compatible Processing**]).

The `xsl:stylesheet` element may contain the following types of elements:

- `xsl:import`
- `xsl:include`
- `xsl:strip-space`

- xsl:preserve-space
- xsl:output
- xsl:key
- xsl:decimal-format
- xsl:namespace-alias
- xsl:attribute-set
- xsl:variable
- xsl:param
- xsl:template

An element occurring as a child of an xsl:stylesheet element is called a **top-level** element.

This example shows the structure of a stylesheet. Ellipses (...) indicate where attribute values or content have been omitted. Although this example shows one of each type of allowed element, stylesheets may contain zero or more of each of these elements.

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">
  <xsl:import href="..."/>

  <xsl:include href="..."/>

  <xsl:strip-space elements="..."/>

  <xsl:preserve-space elements="..."/>

  <xsl:output method="..."/>

  <xsl:key name="..." match="..." use="..."/>

  <xsl:decimal-format name="..."/>

  <xsl:namespace-alias stylesheet-prefix="..."
```

```
result-prefix="..."/>

  <xsl:attribute-set name="...">
    ...
  </xsl:attribute-set>

  <xsl:variable name="...">...</xsl:variable>

  <xsl:param name="...">...</xsl:param>

  <xsl:template match="...">
    ...
  </xsl:template>

  <xsl:template name="...">
    ...
  </xsl:template>

</xsl:stylesheet>
```

The order in which the children of the `xsl:stylesheet` element occur is not significant except for `xsl:import` elements and for error recovery. Users are free to order the elements as they prefer, and stylesheet creation tools need not provide control over the order in which the elements occur.

In addition, the `xsl:stylesheet` element may contain any element not from the XSLT namespace, provided that the expanded-name of the element has a non-null namespace URI. The presence of such top-level elements must not change the behavior of XSLT elements and functions defined in this document; for example, it would not be permitted for such a top-level element to specify that `xsl:apply-templates` was to use different rules to resolve conflicts. Thus, an XSLT processor is always free to ignore such top-level elements, and must ignore a top-level element without giving an error if it does not recognize the namespace URI. Such elements can provide, for example,

- information used by extension elements or extension functions (see [**14 Extensions**]),
- information about what to do with the result tree,
- information about how to obtain the source tree,
- metadata about the stylesheet,

- structured documentation for the stylesheet.

2.3 Literal Result Element as Stylesheet

A simplified syntax is allowed for stylesheets that consist of only a single template for the root node. The stylesheet may consist of just a literal result element (see [7.1.1 Literal Result Elements]). Such a stylesheet is equivalent to a stylesheet with an `xsl:stylesheet` element containing a template rule containing the literal result element; the template rule has a match pattern of `/`. For example

```
<html xsl:version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/
Transform"
      xmlns="http://www.w3.org/TR/xhtml1/strict">
  <head>
    <title>Expense Report Summary</title>
  </head>
  <body>
    <p>Total Amount: <xsl:value-of select="expense-
report/total"/></p>
  </body>
</html>
```

has the same meaning as

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/
XSL/Transform"
                xmlns="http://www.w3.org/TR/xhtml1/
strict">
<xsl:template match="/">
<html>
  <head>
    <title>Expense Report Summary</title>
  </head>
  <body>
    <p>Total Amount: <xsl:value-of select="expense-
report/total"/></p>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

A literal result element that is the document element of a stylesheet must

have an `xsl:version` attribute, which indicates the version of XSLT that the stylesheet requires. For this version of XSLT, the value should be `1.0`; the value must be a Number. Other literal result elements may also have an `xsl:version` attribute. When the `xsl:version` attribute is not equal to `1.0`, forwards-compatible processing mode is enabled (see [**2.5 Forwards-Compatible Processing**]).

The allowed content of a literal result element when used as a stylesheet is no different from when it occurs within a stylesheet. Thus, a literal result element used as a stylesheet cannot contain top-level elements.

In some situations, the only way that a system can recognize that an XML document needs to be processed by an XSLT processor as an XSLT stylesheet is by examining the XML document itself. Using the simplified syntax makes this harder.

> **NOTE:**For example, another XML language (AXL) might also use an `axl:version` on the document element to indicate that an XML document was an AXL document that required processing by an AXL processor; if a document had both an `axl:version` attribute and an `xsl:version` attribute, it would be unclear whether the document should be processed by an XSLT processor or an AXL processor.

Therefore, the simplified syntax should not be used for XSLT stylesheets that may be used in such a situation. This situation can, for example, arise when an XSLT stylesheet is transmitted as a message with a MIME media type of `text/xml` or `application/xml` to a recipient that will use the MIME media type to determine how the message is processed.

2.4 Qualified Names

The name of an internal XSLT object, specifically a named template (see [**6 Named Templates**]), a mode (see [**5.7 Modes**]), an attribute set (see [**7.1.4 Named Attribute Sets**]), a key (see [**12.2 Keys**]), a decimal-format (see [**12.3 Number Formatting**]), a variable or a parameter (see [**11 Variables and Parameters**]) is specified as a QName. If it has a prefix, then the prefix is expanded into a URI reference using the namespace declarations in effect on the attribute in which the name occurs. The expanded-name consisting of the local part of the name and the possibly null URI reference is used as the name of the object. The default namespace is *not* used for unprefixed names.

2.5 Forwards-Compatible Processing

An element enables forwards-compatible mode for itself, its attributes, its descendants and their attributes if either it is an `xsl:stylesheet` element whose `version` attribute is not equal to `1.0`, or it is a literal result element that has an `xsl:version` attribute whose value is not equal to `1.0`, or it is a literal result element that does not have an `xsl:version` attribute and that is the document element of a stylesheet using the simplified syntax (see [**2.3 Literal Result Element as Stylesheet**]). A literal result element that has an `xsl:version` attribute whose value is equal to `1.0` disables forwards-compatible mode for itself, its attributes, its descendants and their attributes.

If an element is processed in forwards-compatible mode, then:

- if it is a top-level element and XSLT 1.0 does not allow such elements as top-level elements, then the element must be ignored along with its content;
- if it is an element in a template and XSLT 1.0 does not allow such elements to occur in templates, then if the element is not instantiated, an error must not be signaled, and if the element is instantiated, the XSLT must perform fallback for the element as specified in [**15 Fallback**];
- if the element has an attribute that XSLT 1.0 does not allow the element to have or if the element has an optional attribute with a value that the XSLT 1.0 does not allow the attribute to have, then the attribute must be ignored.

Thus, any XSLT 1.0 processor must be able to process the following stylesheet without error, although the stylesheet includes elements from the XSLT namespace that are not defined in this specification:

```
<xsl:stylesheet version="1.1"
                xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">
  <xsl:template match="/">
    <xsl:choose>
      <xsl:when test="system-property('xsl:version')
>= 1.1">
        <xsl:exciting-new-1.1-feature/>
      </xsl:when>
      <xsl:otherwise>
        <html>
        <head>
          <title>XSLT 1.1 required</title>
```

```
        </head>
        <body>
          <p>Sorry, this stylesheet requires XSLT
1.1.</p>
        </body>
        </html>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

NOTE: If a stylesheet depends crucially on a top-level element introduced by a version of XSL after 1.0, then the stylesheet can use an `xsl:message` element with `terminate="yes"` (see [**13 Messages**]) to ensure that XSLT processors implementing earlier versions of XSL will not silently ignore the top-level element. For example,

```
<xsl:stylesheet version="1.5"
                xmlns:xsl="http://www.w3.
org/1999/XSL/Transform">

  <xsl:important-new-1.1-declaration/>

  <xsl:template match="/">
    <xsl:choose>
      <xsl:when test="system-property('xsl:
version') &lt; 1.1">
        <xsl:message terminate="yes">
          <xsl:text>Sorry, this stylesheet
requires XSLT 1.1.</xsl:text>
        </xsl:message>
      </xsl:when>
      <xsl:otherwise>
        ...
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  ...
</xsl:stylesheet>
```

If an expression occurs in an attribute that is processed in forwards-compatible mode, then an XSLT processor must recover from errors in the expression as follows:

- if the expression does not match the syntax allowed by the XPath grammar, then an error must not be signaled unless the expression is actually evaluated;

- if the expression calls a function with an unprefixed name that is not part of the XSLT library, then an error must not be signaled unless the function is actually called;

- if the expression calls a function with a number of arguments that XSLT does not allow or with arguments of types that XSLT does not allow, then an error must not be signaled unless the function is actually called.

2.6 Combining Stylesheets

XSLT provides two mechanisms to combine stylesheets:

- an inclusion mechanism that allows stylesheets to be combined without changing the semantics of the stylesheets being combined, and
- an import mechanism that allows stylesheets to override each other.

2.6.1 Stylesheet Inclusion

```
<!-- Category: top-level-element -->
<xsl:include
  href = uri-reference />
```

An XSLT stylesheet may include another XSLT stylesheet using an `xsl:include` element. The `xsl:include` element has an `href` attribute whose value is a URI reference identifying the stylesheet to be included. A relative URI is resolved relative to the base URI of the `xsl:include` element (see [3.2 Base URI]).

The `xsl:include` element is only allowed as a top-level element.

The inclusion works at the XML tree level. The resource located by the `href` attribute value is parsed as an XML document, and the children of the `xsl:stylesheet` element in this document replace the `xsl:include` element in the including document. The fact that template rules or definitions are included does not affect the way they are processed.

The included stylesheet may use the simplified syntax described in [2.3 Literal Result Element as Stylesheet]. The included stylesheet is treated the same as the equivalent `xsl:stylesheet` element.

It is an error if a stylesheet directly or indirectly includes itself.

> **NOTE:**Including a stylesheet multiple times can cause errors because of duplicate definitions. Such multiple inclusions are less obvious when they are indirect. For example, if stylesheet *B* includes stylesheet *A*, stylesheet *C* includes stylesheet *A*, and stylesheet *D* includes both stylesheet *B* and stylesheet *C*, then *A* will be included indirectly by *D* twice. If all of *B*, *C* and *D* are used as independent stylesheets, then the error can be avoided by separating everything in *B* other than the inclusion of *A* into a separate stylesheet *B'* and changing *B* to contain just inclusions of *B'* and *A*, similarly for *C*, and then changing *D* to include *A*, *B'*, *C'*.

2.6.2 Stylesheet Import

```
<xsl:import
  href = uri-reference />
```

An XSLT stylesheet may import another XSLT stylesheet using an `xsl:import` element. Importing a stylesheet is the same as including it (see [2.6.1 Stylesheet Inclusion]) except that definitions and template rules in the importing stylesheet take precedence over template rules and definitions in the imported stylesheet; this is described in more detail below. The `xsl:import` element has an `href` attribute whose value is a URI reference identifying the stylesheet to be imported. A relative URI is resolved relative to the base URI of the `xsl:import` element (see [3.2 Base URI]).

The `xsl:import` element is only allowed as a top-level element. The `xsl:import` element children must precede all other element children of an `xsl:stylesheet` element, including any `xsl:include` element children. When `xsl:include` is used to include a stylesheet, any `xsl:import` elements in the included document are moved up in the including document to after any existing `xsl:import` elements in the including document.

For example,

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">
  <xsl:import href="article.xsl"/>
  <xsl:import href="bigfont.xsl"/>
  <xsl:attribute-set name="note-style">
    <xsl:attribute name="font-style">italic</xsl:
```

```
attribute>
  </xsl:attribute-set>
</xsl:stylesheet>
```

The `xsl:stylesheet` elements encountered during processing of a stylesheet that contains `xsl:import` elements are treated as forming an **import tree**. In the import tree, each `xsl:stylesheet` element has one import child for each `xsl:import` element that it contains. Any `xsl:include` elements are resolved before constructing the import tree. An `xsl:stylesheet` element in the import tree is defined to have lower **import precedence** than another `xsl:stylesheet` element in the import tree if it would be visited before that `xsl:stylesheet` element in a post-order traversal of the import tree (i.e. a traversal of the import tree in which an `xsl:stylesheet` element is visited after its import children). Each definition and template rule has import precedence determined by the `xsl:stylesheet` element that contains it.

For example, suppose

- stylesheet *A* imports stylesheets *B* and *C* in that order;
- stylesheet *B* imports stylesheet *D*;
- stylesheet *C* imports stylesheet *E*.

Then the order of import precedence (lowest first) is *D*, *B*, *E*, *C*, *A*.

> **NOTE:**Since `xsl:import` elements are required to occur before any definitions or template rules, an implementation that processes imported stylesheets at the point at which it encounters the `xsl:import` element will encounter definitions and template rules in increasing order of import precedence.

In general, a definition or template rule with higher import precedence takes precedence over a definition or template rule with lower import precedence. This is defined in detail for each kind of definition and for template rules.

It is an error if a stylesheet directly or indirectly imports itself. Apart from this, the case where a stylesheet with a particular URI is imported in multiple places is not treated specially. The import tree will have a separate `xsl:stylesheet` for each place that it is imported.

> **NOTE:**If `xsl:apply-imports` is used (see **[5.6 Overriding Template Rules]**), the behavior may be different from the

behavior if the stylesheet had been imported only at the place with the highest import precedence.

2.7 Embedding Stylesheets

Normally an XSLT stylesheet is a complete XML document with the `xsl:stylesheet` element as the document element. However, an XSLT stylesheet may also be embedded in another resource. Two forms of embedding are possible:

- the XSLT stylesheet may be textually embedded in a non-XML resource, or
- the `xsl:stylesheet` element may occur in an XML document other than as the document element.

To facilitate the second form of embedding, the `xsl:stylesheet` element is allowed to have an ID attribute that specifies a unique identifier.

NOTE: In order for such an attribute to be used with the XPath **id** function, it must actually be declared in the DTD as being an ID.

The following example shows how the `xml-stylesheet` processing instruction [XML Stylesheet] can be used to allow a document to contain its own stylesheet. The URI reference uses a relative URI with a fragment identifier to locate the `xsl:stylesheet` element:

```
<?xml-stylesheet type="text/xml" href="#style1"?>
<!DOCTYPE doc SYSTEM "doc.dtd">
<doc>
<head>
<xsl:stylesheet id="style1"
                version="1.0"
                xmlns:xsl="http://www.w3.org/1999/
XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/
Format">
<xsl:import href="doc.xsl"/>
<xsl:template match="id('foo')">
  <fo:block font-weight="bold"><xsl:apply-templates/
></fo:block>
</xsl:template>
<xsl:template match="xsl:stylesheet">
  <!-- ignore -->
</xsl:template>
</xsl:stylesheet>
```

```
</head>
<body>
<para id="foo">
...
</para>
</body>
</doc>
```

NOTE:A stylesheet that is embedded in the document to which it is to be applied or that may be included or imported into an stylesheet that is so embedded typically needs to contain a template rule that specifies that `xsl:stylesheet` elements are to be ignored.

3 Data Model

The data model used by XSLT is the same as that used by XPath with the additions described in this section. XSLT operates on source, result and stylesheet documents using the same data model. Any two XML documents that have the same tree will be treated the same by XSLT.

Processing instructions and comments in the stylesheet are ignored: the stylesheet is treated as if neither processing instruction nodes nor comment nodes were included in the tree that represents the stylesheet.

3.1 Root Node Children

The normal restrictions on the children of the root node are relaxed for the result tree. The result tree may have any sequence of nodes as children that would be possible for an element node. In particular, it may have text node children, and any number of element node children. When written out using the XML output method (see **[16 Output]**), it is possible that a result tree will not be a well-formed XML document; however, it will always be a well-formed external general parsed entity.

When the source tree is created by parsing a well-formed XML document, the root node of the source tree will automatically satisfy the normal restrictions of having no text node children and exactly one element child. When the source tree is created in some other way, for example by using the DOM, the usual restrictions are relaxed for the source tree as for the result tree.

3.2 Base URI

Every node also has an associated URI called its base URI, which is used for

resolving attribute values that represent relative URIs into absolute URIs. If an element or processing instruction occurs in an external entity, the base URI of that element or processing instruction is the URI of the external entity; otherwise, the base URI is the base URI of the document. The base URI of the document node is the URI of the document entity. The base URI for a text node, a comment node, an attribute node or a namespace node is the base URI of the parent of the node.

3.3 Unparsed Entities

The root node has a mapping that gives the URI for each unparsed entity declared in the document's DTD. The URI is generated from the system identifier and public identifier specified in the entity declaration. The XSLT processor may use the public identifier to generate a URI for the entity instead of the URI specified in the system identifier. If the XSLT processor does not use the public identifier to generate the URI, it must use the system identifier; if the system identifier is a relative URI, it must be resolved into an absolute URI using the URI of the resource containing the entity declaration as the base URI [RFC2396].

3.4 Whitespace Stripping

After the tree for a source document or stylesheet document has been constructed, but before it is otherwise processed by XSLT, some text nodes are stripped. A text node is never stripped unless it contains only whitespace characters. Stripping the text node removes the text node from the tree. The stripping process takes as input a set of element names for which whitespace must be preserved. The stripping process is applied to both stylesheets and source documents, but the set of whitespace-preserving element names is determined differently for stylesheets and for source documents.

A text node is preserved if any of the following apply:

- The element name of the parent of the text node is in the set of whitespace-preserving element names.
- The text node contains at least one non-whitespace character. As in XML, a whitespace character is #x20, #x9, #xD or #xA.
- An ancestor element of the text node has an `xml:space` attribute with a value of `preserve`, and no closer ancestor element has `xml:space` with a value of `default`.

Otherwise, the text node is stripped.

The `xml:space` attributes are not stripped from the tree.

> **NOTE:**This implies that if an `xml:space` attribute is specified on a literal result element, it will be included in the result.

For stylesheets, the set of whitespace-preserving element names consists of just `xsl:text`.

```
<!-- Category: top-level-element -->
<xsl:strip-space
  elements = tokens />
```

```
<!-- Category: top-level-element -->
<xsl:preserve-space
  elements = tokens />
```

For source documents, the set of whitespace-preserving element names is specified by `xsl:strip-space` and `xsl:preserve-space` top-level elements. These elements each have an `elements` attribute whose value is a whitespace-separated list of NameTests. Initially, the set of whitespace-preserving element names contains all element names. If an element name matches a NameTest in an `xsl:strip-space` element, then it is removed from the set of whitespace-preserving element names. If an element name matches a NameTest in an `xsl:preserve-space` element, then it is added to the set of whitespace-preserving element names. An element matches a NameTest if and only if the NameTest would be true for the element as an XPath node test. Conflicts between matches to `xsl:strip-space` and `xsl:preserve-space` elements are resolved the same way as conflicts between template rules (see [**5.5 Conflict Resolution for Template Rules**]). Thus, the applicable match for a particular element name is determined as follows:

- First, any match with lower import precedence than another match is ignored.
- Next, any match with a NameTest that has a lower default priority than the default priority of the NameTest of another match is ignored.

It is an error if this leaves more than one match. An XSLT processor may signal the error; if it does not signal the error, it must recover by choosing, from amongst the matches that are left, the one that occurs last in the stylesheet.

4 Expressions

XSLT uses the expression language defined by XPath [XPath]. Expressions are used in XSLT for a variety of purposes including:

- selecting nodes for processing;
- specifying conditions for different ways of processing a node;
- generating text to be inserted in the result tree.

An **expression** must match the XPath production Expr.

Expressions occur as the value of certain attributes on XSLT-defined elements and within curly braces in attribute value templates.

In XSLT, an outermost expression (i.e. an expression that is not part of another expression) gets its context as follows:

- the context node comes from the current node
- the context position comes from the position of the current node in the current node list; the first position is 1
- the context size comes from the size of the current node list
- the variable bindings are the bindings in scope on the element which has the attribute in which the expression occurs (see **[11 Variables and Parameters]**)
- the set of namespace declarations are those in scope on the element which has the attribute in which the expression occurs; this includes the implicit declaration of the prefix `xml` required by the the XML Namespaces Recommendation [XML Names]; the default namespace (as declared by `xmlns`) is not part of this set
- the function library consists of the core function library together with the additional functions defined in **[12 Additional Functions]** and extension functions as described in **[14 Extensions]**; it is an error for an expression to include a call to any other function

5 Template Rules

5.1 Processing Model

A list of source nodes is processed to create a result tree fragment. The result tree is constructed by processing a list containing just the root node. A list of source nodes is processed by appending the result tree structure created by processing each of the members of the list in order. A node is processed by finding all the template rules with patterns that match the node, and choosing the best amongst them; the chosen rule's template is then instantiated with the node as the current node and with the list of source nodes as the current node list. A template typically contains instructions that select an additional list of source nodes for processing. The process of matching, instantiation and selection is continued recursively until no new source nodes are selected for processing.

Implementations are free to process the source document in any way that produces the same result as if it were processed using this processing model.

5.2 Patterns

Template rules identify the nodes to which they apply by using a **pattern**. As well as being used in template rules, patterns are used for numbering (see **[7.7 Numbering]**) and for declaring keys (see **[12.2 Keys]**). A pattern specifies a set of conditions on a node. A node that satisfies the conditions matches the pattern; a node that does not satisfy the conditions does not match the pattern. The syntax for patterns is a subset of the syntax for expressions. In particular, location paths that meet certain restrictions can be used as patterns. An expression that is also a pattern always evaluates to an object of type node-set. A node matches a pattern if the node is a member of the result of evaluating the pattern as an expression with respect to some possible context; the possible contexts are those whose context node is the node being matched or one of its ancestors.

Here are some examples of patterns:

- `para` matches any `para` element
- `*` matches any element
- `chapter|appendix` matches any `chapter` element and any `appendix` element
- `olist/item` matches any `item` element with an `olist` parent
- `appendix//para` matches any `para` element with an `appendix` ancestor element

- `/` matches the root node
- `text()` matches any text node
- `processing-instruction()` matches any processing instruction
- `node()` matches any node other than an attribute node and the root node
- `id("W11")` matches the element with unique ID `W11`
- `para[1]` matches any `para` element that is the first `para` child element of its parent
- `*[position()=1 and self::para]` matches any `para` element that is the first child element of its parent
- `para[last()=1]` matches any `para` element that is the only `para` child element of its parent
- `items/item[position()>1]` matches any `item` element that has a `items` parent and that is not the first `item` child of its parent
- `item[position() mod 2 = 1]` would be true for any `item` element that is an odd-numbered `item` child of its parent.
- `div[@class="appendix"]//p` matches any `p` element with a `div` ancestor element that has a `class` attribute with value `appendix`
- `@class` matches any `class` attribute (*not* any element that has a `class` attribute)
- `@*` matches any attribute

A pattern must match the grammar for Pattern. A Pattern is a set of location path patterns separated by `|`. A location path pattern is a location path whose steps all use only the `child` or `attribute` axes. Although patterns must not use the `descendant-or-self` axis, patterns may use the `//` operator as well as the `/` operator. Location path patterns can also start with an **id** or **key** function call with a literal argument. Predicates in a pattern can use arbitrary expressions just like predicates in a location path.

Patterns

[1]	Pattern	:: =	LocationPathPattern | Pattern '|' LocationPathPattern
[2]	LocationPathPattern	:: =	'/' RelativePathPattern? | IdKeyPattern (('/' | '//') RelativePathPattern)? | '//'? RelativePathPattern
[3]	IdKeyPattern	:: =	'id' '(' Literal ')' | 'key' '(' Literal ',' Literal ')'
[4]	RelativePathPattern	:: =	StepPattern | RelativePathPattern '/' StepPattern | RelativePathPattern '//' StepPattern
[5]	StepPattern	:: =	ChildOrAttributeAxisSpecifier NodeTest Predicate*
[6]	ChildOrAttributeAxisSpecifier	:: =	AbbreviatedAxisSpecifier | ('child' | 'attribute') '::'

A pattern is defined to match a node if and only if there is possible context such that when the pattern is evaluated as an expression with that context, the node is a member of the resulting node-set. When a node is being matched, the possible contexts have a context node that is the node being matched or any ancestor of that node, and a context node list containing just the context node.

For example, p matches any p element, because for any p if the expression p is evaluated with the parent of the p element as context the resulting node-set will contain that p element as one of its members.

> **NOTE:**This matches even a p element that is the document element, since the document root is the parent of the document element.

Although the semantics of patterns are specified indirectly in terms of expression evaluation, it is easy to understand the meaning of a pattern directly without thinking in terms of expression evaluation. In a pattern, | indicates alternatives; a pattern with one or more | separated alternatives matches if any one of the alternative matches. A pattern that consists of a

sequence of StepPatterns separated by / or // is matched from right to left. The pattern only matches if the rightmost StepPattern matches and a suitable element matches the rest of the pattern; if the separator is / then only the parent is a suitable element; if the separator is //, then any ancestor is a suitable element. A StepPattern that uses the child axis matches if the NodeTest is true for the node and the node is not an attribute node. A StepPattern that uses the attribute axis matches if the NodeTest is true for the node and the node is an attribute node. When [] is present, then the first PredicateExpr in a StepPattern is evaluated with the node being matched as the context node and the siblings of the context node that match the NodeTest as the context node list, unless the node being matched is an attribute node, in which case the context node list is all the attributes that have the same parent as the attribute being matched and that match the NameTest.

For example

```
appendix//ulist/item[position()=1]
```

matches a node if and only if all of the following are true:

- the NodeTest `item` is true for the node and the node is not an attribute; in other words the node is an `item` element
- evaluating the PredicateExpr `position()=1` with the node as context node and the siblings of the node that are `item` elements as the context node list yields true
- the node has a parent that matches `appendix//ulist`; this will be true if the parent is a `ulist` element that has an `appendix` ancestor element.

5.3 Defining Template Rules

```
<!-- Category: top-level-element -->
<xsl:template
  match = pattern
  name = qname
  priority = number
  mode = qname>
  <!-- Content: (xsl:param*, template) -->
</xsl:template>
```

A template rule is specified with the `xsl:template` element. The `match` attribute is a Pattern that identifies the source node or nodes to which the rule applies. The `match` attribute is required unless the `xsl:template` element has a `name` attribute (see [**6 Named Templates**]). It is an error for the value of the `match` attribute to contain a VariableReference. The content of the `xsl:template` element is the template that is instantiated when the template rule is applied.

For example, an XML document might contain:

```
This is an <emph>important</emph> point.
```

The following template rule matches `emph` elements and produces a `fo:inline-sequence` formatting object with a `font-weight` property of `bold`.

```
<xsl:template match="emph">
  <fo:inline-sequence font-weight="bold">
    <xsl:apply-templates/>
  </fo:inline-sequence>
</xsl:template>
```

NOTE: Examples in this document use the `fo:` prefix for the namespace `http://www.w3.org/1999/XSL/Format`, which is the namespace of the formatting objects defined in [XSL].

As described next, the `xsl:apply-templates` element recursively processes the children of the source element.

5.4 Applying Template Rules

```
<!-- Category: instruction -->
<xsl:apply-templates
  select = node-set-expression
  mode = qname>
  <!-- Content: (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>
```

This example creates a block for a `chapter` element and then processes its immediate children.

```
<xsl:template match="chapter">
  <fo:block>
    <xsl:apply-templates/>
  </fo:block>
```

```
</xsl:template>
```

In the absence of a `select` attribute, the `xsl:apply-templates` instruction processes all of the children of the current node, including text nodes. However, text nodes that have been stripped as specified in [**3.4 Whitespace Stripping**] will not be processed. If stripping of whitespace nodes has not been enabled for an element, then all whitespace in the content of the element will be processed as text, and thus whitespace between child elements will count in determining the position of a child element as returned by the **position** function.

A `select` attribute can be used to process nodes selected by an expression instead of processing all children. The value of the `select` attribute is an expression. The expression must evaluate to a node-set. The selected set of nodes is processed in document order, unless a sorting specification is present (see [**10 Sorting**]). The following example processes all of the `author` children of the `author-group`:

```
<xsl:template match="author-group">
  <fo:inline-sequence>
    <xsl:apply-templates select="author"/>
  </fo:inline-sequence>
</xsl:template>
```

The following example processes all of the `given-names` of the `authors` that are children of `author-group`:

```
<xsl:template match="author-group">
  <fo:inline-sequence>
    <xsl:apply-templates select="author/given-name"/>
  </fo:inline-sequence>
</xsl:template>
```

This example processes all of the `heading` descendant elements of the `book` element.

```
<xsl:template match="book">
  <fo:block>
    <xsl:apply-templates select=".//heading"/>
  </fo:block>
</xsl:template>
```

It is also possible to process elements that are not descendants of the current node. This example assumes that a `department` element has `group`

children and `employee` descendants. It finds an employee's department and then processes the `group` children of the `department`.

```
<xsl:template match="employee">
  <fo:block>
    Employee <xsl:apply-templates select="name"/>
belongs to group
    <xsl:apply-templates select="ancestor::
department/group"/>
  </fo:block>
</xsl:template>
```

Multiple `xsl:apply-templates` elements can be used within a single template to do simple reordering. The following example creates two HTML tables. The first table is filled with domestic sales while the second table is filled with foreign sales.

```
<xsl:template match="product">
  <table>
    <xsl:apply-templates select="sales/domestic"/>
  </table>
  <table>
    <xsl:apply-templates select="sales/foreign"/>
  </table>
</xsl:template>
```

NOTE: It is possible for there to be two matching descendants where one is a descendant of the other. This case is not treated specially: both descendants will be processed as usual. For example, given a source document

```
<doc><div><div></div></div></doc>
```

the rule

```
<xsl:template match="doc">
  <xsl:apply-templates select=".//div"/>
</xsl:template>
```

will process both the outer `div` and inner `div` elements.

NOTE: Typically, `xsl:apply-templates` is used to process only nodes that are descendants of the current node. Such use of `xsl:apply-templates` cannot result in non-terminating processing loops. However, when `xsl:apply-templates` is

used to process elements that are not descendants of the current node, the possibility arises of non-terminating loops. For example,

```
<xsl:template match="foo">
  <xsl:apply-templates select="."/>
</xsl:template>
```

Implementations may be able to detect such loops in some cases, but the possibility exists that a stylesheet may enter a non-terminating loop that an implementation is unable to detect. This may present a denial of service security risk.

5.5 Conflict Resolution for Template Rules

It is possible for a source node to match more than one template rule. The template rule to be used is determined as follows:

1. First, all matching template rules that have lower import precedence than the matching template rule or rules with the highest import precedence are eliminated from consideration.

2. Next, all matching template rules that have lower priority than the matching template rule or rules with the highest priority are eliminated from consideration. The priority of a template rule is specified by the `priority` attribute on the template rule. The value of this must be a real number (positive or negative), matching the production Number with an optional leading minus sign (`-`). The **default priority** is computed as follows:

   - If the pattern contains multiple alternatives separated by `|`, then it is treated equivalently to a set of template rules, one for each alternative.

   - If the pattern has the form of a QName preceded by a ChildOrAttributeAxisSpecifier or has the form `processing-instruction(`Literal`)` preceded by a ChildOrAttributeAxisSpecifier, then the priority is 0.

   - If the pattern has the form NCName`:*` preceded by a ChildOrAttributeAxisSpecifier, then the priority is -0.25.

   - Otherwise, if the pattern consists of just a NodeTest preceded by

a ChildOrAttributeAxisSpecifier, then the priority is -0.5.

- Otherwise, the priority is 0.5.

Thus, the most common kind of pattern (a pattern that tests for a node with a particular type and a particular expanded-name) has priority 0. The next less specific kind of pattern (a pattern that tests for a node with a particular type and an expanded-name with a particular namespace URI) has priority -0.25. Patterns less specific than this (patterns that just tests for nodes with particular types) have priority -0.5. Patterns more specific than the most common kind of pattern have priority 0.5.

It is an error if this leaves more than one matching template rule. An XSLT processor may signal the error; if it does not signal the error, it must recover by choosing, from amongst the matching template rules that are left, the one that occurs last in the stylesheet.

5.6 Overriding Template Rules

```
<!-- Category: instruction -->
<xsl:apply-imports />
```

A template rule that is being used to override a template rule in an imported stylesheet (see [**5.5 Conflict Resolution for Template Rules**]) can use the `xsl:apply-imports` element to invoke the overridden template rule.

At any point in the processing of a stylesheet, there is a **current template rule**. Whenever a template rule is chosen by matching a pattern, the template rule becomes the current template rule for the instantiation of the rule's template. When an `xsl:for-each` element is instantiated, the current template rule becomes null for the instantiation of the content of the `xsl:for-each` element.

`xsl:apply-imports` processes the current node using only template rules that were imported into the stylesheet element containing the current template rule; the node is processed in the current template rule's mode. It is an error if `xsl:apply-imports` is instantiated when the current template rule is null.

For example, suppose the stylesheet `doc.xsl` contains a template rule for `example` elements:

```
<xsl:template match="example">
  <pre><xsl:apply-templates/></pre>
```

```
</xsl:template>
```

Another stylesheet could import `doc.xsl` and modify the treatment of `example` elements as follows:

```
<xsl:import href="doc.xsl"/>

<xsl:template match="example">
  <div style="border: solid red">
     <xsl:apply-imports/>
  </div>
</xsl:template>
```

The combined effect would be to transform an `example` into an element of the form:

```
<div style="border: solid red"><pre>...</pre></div>
```

5.7 Modes

Modes allow an element to be processed multiple times, each time producing a different result.

Both `xsl:template` and `xsl:apply-templates` have an optional `mode` attribute. The value of the `mode` attribute is a QName, which is expanded as described in [**2.4 Qualified Names**]. If `xsl:template` does not have a `match` attribute, it must not have a `mode` attribute. If an `xsl:apply-templates` element has a `mode` attribute, then it applies only to those template rules from `xsl:template` elements that have a `mode` attribute with the same value; if an `xsl:apply-templates` element does not have a `mode` attribute, then it applies only to those template rules from `xsl:template` elements that do not have a `mode` attribute.

5.8 Built-in Template Rules

There is a built-in template rule to allow recursive processing to continue in the absence of a successful pattern match by an explicit template rule in the stylesheet. This template rule applies to both element nodes and the root node. The following shows the equivalent of the built-in template rule:

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

There is also a built-in template rule for each mode, which allows recursive processing to continue in the same mode in the absence of a successful pattern match by an explicit template rule in the stylesheet. This template rule applies to both element nodes and the root node. The following shows the equivalent of the built-in template rule for mode *m*.

```
<xsl:template match="*|/" mode="m">
  <xsl:apply-templates mode="m"/>
</xsl:template>
```

There is also a built-in template rule for text and attribute nodes that copies text through:

```
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
```

The built-in template rule for processing instructions and comments is to do nothing.

```
<xsl:template match="processing-instruction()|comment
()"/>
```

The built-in template rule for namespace nodes is also to do nothing. There is no pattern that can match a namespace node; so, the built-in template rule is the only template rule that is applied for namespace nodes.

The built-in template rules are treated as if they were imported implicitly before the stylesheet and so have lower import precedence than all other template rules. Thus, the author can override a built-in template rule by including an explicit template rule.

6 Named Templates

```
<!-- Category: instruction -->
<xsl:call-template
  name = qname>
  <!-- Content: xsl:with-param* -->
</xsl:call-template>
```

Templates can be invoked by name. An `xsl:template` element with a `name` attribute specifies a named template. The value of the `name` attribute is a QName, which is expanded as described in [**2.4 Qualified Names**]. If an `xsl:template` element has a `name` attribute, it may, but need not, also have a

match attribute. An xsl:call-template element invokes a template by name; it has a required name attribute that identifies the template to be invoked. Unlike xsl:apply-templates, xsl:call-template does not change the current node or the current node list.

The match, mode and priority attributes on an xsl:template element do not affect whether the template is invoked by an xsl:call-template element. Similarly, the name attribute on an xsl:template element does not affect whether the template is invoked by an xsl:apply-templates element.

It is an error if a stylesheet contains more than one template with the same name and same import precedence.

7 Creating the Result Tree

This section describes instructions that directly create nodes in the result tree.

7.1 Creating Elements and Attributes

7.1.1 Literal Result Elements

In a template, an element in the stylesheet that does not belong to the XSLT namespace and that is not an extension element (see [**14.1 Extension Elements**]) is instantiated to create an element node with the same expanded-name. The content of the element is a template, which is instantiated to give the content of the created element node. The created element node will have the attribute nodes that were present on the element node in the stylesheet tree, other than attributes with names in the XSLT namespace.

The created element node will also have a copy of the namespace nodes that were present on the element node in the stylesheet tree with the exception of any namespace node whose string-value is the XSLT namespace URI (http://www.w3.org/1999/XSL/Transform), a namespace URI declared as an extension namespace (see [**14.1 Extension Elements**]), or a namespace URI designated as an excluded namespace. A namespace URI is designated as an excluded namespace by using an exclude-result-prefixes attribute on an xsl:stylesheet element or an xsl:exclude-result-prefixes attribute on a literal result element. The value of both these attributes is a whitespace-separated list of namespace prefixes. The namespace bound to each of the prefixes is designated as an excluded namespace. It is an error if there is no namespace bound to the prefix on the element bearing the exclude-result-prefixes or xsl:exclude-result-prefixes attribute. The default namespace (as declared by

xmlns) may be designated as an excluded namespace by including #default in the list of namespace prefixes. The designation of a namespace as an excluded namespace is effective within the subtree of the stylesheet rooted at the element bearing the exclude-result-prefixes or xsl:exclude-result-prefixes attribute; a subtree rooted at an xsl:stylesheet element does not include any stylesheets imported or included by children of that xsl:stylesheet element.

> **NOTE:**When a stylesheet uses a namespace declaration only for the purposes of addressing the source tree, specifying the prefix in the exclude-result-prefixes attribute will avoid superfluous namespace declarations in the result tree.

The value of an attribute of a literal result element is interpreted as an attribute value template: it can contain expressions contained in curly braces ({}).

A namespace URI in the stylesheet tree that is being used to specify a namespace URI in the result tree is called a **literal namespace URI**. This applies to:

- the namespace URI in the expanded-name of a literal result element in the stylesheet
- the namespace URI in the expanded-name of an attribute specified on a literal result element in the stylesheet
- the string-value of a namespace node on a literal result element in the stylesheet

```
<!-- Category: top-level-element -->
<xsl:namespace-alias
  stylesheet-prefix = prefix | "#default"
  result-prefix = prefix | "#default" />
```

A stylesheet can use the xsl:namespace-alias element to declare that one namespace URI is an **alias** for another namespace URI. When a literal namespace URI has been declared to be an alias for another namespace URI, then the namespace URI in the result tree will be the namespace URI that the literal namespace URI is an alias for, instead of the literal namespace URI itself. The xsl:namespace-alias element declares that the namespace URI bound to the prefix specified by the stylesheet-prefix attribute is an alias for the namespace URI bound to the prefix specified by the result-prefix attribute. Thus, the stylesheet-prefix attribute

specifies the namespace URI that will appear in the stylesheet, and the `result-prefix` attribute specifies the corresponding namespace URI that will appear in the result tree. The default namespace (as declared by `xmlns`) may be specified by using `#default` instead of a prefix. If a namespace URI is declared to be an alias for multiple different namespace URIs, then the declaration with the highest import precedence is used. It is an error if there is more than one such declaration. An XSLT processor may signal the error; if it does not signal the error, it must recover by choosing, from amongst the declarations with the highest import precedence, the one that occurs last in the stylesheet.

When literal result elements are being used to create element, attribute, or namespace nodes that use the XSLT namespace URI, the stylesheet must use an alias. For example, the stylesheet

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:axsl="http://www.w3.org/1999/XSL/
TransformAlias">

<xsl:namespace-alias stylesheet-prefix="axsl" result-
prefix="xsl"/>

<xsl:template match="/">
  <axsl:stylesheet>
    <xsl:apply-templates/>
  </axsl:stylesheet>
</xsl:template>

<xsl:template match="block">
  <axsl:template match="{.}">
     <fo:block><axsl:apply-templates/></fo:block>
  </axsl:template>
</xsl:template>

</xsl:stylesheet>
```

will generate an XSLT stylesheet from a document of the form:

```
<elements>
<block>p</block>
<block>h1</block>
<block>h2</block>
```

```
<block>h3</block>
<block>h4</block>
</elements>
```

NOTE: It may be necessary also to use aliases for namespaces other than the XSLT namespace URI. For example, literal result elements belonging to a namespace dealing with digital signatures might cause XSLT stylesheets to be mishandled by general-purpose security software; using an alias for the namespace would avoid the possibility of such mishandling.

7.1.2 Creating Elements with `xsl:element`

```
<!-- Category: instruction -->
<xsl:element
  name = { qname }
  namespace = { uri-reference }
  use-attribute-sets = qnames>
  <!-- Content: template -->
</xsl:element>
```

The `xsl:element` element allows an element to be created with a computed name. The expanded-name of the element to be created is specified by a required `name` attribute and an optional `namespace` attribute. The content of the `xsl:element` element is a template for the attributes and children of the created element.

The `name` attribute is interpreted as an attribute value template. It is an error if the string that results from instantiating the attribute value template is not a QName. An XSLT processor may signal the error; if it does not signal the error, then it must recover by making the the result of instantiating the `xsl:element` element be the sequence of nodes created by instantiating the content of the `xsl:element` element, excluding any initial attribute nodes. If the `namespace` attribute is not present then the QName is expanded into an expanded-name using the namespace declarations in effect for the `xsl:element` element, including any default namespace declaration.

If the `namespace` attribute is present, then it also is interpreted as an attribute value template. The string that results from instantiating the attribute value template should be a URI reference. It is not an error if the string is not a syntactically legal URI reference. If the string is empty, then the expanded-name of the element has a null namespace URI. Otherwise, the string is used as the namespace URI of the expanded-name of the element to be created. The local part of the QName specified by the `name` attribute is used as the

local part of the expanded-name of the element to be created.

XSLT processors may make use of the prefix of the QName specified in the name attribute when selecting the prefix used for outputting the created element as XML; however, they are not required to do so.

7.1.3 Creating Attributes with `xsl:attribute`

```
<!-- Category: instruction -->
<xsl:attribute
  name = { qname }
  namespace = { uri-reference }>
  <!-- Content: template -->
</xsl:attribute>
```

The `xsl:attribute` element can be used to add attributes to result elements whether created by literal result elements in the stylesheet or by instructions such as `xsl:element`. The expanded-name of the attribute to be created is specified by a required `name` attribute and an optional `namespace` attribute. Instantiating an `xsl:attribute` element adds an attribute node to the containing result element node. The content of the `xsl:attribute` element is a template for the value of the created attribute.

The `name` attribute is interpreted as an attribute value template. It is an error if the string that results from instantiating the attribute value template is not a QName or is the string `xmlns`. An XSLT processor may signal the error; if it does not signal the error, it must recover by not adding the attribute to the result tree. If the `namespace` attribute is not present, then the QName is expanded into an expanded-name using the namespace declarations in effect for the `xsl:attribute` element, *not* including any default namespace declaration.

If the `namespace` attribute is present, then it also is interpreted as an attribute value template. The string that results from instantiating it should be a URI reference. It is not an error if the string is not a syntactically legal URI reference. If the string is empty, then the expanded-name of the attribute has a null namespace URI. Otherwise, the string is used as the namespace URI of the expanded-name of the attribute to be created. The local part of the QName specified by the `name` attribute is used as the local part of the expanded-name of the attribute to be created.

XSLT processors may make use of the prefix of the QName specified in the `name` attribute when selecting the prefix used for outputting the created attribute as XML; however, they are not required to do so and, if the prefix is

xmlns, they must not do so. Thus, although it is not an error to do:

```
<xsl:attribute name="xmlns:xsl"
namespace="whatever">http://www.w3.org/1999/XSL/
Transform</xsl:attribute>
```

it will not result in a namespace declaration being output.

Adding an attribute to an element replaces any existing attribute of that element with the same expanded-name.

The following are all errors:

- Adding an attribute to an element after children have been added to it; implementations may either signal the error or ignore the attribute.
- Adding an attribute to a node that is not an element; implementations may either signal the error or ignore the attribute.
- Creating nodes other than text nodes during the instantiation of the content of the xsl:attribute element; implementations may either signal the error or ignore the offending nodes.

  NOTE: When an xsl:attribute contains a text node with a newline, then the XML output must contain a character reference. For example,

  ```
  <xsl:attribute name="a">x
  y</xsl:attribute>
  ```

  will result in the output

  ```
  a="x&#xA;y"
  ```

  (or with any equivalent character reference). The XML output cannot be

  ```
  a="x
  y"
  ```

  This is because XML 1.0 requires newline characters in attribute values to be normalized into spaces but requires character references to newline characters not to be normalized. The attribute values in the data model represent the attribute value

after normalization. If a newline occurring in an attribute value in the tree were output as a newline character rather than as character reference, then the attribute value in the tree created by reparsing the XML would contain a space not a newline, which would mean that the tree had not been output correctly.

7.1.4 Named Attribute Sets

```
<!-- Category: top-level-element -->
<xsl:attribute-set
  name = qname
  use-attribute-sets = qnames>
  <!-- Content: xsl:attribute* -->
</xsl:attribute-set>
```

The `xsl:attribute-set` element defines a named set of attributes. The `name` attribute specifies the name of the attribute set. The value of the `name` attribute is a QName, which is expanded as described in [**2.4 Qualified Names**]. The content of the `xsl:attribute-set` element consists of zero or more `xsl:attribute` elements that specify the attributes in the set.

Attribute sets are used by specifying a `use-attribute-sets` attribute on `xsl:element`, `xsl:copy` (see [**7.5 Copying**]) or `xsl:attribute-set` elements. The value of the `use-attribute-sets` attribute is a whitespace-separated list of names of attribute sets. Each name is specified as a QName, which is expanded as described in [**2.4 Qualified Names**]. Specifying a `use-attribute-sets` attribute is equivalent to adding `xsl:attribute` elements for each of the attributes in each of the named attribute sets to the beginning of the content of the element with the `use-attribute-sets` attribute, in the same order in which the names of the attribute sets are specified in the `use-attribute-sets` attribute. It is an error if use of `use-attribute-sets` attributes on `xsl:attribute-set` elements causes an attribute set to directly or indirectly use itself.

Attribute sets can also be used by specifying an `xsl:use-attribute-sets` attribute on a literal result element. The value of the `xsl:use-attribute-sets` attribute is a whitespace-separated list of names of attribute sets. The `xsl:use-attribute-sets` attribute has the same effect as the `use-attribute-sets` attribute on `xsl:element` with the additional rule that attributes specified on the literal result element itself are treated as if they were specified by `xsl:attribute` elements before any actual `xsl:attribute` elements but after any `xsl:attribute` elements implied by the `xsl:use-attribute-sets` attribute. Thus, for a literal result element, attributes from attribute sets named in an `xsl:use-attribute-sets`

attribute will be added first, in the order listed in the attribute; next, attributes specified on the literal result element will be added; finally, any attributes specified by `xsl:attribute` elements will be added. Since adding an attribute to an element replaces any existing attribute of that element with the same name, this means that attributes specified in attribute sets can be overridden by attributes specified on the literal result element itself.

The template within each `xsl:attribute` element in an `xsl:attribute-set` element is instantiated each time the attribute set is used; it is instantiated using the same current node and current node list as is used for instantiating the element bearing the `use-attribute-sets` or `xsl:use-attribute-sets` attribute. However, it is the position in the stylesheet of the `xsl:attribute` element rather than of the element bearing the `use-attribute-sets` or `xsl:use-attribute-sets` attribute that determines which variable bindings are visible (see **[11 Variables and Parameters]**); thus, only variables and parameters declared by top-level `xsl:variable` and `xsl:param` elements are visible.

The following example creates a named attribute set `title-style` and uses it in a template rule.

```
<xsl:template match="chapter/heading">
  <fo:block quadding="start" xsl:use-attribute-
sets="title-style">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:attribute-set name="title-style">
  <xsl:attribute name="font-size">12pt</xsl:
attribute>
  <xsl:attribute name="font-weight">bold</xsl:
attribute>
</xsl:attribute-set>
```

Multiple definitions of an attribute set with the same expanded-name are merged. An attribute from a definition that has higher import precedence takes precedence over an attribute from a definition that has lower import precedence. It is an error if there are two attribute sets that have the same expanded-name and equal import precedence and that both contain the same attribute, unless there is a definition of the attribute set with higher import precedence that also contains the attribute. An XSLT processor may signal the error; if it does not signal the error, it must recover by choosing from amongst the definitions that specify the attribute that have the highest import precedence the one that was specified last in the stylesheet. Where the

attributes in an attribute set were specified is relevant only in merging the attributes into the attribute set; it makes no difference when the attribute set is used.

7.2 Creating Text

A template can also contain text nodes. Each text node in a template remaining after whitespace has been stripped as specified in [3.4 Whitespace Stripping] will create a text node with the same string-value in the result tree. Adjacent text nodes in the result tree are automatically merged.

Note that text is processed at the tree level. Thus, markup of `<` in a template will be represented in the stylesheet tree by a text node that includes the character <. This will create a text node in the result tree that contains a < character, which will be represented by the markup `<` (or an equivalent character reference) when the result tree is externalized as an XML document (unless output escaping is disabled as described in [16.4 Disabling Output Escaping]).

```
<!-- Category: instruction -->
<xsl:text
  disable-output-escaping = "yes" | "no">
  <!-- Content: #PCDATA -->
</xsl:text>
```

Literal data characters may also be wrapped in an `xsl:text` element. This wrapping may change what whitespace characters are stripped (see [3.4 Whitespace Stripping]) but does not affect how the characters are handled by the XSLT processor thereafter.

> **NOTE:**The `xml:lang` and `xml:space` attributes are not treated specially by XSLT. In particular,
>
> - it is the responsibility of the stylesheet author explicitly to generate any `xml:lang` or `xml:space` attributes that are needed in the result;
> - specifying an `xml:lang` or `xml:space` attribute on an element in the XSLT namespace will not cause any `xml:lang` or `xml:space` attributes to appear in the result.

7.3 Creating Processing Instructions

```
<!-- Category: instruction -->
<xsl:processing-instruction
  name = { ncname }>
  <!-- Content: template -->
</xsl:processing-instruction>
```

The xsl:processing-instruction element is instantiated to create a processing instruction node. The content of the xsl:processing-instruction element is a template for the string-value of the processing instruction node. The xsl:processing-instruction element has a required name attribute that specifies the name of the processing instruction node. The value of the name attribute is interpreted as an attribute value template.

For example, this

```
<xsl:processing-instruction name="xml-
stylesheet">href="book.css" type="text/css"</xsl:
processing-instruction>
```

would create the processing instruction

```
<?xml-stylesheet href="book.css" type="text/css"?>
```

It is an error if the string that results from instantiating the name attribute is not both an NCName and a PITarget. An XSLT processor may signal the error; if it does not signal the error, it must recover by not adding the processing instruction to the result tree.

> **NOTE:**This means that xsl:processing-instruction cannot be used to output an XML declaration. The xsl:output element should be used instead (see [**16 Output**]).

It is an error if instantiating the content of xsl:processing-instruction creates nodes other than text nodes. An XSLT processor may signal the error; if it does not signal the error, it must recover by ignoring the offending nodes together with their content.

It is an error if the result of instantiating the content of the xsl:processing-instruction contains the string ?>. An XSLT processor may signal the error; if it does not signal the error, it must recover by inserting a space after any occurrence of ? that is followed by a >.

7.4 Creating Comments

```
<!-- Category: instruction -->
<xsl:comment>
  <!-- Content: template -->
</xsl:comment>
```

The xsl:comment element is instantiated to create a comment node in the result tree. The content of the xsl:comment element is a template for the string-value of the comment node.

For example, this

```
<xsl:comment>This file is automatically generated.
Do not edit!</xsl:comment>
```

would create the comment

```
<!--This file is automatically generated. Do not
edit!-->
```

It is an error if instantiating the content of xsl:comment creates nodes other than text nodes. An XSLT processor may signal the error; if it does not signal the error, it must recover by ignoring the offending nodes together with their content.

It is an error if the result of instantiating the content of the xsl:comment contains the string -- or ends with -. An XSLT processor may signal the error; if it does not signal the error, it must recover by inserting a space after any occurrence of - that is followed by another - or that ends the comment.

7.5 Copying

```
<!-- Category: instruction -->
<xsl:copy
  use-attribute-sets = qnames>
  <!-- Content: template -->
</xsl:copy>
```

The xsl:copy element provides an easy way of copying the current node. Instantiating the xsl:copy element creates a copy of the current node. The namespace nodes of the current node are automatically copied as well, but the attributes and children of the node are not automatically copied. The content of the xsl:copy element is a template for the attributes and children of the created node; the content is instantiated only for nodes of types that

can have attributes or children (i.e. root nodes and element nodes).

The `xsl:copy` element may have a `use-attribute-sets` attribute (see [7.1.4 Named Attribute Sets]). This is used only when copying element nodes.

The root node is treated specially because the root node of the result tree is created implicitly. When the current node is the root node, `xsl:copy` will not create a root node, but will just use the content template.

For example, the identity transformation can be written using `xsl:copy` as follows:

```
<xsl:template match="@*|node()">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

When the current node is an attribute, then if it would be an error to use `xsl:attribute` to create an attribute with the same name as the current node, then it is also an error to use `xsl:copy` (see [7.1.3 Creating Attributes with `xsl:attribute`]).

The following example shows how `xml:lang` attributes can be easily copied through from source to result. If a stylesheet defines the following named template:

```
<xsl:template name="apply-templates-copy-lang">
 <xsl:for-each select="@xml:lang">
   <xsl:copy/>
 </xsl:for-each>
 <xsl:apply-templates/>
</xsl:template>
```

then it can simply do

```
<xsl:call-template name="apply-templates-copy-lang"/>
```

instead of

```
<xsl:apply-templates/>
```

when it wants to copy the `xml:lang` attribute.

7.6 Computing Generated Text

Within a template, the `xsl:value-of` element can be used to compute generated text, for example by extracting text from the source tree or by inserting the value of a variable. The `xsl:value-of` element does this with an expression that is specified as the value of the `select` attribute. Expressions can also be used inside attribute values of literal result elements by enclosing the expression in curly braces (`{}`).

7.6.1 Generating Text with `xsl:value-of`

```
<!-- Category: instruction -->
<xsl:value-of
  select = string-expression
  disable-output-escaping = "yes" | "no" />
```

The `xsl:value-of` element is instantiated to create a text node in the result tree. The required `select` attribute is an expression; this expression is evaluated and the resulting object is converted to a string as if by a call to the **string** function. The string specifies the string-value of the created text node. If the string is empty, no text node will be created. The created text node will be merged with any adjacent text nodes.

The `xsl:copy-of` element can be used to copy a node-set over to the result tree without converting it to a string. See [**11.3 Using Values of Variables and Parameters with `xsl:copy-of`**].

For example, the following creates an HTML paragraph from a `person` element with `given-name` and `family-name` attributes. The paragraph will contain the value of the `given-name` attribute of the current node followed by a space and the value of the `family-name` attribute of the current node.

```
<xsl:template match="person">
  <p>
   <xsl:value-of select="@given-name"/>
   <xsl:text> </xsl:text>
   <xsl:value-of select="@family-name"/>
  </p>
</xsl:template>
```

For another example, the following creates an HTML paragraph from a `person` element with `given-name` and `family-name` children elements. The paragraph will contain the string-value of the first `given-name` child

element of the current node followed by a space and the string-value of the first `family-name` child element of the current node.

```
<xsl:template match="person">
  <p>
   <xsl:value-of select="given-name"/>
   <xsl:text> </xsl:text>
   <xsl:value-of select="family-name"/>
  </p>
</xsl:template>
```

The following precedes each `procedure` element with a paragraph containing the security level of the procedure. It assumes that the security level that applies to a procedure is determined by a `security` attribute on the procedure element or on an ancestor element of the procedure. It also assumes that if more than one such element has a `security` attribute then the security level is determined by the element that is closest to the procedure.

```
<xsl:template match="procedure">
  <fo:block>
    <xsl:value-of select="ancestor-or-self::*
[@security][1]/@security"/>
  </fo:block>
  <xsl:apply-templates/>
</xsl:template>
```

7.6.2 Attribute Value Templates

In an attribute value that is interpreted as an **attribute value template**, such as an attribute of a literal result element, an expression can be used by surrounding the expression with curly braces (`{}`). The attribute value template is instantiated by replacing the expression together with surrounding curly braces by the result of evaluating the expression and converting the resulting object to a string as if by a call to the **string** function. Curly braces are not recognized in an attribute value in an XSLT stylesheet unless the attribute is specifically stated to be one that is interpreted as an attribute value template; in an element syntax summary, the value of such attributes is surrounded by curly braces.

> **NOTE:**Not all attributes are interpreted as attribute value templates. Attributes whose value is an expression or pattern, attributes of top-level elements and attributes that refer to named XSLT objects are not interpreted as attribute value templates. In addition, `xmlns` attributes are not interpreted as attribute value templates; it would not be conformant with the XML Namespaces

Recommendation to do this.

The following example creates an `img` result element from a `photograph` element in the source; the value of the `src` attribute of the `img` element is computed from the value of the `image-dir` variable and the string-value of the `href` child of the `photograph` element; the value of the `width` attribute of the `img` element is computed from the value of the `width` attribute of the `size` child of the `photograph` element:

```
<xsl:variable name="image-dir">/images</xsl:variable>

<xsl:template match="photograph">
<img src="{$image-dir}/{href}" width="{size/@width}"/
>
</xsl:template>
```

With this source

```
<photograph>
  <href>headquarters.jpg</href>
  <size width="300"/>
</photograph>
```

the result would be

```
<img src="/images/headquarters.jpg" width="300"/>
```

When an attribute value template is instantiated, a double left or right curly brace outside an expression will be replaced by a single curly brace. It is an error if a right curly brace occurs in an attribute value template outside an expression without being followed by a second right curly brace. A right curly brace inside a Literal in an expression is not recognized as terminating the expression.

Curly braces are *not* recognized recursively inside expressions. For example:

```
<a href="#{id({@ref})/title}">
```

is *not* allowed. Instead, use simply:

```
<a href="#{id(@ref)/title}">
```

7.7 Numbering

```
<!-- Category: instruction -->
<xsl:number
  level = "single" | "multiple" | "any"
  count = pattern
  from = pattern
  value = number-expression
  format = { string }
  lang = { nmtoken }
  letter-value = { "alphabetic" | "traditional" }
  grouping-separator = { char }
  grouping-size = { number } />
```

The `xsl:number` element is used to insert a formatted number into the result tree. The number to be inserted may be specified by an expression. The `value` attribute contains an expression. The expression is evaluated and the resulting object is converted to a number as if by a call to the **number** function. The number is rounded to an integer and then converted to a string using the attributes specified in [**7.7.1 Number to String Conversion Attributes**]; in this context, the value of each of these attributes is interpreted as an attribute value template. After conversion, the resulting string is inserted in the result tree. For example, the following example numbers a sorted list:

```
<xsl:template match="items">
  <xsl:for-each select="item">
    <xsl:sort select="."/>
    <p>
      <xsl:number value="position()" format="1. "/>
      <xsl:value-of select="."/>
    </p>
  </xsl:for-each>
</xsl:template>
```

If no `value` attribute is specified, then the `xsl:number` element inserts a number based on the position of the current node in the source tree. The following attributes control how the current node is to be numbered:

- The `level` attribute specifies what levels of the source tree should be considered; it has the values `single`, `multiple` or `any`. The default is `single`.

- The `count` attribute is a pattern that specifies what nodes should be counted at those levels. If `count` attribute is not specified, then it defaults to the pattern that matches any node with the same node type as the current node and, if the current node has an expanded-name, with the same expanded-name as the current node.

- The `from` attribute is a pattern that specifies where counting starts.

In addition, the attributes specified in [**7.7.1 Number to String Conversion Attributes**] are used for number to string conversion, as in the case when the `value` attribute is specified.

The `xsl:number` element first constructs a list of positive integers using the `level`, `count` and `from` attributes:

- When `level="single"`, it goes up to the first node in the ancestor-or-self axis that matches the `count` pattern, and constructs a list of length one containing one plus the number of preceding siblings of that ancestor that match the `count` pattern. If there is no such ancestor, it constructs an empty list. If the `from` attribute is specified, then the only ancestors that are searched are those that are descendants of the nearest ancestor that matches the `from` pattern. Preceding siblings has the same meaning here as with the `preceding-sibling` axis.

- When `level="multiple"`, it constructs a list of all ancestors of the current node in document order followed by the element itself; it then selects from the list those nodes that match the `count` pattern; it then maps each node in the list to one plus the number of preceding siblings of that node that match the `count` pattern. If the `from` attribute is specified, then the only ancestors that are searched are those that are descendants of the nearest ancestor that matches the `from` pattern. Preceding siblings has the same meaning here as with the `preceding-sibling` axis.

- When `level="any"`, it constructs a list of length one containing the number of nodes that match the `count` pattern and belong to the set containing the current node and all nodes at any level of the document that are before the current node in document order, excluding any namespace and attribute nodes (in other words the union of the members of the `preceding` and `ancestor-or-self` axes). If the `from` attribute is specified, then only nodes after the first node before the current node that match the `from` pattern are considered.

The list of numbers is then converted into a string using the attributes specified in [**7.7.1 Number to String Conversion Attributes**]; in this context, the value of each of these attributes is interpreted as an attribute value template. After conversion, the resulting string is inserted in the result tree.

The following would number the items in an ordered list:

```
<xsl:template match="ol/item">
  <fo:block>
    <xsl:number/><xsl:text>. </xsl:text><xsl:apply-
templates/>
  </fo:block>
<xsl:template>
```

The following two rules would number `title` elements. This is intended for a document that contains a sequence of chapters followed by a sequence of appendices, where both chapters and appendices contain sections, which in turn contain subsections. Chapters are numbered 1, 2, 3; appendices are numbered A, B, C; sections in chapters are numbered 1.1, 1.2, 1.3; sections in appendices are numbered A.1, A.2, A.3.

```
<xsl:template match="title">
  <fo:block>
     <xsl:number level="multiple"
                 count="chapter|section|subsection"
                 format="1.1 "/>
     <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="appendix//title" priority="1">
  <fo:block>
     <xsl:number level="multiple"
                 count="appendix|section|subsection"
                 format="A.1 "/>
     <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

The following example numbers notes sequentially within a chapter:

```
<xsl:template match="note">
  <fo:block>
     <xsl:number level="any" from="chapter"
format="(1) "/>
     <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

The following example would number `H4` elements in HTML with a three-part label:

```
<xsl:template match="H4">
 <fo:block>
   <xsl:number level="any" from="H1" count="H2"/>
   <xsl:text>.</xsl:text>
   <xsl:number level="any" from="H2" count="H3"/>
   <xsl:text>.</xsl:text>
   <xsl:number level="any" from="H3" count="H4"/>
   <xsl:text> </xsl:text>
   <xsl:apply-templates/>
 </fo:block>
</xsl:template>
```

7.7.1 Number to String Conversion Attributes

The following attributes are used to control conversion of a list of numbers into a string. The numbers are integers greater than 0. The attributes are all optional.

The main attribute is `format`. The default value for the `format` attribute is `1`. The `format` attribute is split into a sequence of tokens where each token is a maximal sequence of alphanumeric characters or a maximal sequence of non-alphanumeric characters. Alphanumeric means any character that has a Unicode category of Nd, Nl, No, Lu, Ll, Lt, Lm or Lo. The alphanumeric tokens (format tokens) specify the format to be used for each number in the list. If the first token is a non-alphanumeric token, then the constructed string will start with that token; if the last token is non-alphanumeric token, then the constructed string will end with that token. Non-alphanumeric tokens that occur between two format tokens are separator tokens that are used to join numbers in the list. The *n*th format token will be used to format the *n*th number in the list. If there are more numbers than format tokens, then the last format token will be used to format remaining numbers. If there are no format tokens, then a format token of `1` is used to format all numbers. The format token specifies the string to be used to represent the number 1. Each number after the first will be separated from the preceding number by the separator token preceding the format token used to format that number, or, if there are no separator tokens, then by `.` (a period character).

Format tokens are a superset of the allowed values for the `type` attribute for the `OL` element in HTML 4.0 and are interpreted as follows:

- Any token where the last character has a decimal digit value of 1 (as specified in the Unicode character property database), and the Unicode value of preceding characters is one less than the Unicode value of the last character generates a decimal representation of the number where each number is at least as long as the format token. Thus, a format

token 1 generates the sequence 1 2 ... 10 11 12 ..., and a format token 01 generates the sequence 01 02 ... 09 10 11 12 ... 99 100 101.

- A format token A generates the sequence A B C ... Z AA AB AC....
- A format token a generates the sequence a b c ... z aa ab ac....
- A format token i generates the sequence i ii iii iv v vi vii viii ix x ....
- A format token I generates the sequence I II III IV V VI VII VIII IX X ....
- Any other format token indicates a numbering sequence that starts with that token. If an implementation does not support a numbering sequence that starts with that token, it must use a format token of 1.

When numbering with an alphabetic sequence, the lang attribute specifies which language's alphabet is to be used; it has the same range of values as xml:lang [XML]; if no lang value is specified, the language should be determined from the system environment. Implementers should document for which languages they support numbering.

NOTE: Implementers should not make any assumptions about how numbering works in particular languages and should properly research the languages that they wish to support. The numbering conventions of many languages are very different from English.

The letter-value attribute disambiguates between numbering sequences that use letters. In many languages there are two commonly used numbering sequences that use letters. One numbering sequence assigns numeric values to letters in alphabetic sequence, and the other assigns numeric values to each letter in some other manner traditional in that language. In English, these would correspond to the numbering sequences specified by the format tokens a and i. In some languages, the first member of each sequence is the same, and so the format token alone would be ambiguous. A value of alphabetic specifies the alphabetic sequence; a value of traditional specifies the other sequence. If the letter-value attribute is not specified, then it is implementation-dependent how any ambiguity is resolved.

NOTE: It is possible for two conforming XSLT processors not to

convert a number to exactly the same string. Some XSLT processors may not support some languages. Furthermore, there may be variations possible in the way conversions are performed for any particular language that are not specifiable by the attributes on `xsl:number`. Future versions of XSLT may provide additional attributes to provide control over these variations. Implementations may also use implementation-specific namespaced attributes on `xsl:number` for this.

The `grouping-separator` attribute gives the separator used as a grouping (e.g. thousands) separator in decimal numbering sequences, and the optional `grouping-size` specifies the size (normally 3) of the grouping. For example, `grouping-separator=","` and `grouping-size="3"` would produce numbers of the form `1,000,000`. If only one of the `grouping-separator` and `grouping-size` attributes is specified, then it is ignored.

Here are some examples of conversion specifications:

- `format="&#x30A2;"` specifies Katakana numbering
- `format="&#x30A4;"` specifies Katakana numbering in the "iroha" order
- `format="&#x0E51;"` specifies numbering with Thai digits
- `format="&#x05D0;" letter-value="traditional"` specifies "traditional" Hebrew numbering
- `format="&#x10D0;" letter-value="traditional"` specifies Georgian numbering
- `format="&#x03B1;" letter-value="traditional"` specifies "classical" Greek numbering
- `format="&#x0430;" letter-value="traditional"` specifies Old Slavic numbering

8 Repetition

```
<!-- Category: instruction -->
<xsl:for-each
  select = node-set-expression>
  <!-- Content: (xsl:sort*, template) -->
</xsl:for-each>
```

When the result has a known regular structure, it is useful to be able to specify directly the template for selected nodes. The `xsl:for-each` instruction contains a template, which is instantiated for each node selected by the expression specified by the `select` attribute. The `select` attribute is required. The expression must evaluate to a node-set. The template is instantiated with the selected node as the current node, and with a list of all of the selected nodes as the current node list. The nodes are processed in document order, unless a sorting specification is present (see [**10 Sorting**]).

For example, given an XML document with this structure

```
<customers>
  <customer>
    <name>...</name>
    <order>...</order>
    <order>...</order>
  </customer>
  <customer>
    <name>...</name>
    <order>...</order>
    <order>...</order>
  </customer>
</customers>
```

the following would create an HTML document containing a table with a row for each `customer` element

```
<xsl:template match="/">
  <html>
    <head>
      <title>Customers</title>
    </head>
    <body>
      <table>
        <tbody>
          <xsl:for-each select="customers/customer">
            <tr>
              <th>
                <xsl:apply-templates select="name"/>
              </th>
              <xsl:for-each select="order">
                <td>
                  <xsl:apply-templates/>
                </td>
```

```
            </xsl:for-each>
          </tr>
        </xsl:for-each>
      </tbody>
    </table>
  </body>
 </html>
</xsl:template>
```

9 Conditional Processing

There are two instructions in XSLT that support conditional processing in a template: `xsl:if` and `xsl:choose`. The `xsl:if` instruction provides simple if-then conditionality; the `xsl:choose` instruction supports selection of one choice when there are several possibilities.

9.1 Conditional Processing with `xsl:if`

```
<!-- Category: instruction -->
<xsl:if
  test = boolean-expression>
  <!-- Content: template -->
</xsl:if>
```

The `xsl:if` element has a `test` attribute, which specifies an expression. The content is a template. The expression is evaluated and the resulting object is converted to a boolean as if by a call to the **boolean** function. If the result is true, then the content template is instantiated; otherwise, nothing is created. In the following example, the names in a group of names are formatted as a comma separated list:

```
<xsl:template match="namelist/name">
  <xsl:apply-templates/>
  <xsl:if test="not(position()=last())">, </xsl:if>
</xsl:template>
```

The following colors every other table row yellow:

```
<xsl:template match="item">
  <tr>
    <xsl:if test="position() mod 2 = 0">
       <xsl:attribute name="bgcolor">yellow</xsl:
attribute>
    </xsl:if>
```

```
      <xsl:apply-templates/>
    </tr>
  </xsl:template>
```

9.2 Conditional Processing with xsl:choose

```
<!-- Category: instruction -->
<xsl:choose>
  <!-- Content: (xsl:when+, xsl:otherwise?) -->
</xsl:choose>
```

```
<xsl:when
  test = boolean-expression>
  <!-- Content: template -->
</xsl:when>
```

```
<xsl:otherwise>
  <!-- Content: template -->
</xsl:otherwise>
```

The xsl:choose element selects one among a number of possible alternatives. It consists of a sequence of xsl:when elements followed by an optional xsl:otherwise element. Each xsl:when element has a single attribute, test, which specifies an expression. The content of the xsl:when and xsl:otherwise elements is a template. When an xsl:choose element is processed, each of the xsl:when elements is tested in turn, by evaluating the expression and converting the resulting object to a boolean as if by a call to the **boolean** function. The content of the first, and only the first, xsl:when element whose test is true is instantiated. If no xsl:when is true, the content of the xsl:otherwise element is instantiated. If no xsl:when element is true, and no xsl:otherwise element is present, nothing is created.

The following example enumerates items in an ordered list using arabic numerals, letters, or roman numerals depending on the depth to which the ordered lists are nested.

```
<xsl:template match="orderedlist/listitem">
  <fo:list-item indent-start='2pi'>
    <fo:list-item-label>
      <xsl:variable name="level"
                    select="count(ancestor::
orderedlist) mod 3"/>
      <xsl:choose>
```

```
            <xsl:when test='$level=1'>
              <xsl:number format="i"/>
            </xsl:when>
            <xsl:when test='$level=2'>
              <xsl:number format="a"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:number format="1"/>
            </xsl:otherwise>
          </xsl:choose>
          <xsl:text>. </xsl:text>
        </fo:list-item-label>
        <fo:list-item-body>
          <xsl:apply-templates/>
        </fo:list-item-body>
      </fo:list-item>
    </xsl:template>
```

10 Sorting

```
<xsl:sort
  select = string-expression
  lang = { nmtoken }
  data-type = { "text" | "number" | qname-but-not-
ncname }
  order = { "ascending" | "descending" }
  case-order = { "upper-first" | "lower-first" } />
```

Sorting is specified by adding `xsl:sort` elements as children of an `xsl:apply-templates` or `xsl:for-each` element. The first `xsl:sort` child specifies the primary sort key, the second `xsl:sort` child specifies the secondary sort key and so on. When an `xsl:apply-templates` or `xsl:for-each` element has one or more `xsl:sort` children, then instead of processing the selected nodes in document order, it sorts the nodes according to the specified sort keys and then processes them in sorted order. When used in `xsl:for-each`, `xsl:sort` elements must occur first. When a template is instantiated by `xsl:apply-templates` and `xsl:for-each`, the current node list list consists of the complete list of nodes being processed in sorted order.

`xsl:sort` has a `select` attribute whose value is an expression. For each node to be processed, the expression is evaluated with that node as the current node and with the complete list of nodes being processed in unsorted order as the current node list. The resulting object is converted to a string as if by a call to the **string** function; this string is used as the sort key for that node.

The default value of the `select` attribute is `.`, which will cause the string-value of the current node to be used as the sort key.

This string serves as a sort key for the node. The following optional attributes on `xsl:sort` control how the list of sort keys are sorted; the values of all of these attributes are interpreted as attribute value templates.

- `order` specifies whether the strings should be sorted in ascending or descending order; `ascending` specifies ascending order; `descending` specifies descending order; the default is `ascending`

- `lang` specifies the language of the sort keys; it has the same range of values as `xml:lang` [XML]; if no `lang` value is specified, the language should be determined from the system environment

- `data-type` specifies the data type of the strings; the following values are allowed:

  - `text` specifies that the sort keys should be sorted lexicographically in the culturally correct manner for the language specified by `lang`

  - `number` specifies that the sort keys should be converted to numbers and then sorted according to the numeric value; the sort key is converted to a number as if by a call to the **number** function; the `lang` attribute is ignored

  - a QName with a prefix is expanded into an expanded-name as described in [**2.4 Qualified Names**]; the expanded-name identifies the data-type; the behavior in this case is not specified by this document

  The default value is `text`.

  NOTE: The XSL Working Group plans that future versions of XSLT will leverage XML Schemas to define further values for this attribute.

- `case-order` has the value `upper-first` or `lower-first`; this applies when `data-type="text"`, and specifies that upper-case letters should sort before lower-case letters or vice-versa respectively. For example, if `lang="en"`, then `A a B b` are sorted with `case-order="upper-first"` and `a A b B` are sorted with `case-order="lower-first"`. The default value is language dependent.

NOTE: It is possible for two conforming XSLT processors not to sort exactly the same. Some XSLT processors may not support some languages. Furthermore, there may be variations possible in the sorting of any particular language that are not specified by the attributes on `xsl:sort`, for example, whether Hiragana or Katakana is sorted first in Japanese. Future versions of XSLT may provide additional attributes to provide control over these variations. Implementations may also use implementation-specific namespaced attributes on `xsl:sort` for this.

NOTE: It is recommended that implementers consult [UNICODE TR10] for information on internationalized sorting.

The sort must be stable: in the sorted list of nodes, any sub list that has sort keys that all compare equal must be in document order.

For example, suppose an employee database has the form

```
<employees>
  <employee>
    <name>
      <given>James</given>
      <family>Clark</family>
    </name>
    ...
  </employee>
</employees>
```

Then a list of employees sorted by name could be generated using:

```
<xsl:template match="employees">
  <ul>
    <xsl:apply-templates select="employee">
      <xsl:sort select="name/family"/>
      <xsl:sort select="name/given"/>
    </xsl:apply-templates>
  </ul>
</xsl:template>

<xsl:template match="employee">
  <li>
    <xsl:value-of select="name/given"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="name/family"/>
```

```
      </li>
    </xsl:template>
```

11 Variables and Parameters

```
<!-- Category: top-level-element -->
<!-- Category: instruction -->
<xsl:variable
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:variable>
```

```
<!-- Category: top-level-element -->
<xsl:param
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:param>
```

A variable is a name that may be bound to a value. The value to which a variable is bound (the **value** of the variable) can be an object of any of the types that can be returned by expressions. There are two elements that can be used to bind variables: `xsl:variable` and `xsl:param`. The difference is that the value specified on the `xsl:param` variable is only a default value for the binding; when the template or stylesheet within which the `xsl:param` element occurs is invoked, parameters may be passed that are used in place of the default values.

Both `xsl:variable` and `xsl:param` have a required `name` attribute, which specifies the name of the variable. The value of the `name` attribute is a QName, which is expanded as described in [**2.4 Qualified Names**].

For any use of these variable-binding elements, there is a region of the stylesheet tree within which the binding is visible; within this region, any binding of the variable that was visible on the variable-binding element itself is hidden. Thus, only the innermost binding of a variable is visible. The set of variable bindings in scope for an expression consists of those bindings that are visible at the point in the stylesheet where the expression occurs.

11.1 Result Tree Fragments

Variables introduce an additional data-type into the expression language. This additional data type is called **result tree fragment**. A variable may be bound

to a result tree fragment instead of one of the four basic XPath data-types (string, number, boolean, node-set). A result tree fragment represents a fragment of the result tree. A result tree fragment is treated equivalently to a node-set that contains just a single root node. However, the operations permitted on a result tree fragment are a subset of those permitted on a node-set. An operation is permitted on a result tree fragment only if that operation would be permitted on a string (the operation on the string may involve first converting the string to a number or boolean). In particular, it is not permitted to use the `/`, `//`, and `[]` operators on result tree fragments. When a permitted operation is performed on a result tree fragment, it is performed exactly as it would be on the equivalent node-set.

When a result tree fragment is copied into the result tree (see [**11.3 Using Values of Variables and Parameters with `xsl:copy-of`**]), then all the nodes that are children of the root node in the equivalent node-set are added in sequence to the result tree.

Expressions can only return values of type result tree fragment by referencing variables of type result tree fragment or calling extension functions that return a result tree fragment or getting a system property whose value is a result tree fragment.

11.2 Values of Variables and Parameters

A variable-binding element can specify the value of the variable in three alternative ways.

- If the variable-binding element has a `select` attribute, then the value of the attribute must be an expression and the value of the variable is the object that results from evaluating the expression. In this case, the content must be empty.

- If the variable-binding element does not have a `select` attribute and has non-empty content (i.e. the variable-binding element has one or more child nodes), then the content of the variable-binding element specifies the value. The content of the variable-binding element is a template, which is instantiated to give the value of the variable. The value is a result tree fragment equivalent to a node-set containing just a single root node having as children the sequence of nodes produced by instantiating the template. The base URI of the nodes in the result tree fragment is the base URI of the variable-binding element.

  It is an error if a member of the sequence of nodes created by instantiating the template is an attribute node or a namespace node, since a root node cannot have an attribute node or a namespace node

as a child. An XSLT processor may signal the error; if it does not signal the error, it must recover by not adding the attribute node or namespace node.

- If the variable-binding element has empty content and does not have a `select` attribute, then the value of the variable is an empty string. Thus

```
<xsl:variable name="x"/>
```

is equivalent to

```
<xsl:variable name="x" select="''"/>
```

NOTE: When a variable is used to select nodes by position, be careful not to do:

```
<xsl:variable name="n">2</xsl:variable>
...
<xsl:value-of select="item[$n]"/>
```

This will output the value of the first item element, because the variable n will be bound to a result tree fragment, not a number. Instead, do either

```
<xsl:variable name="n" select="2"/>
...
<xsl:value-of select="item[$n]"/>
```

or

```
<xsl:variable name="n">2</xsl:variable>
...
<xsl:value-of select="item[position()=$n]"/>
```

NOTE: One convenient way to specify the empty node-set as the default value of a parameter is:

```
<xsl:param name="x" select="/.."/>
```

11.3 Using Values of Variables and Parameters with `xsl:copy-of`

```
<!-- Category: instruction -->
<xsl:copy-of
```

```
select = expression />
```

The `xsl:copy-of` element can be used to insert a result tree fragment into the result tree, without first converting it to a string as `xsl:value-of` does (see [**7.6.1 Generating Text with `xsl:value-of`**]). The required `select` attribute contains an expression. When the result of evaluating the expression is a result tree fragment, the complete fragment is copied into the result tree. When the result is a node-set, all the nodes in the set are copied in document order into the result tree; copying an element node copies the attribute nodes, namespace nodes and children of the element node as well as the element node itself; a root node is copied by copying its children. When the result is neither a node-set nor a result tree fragment, the result is converted to a string and then inserted into the result tree, as with `xsl:value-of`.

11.4 Top-level Variables and Parameters

Both `xsl:variable` and `xsl:param` are allowed as top-level elements. A top-level variable-binding element declares a global variable that is visible everywhere. A top-level `xsl:param` element declares a parameter to the stylesheet; XSLT does not define the mechanism by which parameters are passed to the stylesheet. It is an error if a stylesheet contains more than one binding of a top-level variable with the same name and same import precedence. At the top-level, the expression or template specifying the variable value is evaluated with the same context as that used to process the root node of the source document: the current node is the root node of the source document and the current node list is a list containing just the root node of the source document. If the template or expression specifying the value of a global variable *x* references a global variable *y*, then the value for *y* must be computed before the value of *x*. It is an error if it is impossible to do this for all global variable definitions; in other words, it is an error if the definitions are circular.

This example declares a global variable `para-font-size`, which it references in an attribute value template.

```
<xsl:variable name="para-font-size">12pt</xsl:
variable>

<xsl:template match="para">
 <fo:block font-size="{$para-font-size}">
   <xsl:apply-templates/>
 </fo:block>
</xsl:template>
```

11.5 Variables and Parameters within Templates

As well as being allowed at the top-level, both `xsl:variable` and `xsl:param` are also allowed in templates. `xsl:variable` is allowed anywhere within a template that an instruction is allowed. In this case, the binding is visible for all following siblings and their descendants. Note that the binding is not visible for the `xsl:variable` element itself. `xsl:param` is allowed as a child at the beginning of an `xsl:template` element. In this context, the binding is visible for all following siblings and their descendants. Note that the binding is not visible for the `xsl:param` element itself.

A binding **shadows** another binding if the binding occurs at a point where the other binding is visible, and the bindings have the same name. It is an error if a binding established by an `xsl:variable` or `xsl:param` element within a template shadows another binding established by an `xsl:variable` or `xsl:param` element also within the template. It is not an error if a binding established by an `xsl:variable` or `xsl:param` element in a template shadows another binding established by an `xsl:variable` or `xsl:param` top-level element. Thus, the following is an error:

```
<xsl:template name="foo">
<xsl:param name="x" select="1"/>
<xsl:variable name="x" select="2"/>
</xsl:template>
```

However, the following is allowed:

```
<xsl:param name="x" select="1"/>
<xsl:template name="foo">
<xsl:variable name="x" select="2"/>
</xsl:template>
```

NOTE: The nearest equivalent in Java to an `xsl:variable` element in a template is a final local variable declaration with an initializer. For example,

```
<xsl:variable name="x" select="'value'"/>
```

has similar semantics to

```
final Object x = "value";
```

XSLT does not provide an equivalent to the Java assignment operator

```
x = "value";
```

because this would make it harder to create an implementation that processes a document other than in a batch-like way, starting at the beginning and continuing through to the end.

11.6 Passing Parameters to Templates

```
<xsl:with-param
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:with-param>
```

Parameters are passed to templates using the `xsl:with-param` element. The required `name` attribute specifies the name of the parameter (the variable the value of whose binding is to be replaced). The value of the `name` attribute is a QName, which is expanded as described in [**2.4 Qualified Names**]. `xsl:with-param` is allowed within both `xsl:call-template` and `xsl:apply-templates`. The value of the parameter is specified in the same way as for `xsl:variable` and `xsl:param`. The current node and current node list used for computing the value specified by `xsl:with-param` element is the same as that used for the `xsl:apply-templates` or `xsl:call-template` element within which it occurs. It is not an error to pass a parameter *x* to a template that does not have an `xsl:param` element for *x*; the parameter is simply ignored.

This example defines a named template for a `numbered-block` with an argument to control the format of the number.

```
<xsl:template name="numbered-block">
  <xsl:param name="format">1. </xsl:param>
  <fo:block>
    <xsl:number format="{$format}"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="ol//ol/li">
  <xsl:call-template name="numbered-block">
    <xsl:with-param name="format">a. </xsl:with-
param>
  </xsl:call-template>
</xsl:template>
```

12 Additional Functions

This section describes XSLT-specific additions to the core XPath function library. Some of these additional functions also make use of information specified by top-level elements in the stylesheet; this section also describes these elements.

12.1 Multiple Source Documents

Function: *node-set* **document**(*object*, *node-set*?)

The **document** function allows access to XML documents other than the main source document.

When the **document** function has exactly one argument and the argument is a node-set, then the result is the union, for each node in the argument node-set, of the result of calling the **document** function with the first argument being the string-value of the node, and the second argument being a node-set with the node as its only member. When the **document** function has two arguments and the first argument is a node-set, then the result is the union, for each node in the argument node-set, of the result of calling the **document** function with the first argument being the string-value of the node, and with the second argument being the second argument passed to the **document** function.

When the first argument to the **document** function is not a node-set, the first argument is converted to a string as if by a call to the **string** function. This string is treated as a URI reference; the resource identified by the URI is retrieved. The data resulting from the retrieval action is parsed as an XML document and a tree is constructed in accordance with the data model (see [**3 Data Model**]). If there is an error retrieving the resource, then the XSLT processor may signal an error; if it does not signal an error, it must recover by returning an empty node-set. One possible kind of retrieval error is that the XSLT processor does not support the URI scheme used by the URI. An XSLT processor is not required to support any particular URI schemes. The documentation for an XSLT processor should specify which URI schemes the XSLT processor supports.

If the URI reference does not contain a fragment identifier, then a node-set containing just the root node of the document is returned. If the URI reference does contain a fragment identifier, the function returns a node-set containing the nodes in the tree identified by the fragment identifier of the URI reference.

The semantics of the fragment identifier is dependent on the media type of the result of retrieving the URI. If there is an error in processing the fragment identifier, the XSLT processor may signal the error; if it does not signal the error, it must recover by returning an empty node-set. Possible errors include:

- The fragment identifier identifies something that cannot be represented by an XSLT node-set (such as a range of characters within a text node).
- The XSLT processor does not support fragment identifiers for the media-type of the retrieval result. An XSLT processor is not required to support any particular media types. The documentation for an XSLT processor should specify for which media types the XSLT processor supports fragment identifiers.

The data resulting from the retrieval action is parsed as an XML document regardless of the media type of the retrieval result; if the top-level media type is `text`, then it is parsed in the same way as if the media type were `text/xml`; otherwise, it is parsed in the same way as if the media type were `application/xml`.

> **NOTE:**Since there is no top-level `xml` media type, data with a media type other than `text/xml` or `application/xml` may in fact be XML.

The URI reference may be relative. The base URI (see **[3.2 Base URI]**) of the node in the second argument node-set that is first in document order is used as the base URI for resolving the relative URI into an absolute URI. If the second argument is omitted, then it defaults to the node in the stylesheet that contains the expression that includes the call to the **document** function. Note that a zero-length URI reference is a reference to the document relative to which the URI reference is being resolved; thus `document("")` refers to the root node of the stylesheet; the tree representation of the stylesheet is exactly the same as if the XML document containing the stylesheet was the initial source document.

Two documents are treated as the same document if they are identified by the same URI. The URI used for the comparison is the absolute URI into which any relative URI was resolved and does not include any fragment identifier. One root node is treated as the same node as another root node if the two nodes are from the same document. Thus, the following expression will always be true:

```
generate-id(document("foo.xml"))=generate-id(document
("foo.xml"))
```

The **document** function gives rise to the possibility that a node-set may contain nodes from more than one document. With such a node-set, the relative document order of two nodes in the same document is the normal document order defined by XPath [XPath]. The relative document order of two nodes in different documents is determined by an implementation-dependent ordering of the documents containing the two nodes. There are no constraints on how the implementation orders documents other than that it must do so consistently: an implementation must always use the same order for the same set of documents.

12.2 Keys

Keys provide a way to work with documents that contain an implicit cross-reference structure. The `ID`, `IDREF` and `IDREFS` attribute types in XML provide a mechanism to allow XML documents to make their cross-reference explicit. XSLT supports this through the XPath **id** function. However, this mechanism has a number of limitations:

- ID attributes must be declared as such in the DTD. If an ID attribute is declared as an ID attribute only in the external DTD subset, then it will be recognized as an ID attribute only if the XML processor reads the external DTD subset. However, XML does not require XML processors to read the external DTD, and they may well choose not to do so, especially if the document is declared `standalone="yes"`.
- A document can contain only a single set of unique IDs. There cannot be separate independent sets of unique IDs.
- The ID of an element can only be specified in an attribute; it cannot be specified by the content of the element, or by a child element.
- An ID is constrained to be an XML name. For example, it cannot contain spaces.
- An element can have at most one ID.
- At most one element can have a particular ID.

Because of these limitations XML documents sometimes contain a cross-reference structure that is not explicitly declared by ID/IDREF/IDREFS attributes.

A key is a triple containing:

1. the node which has the key
2. the name of the key (an expanded-name)
3. the value of the key (a string)

A stylesheet declares a set of keys for each document using the `xsl:key` element. When this set of keys contains a member with node *x*, name *y* and value *z*, we say that node *x* has a key with name *y* and value *z*.

Thus, a key is a kind of generalized ID, which is not subject to the same limitations as an XML ID:

- Keys are declared in the stylesheet using `xsl:key` elements.
- A key has a name as well as a value; each key name may be thought of as distinguishing a separate, independent space of identifiers.
- The value of a named key for an element may be specified in any convenient place; for example, in an attribute, in a child element or in content. An XPath expression is used to specify where to find the value for a particular named key.
- The value of a key can be an arbitrary string; it is not constrained to be a name.
- There can be multiple keys in a document with the same node, same key name, but different key values.
- There can be multiple keys in a document with the same key name, same key value, but different nodes.

```
<!-- Category: top-level-element -->
<xsl:key
  name = qname
  match = pattern
  use = expression />
```

The `xsl:key` element is used to declare keys. The `name` attribute specifies the name of the key. The value of the `name` attribute is a QName, which is expanded as described in [**2.4 Qualified Names**]. The `match` attribute is a Pattern; an `xsl:key` element gives information about the keys of any node that matches the pattern specified in the match attribute. The `use` attribute is an expression specifying the values of the key; the expression is evaluated

once for each node that matches the pattern. If the result is a node-set, then for each node in the node-set, the node that matches the pattern has a key of the specified name whose value is the string-value of the node in the node-set; otherwise, the result is converted to a string, and the node that matches the pattern has a key of the specified name with value equal to that string. Thus, a node *x* has a key with name *y* and value *z* if and only if there is an `xsl:key` element such that:

- *x* matches the pattern specified in the `match` attribute of the `xsl:key` element;
- the value of the `name` attribute of the `xsl:key` element is equal to *y*; and
- when the expression specified in the `use` attribute of the `xsl:key` element is evaluated with *x* as the current node and with a node list containing just *x* as the current node list resulting in an object *u*, then either *z* is equal to the result of converting *u* to a string as if by a call to the **string** function, or *u* is a node-set and *z* is equal to the string-value of one or more of the nodes in *u*.

Note also that there may be more than one `xsl:key` element that matches a given node; all of the matching `xsl:key` elements are used, even if they do not have the same import precedence.

It is an error for the value of either the `use` attribute or the `match` attribute to contain a VariableReference.

Function: *node-set* **key**(*string*, *object*)

The **key** function does for keys what the **id** function does for IDs. The first argument specifies the name of the key. The value of the argument must be a QName, which is expanded as described in [**2.4 Qualified Names**]. When the second argument to the **key** function is of type node-set, then the result is the union of the result of applying the **key** function to the string value of each of the nodes in the argument node-set. When the second argument to **key** is of any other type, the argument is converted to a string as if by a call to the **string** function; it returns a node-set containing the nodes in the same document as the context node that have a value for the named key equal to this string.

For example, given a declaration

```
<xsl:key name="idkey" match="div" use="@id"/>
```

an expression `key("idkey",@ref)` will return the same node-set as `id(@ref)`, assuming that the only ID attribute declared in the XML source document is:

```
<!ATTLIST div id ID #IMPLIED>
```

and that the `ref` attribute of the current node contains no whitespace.

Suppose a document describing a function library uses a `prototype` element to define functions

```
<prototype name="key" return-type="node-set">
<arg type="string"/>
<arg type="object"/>
</prototype>
```

and a `function` element to refer to function names

```
<function>key</function>
```

Then the stylesheet could generate hyperlinks between the references and definitions as follows:

```
<xsl:key name="func" match="prototype" use="@name"/>

<xsl:template match="function">
<b>
  <a href="#{generate-id(key('func',.))}">
    <xsl:apply-templates/>
  </a>
</b>
</xsl:template>

<xsl:template match="prototype">
<p><a name="{generate-id()}">
<b>Function: </b>
...
</a></p>
</xsl:template>
```

The **key** can be used to retrieve a key from a document other than the document containing the context node. For example, suppose a document contains bibliographic references in the form `<bibref>XSLT</bibref>`, and there is a separate XML document `bib.xml` containing a bibliographic

database with entries in the form:

```
<entry name="XSLT">...</entry>
```

Then the stylesheet could use the following to transform the `bibref` elements:

```
<xsl:key name="bib" match="entry" use="@name"/>

<xsl:template match="bibref">
  <xsl:variable name="name" select="."/>
  <xsl:for-each select="document('bib.xml')">
    <xsl:apply-templates select="key('bib',$name)"/>
  </xsl:for-each>
</xsl:template>
```

12.3 Number Formatting

Function: *string* **format-number**(*number*, *string*, *string*?)

The **format-number** function converts its first argument to a string using the format pattern string specified by the second argument and the decimal-format named by the third argument, or the default decimal-format, if there is no third argument. The format pattern string is in the syntax specified by the JDK 1.1 DecimalFormat class. The format pattern string is in a localized notation: the decimal-format determines what characters have a special meaning in the pattern (with the exception of the quote character, which is not localized). The format pattern must not contain the currency sign (#x00A4); support for this feature was added after the initial release of JDK 1.1. The decimal-format name must be a QName, which is expanded as described in [**2.4 Qualified Names**]. It is an error if the stylesheet does not contain a declaration of the decimal-format with the specified expanded-name.

> **NOTE:**Implementations are not required to use the JDK 1.1 implementation, nor are implementations required to be implemented in Java.

> **NOTE:**Stylesheets can use other facilities in XPath to control rounding.

```
<!-- Category: top-level-element -->
<xsl:decimal-format
  name = qname
  decimal-separator = char
```

```
  grouping-separator = char
  infinity = string
  minus-sign = char
  NaN = string
  percent = char
  per-mille = char
  zero-digit = char
  digit = char
  pattern-separator = char />
```

The `xsl:decimal-format` element declares a decimal-format, which controls the interpretation of a format pattern used by the **format-number** function. If there is a `name` attribute, then the element declares a named decimal-format; otherwise, it declares the default decimal-format. The value of the `name` attribute is a QName, which is expanded as described in [**2.4 Qualified Names**]. It is an error to declare either the default decimal-format or a decimal-format with a given name more than once (even with different import precedence), unless it is declared every time with the same value for all attributes (taking into account any default values).

The other attributes on `xsl:decimal-format` correspond to the methods on the JDK 1.1 DecimalFormatSymbols class. For each `get`/`set` method pair there is an attribute defined for the `xsl:decimal-format` element.

The following attributes both control the interpretation of characters in the format pattern and specify characters that may appear in the result of formatting the number:

- `decimal-separator` specifies the character used for the decimal sign; the default value is the period character (`.`)
- `grouping-separator` specifies the character used as a grouping (e. g. thousands) separator; the default value is the comma character (`,`)
- `percent` specifies the character used as a percent sign; the default value is the percent character (%)
- `per-mille` specifies the character used as a per mille sign; the default value is the Unicode per-mille character (#x2030)
- `zero-digit` specifies the character used as the digit zero; the default value is the digit zero (`0`)

The following attributes control the interpretation of characters in the format

pattern:

- `digit` specifies the character used for a digit in the format pattern; the default value is the number sign character (#)
- `pattern-separator` specifies the character used to separate positive and negative sub patterns in a pattern; the default value is the semi-colon character (`;`)

The following attributes specify characters or strings that may appear in the result of formatting the number:

- `infinity` specifies the string used to represent infinity; the default value is the string `Infinity`
- `NaN` specifies the string used to represent the NaN value; the default value is the string `NaN`
- `minus-sign` specifies the character used as the default minus sign; the default value is the hyphen-minus character (`-`, #x2D)

12.4 Miscellaneous Additional Functions

Function: *node-set* **current**()

The **current** function returns a node-set that has the current node as its only member. For an outermost expression (an expression not occurring within another expression), the current node is always the same as the context node. Thus,

```
<xsl:value-of select="current()"/>
```

means the same as

```
<xsl:value-of select="."/>
```

However, within square brackets the current node is usually different from the context node. For example,

```
<xsl:apply-templates select="//glossary/item
[@name=current()/@ref]"/>
```

will process all `item` elements that have a `glossary` parent element and that have a `name` attribute with value equal to the value of the current node's

`ref` attribute. This is different from

```
<xsl:apply-templates select="//glossary/item[@name=./
@ref]"/>
```

which means the same as

```
<xsl:apply-templates select="//glossary/item
[@name=@ref]"/>
```

and so would process all `item` elements that have a `glossary` parent element and that have a `name` attribute and a `ref` attribute with the same value.

It is an error to use the **current** function in a pattern.

Function: *string* **unparsed-entity-uri**(*string*)

The **unparsed-entity-uri** returns the URI of the unparsed entity with the specified name in the same document as the context node (see [**3.3 Unparsed Entities**]). It returns the empty string if there is no such entity.

Function: *string* **generate-id**(*node-set*?)

The **generate-id** function returns a string that uniquely identifies the node in the argument node-set that is first in document order. The unique identifier must consist of ASCII alphanumeric characters and must start with an alphabetic character. Thus, the string is syntactically an XML name. An implementation is free to generate an identifier in any convenient way provided that it always generates the same identifier for the same node and that different identifiers are always generated from different nodes. An implementation is under no obligation to generate the same identifiers each time a document is transformed. There is no guarantee that a generated unique identifier will be distinct from any unique IDs specified in the source document. If the argument node-set is empty, the empty string is returned. If the argument is omitted, it defaults to the context node.

Function: *object* **system-property**(*string*)

The argument must evaluate to a string that is a QName. The QName is expanded into a name using the namespace declarations in scope for the expression. The **system-property** function returns an object representing the value of the system property identified by the name. If there is no such system property, the empty string should be returned.

Implementations must provide the following system properties, which are all in the XSLT namespace:

- `xsl:version`, a number giving the version of XSLT implemented by the processor; for XSLT processors implementing the version of XSLT specified by this document, this is the number 1.0
- `xsl:vendor`, a string identifying the vendor of the XSLT processor
- `xsl:vendor-url`, a string containing a URL identifying the vendor of the XSLT processor; typically this is the host page (home page) of the vendor's Web site.

13 Messages

```
<!-- Category: instruction -->
<xsl:message
  terminate = "yes" | "no">
  <!-- Content: template -->
</xsl:message>
```

The `xsl:message` instruction sends a message in a way that is dependent on the XSLT processor. The content of the `xsl:message` instruction is a template. The `xsl:message` is instantiated by instantiating the content to create an XML fragment. This XML fragment is the content of the message.

> **NOTE:**An XSLT processor might implement `xsl:message` by popping up an alert box or by writing to a log file.

If the `terminate` attribute has the value `yes`, then the XSLT processor should terminate processing after sending the message. The default value is `no`.

One convenient way to do localization is to put the localized information (message text, etc.) in an XML document, which becomes an additional input file to the stylesheet. For example, suppose messages for a language *L* are stored in an XML file `resources/L.xml` in the form:

```
<messages>
  <message name="problem">A problem was detected.</
message>
  <message name="error">An error was detected.</
message>
</messages>
```

Then a stylesheet could use the following approach to localize messages:

```
<xsl:param name="lang" select="en"/>
<xsl:variable name="messages"
  select="document(concat('resources/', $lang, '.
xml'))/messages"/>

<xsl:template name="localized-message">
  <xsl:param name="name"/>
  <xsl:message>
    <xsl:value-of select="$messages/message[@name=
$name]"/>
  </xsl:message>
</xsl:template>

<xsl:template name="problem">
  <xsl:call-template name="localized-message"/>
    <xsl:with-param name="name">problem</xsl:with-
param>
  </xsl:call-template>
</xsl:template>
```

14 Extensions

XSLT allows two kinds of extension, extension elements and extension functions.

This version of XSLT does not provide a mechanism for defining implementations of extensions. Therefore, an XSLT stylesheet that must be portable between XSLT implementations cannot rely on particular extensions being available. XSLT provides mechanisms that allow an XSLT stylesheet to determine whether the XSLT processor by which it is being processed has implementations of particular extensions available, and to specify what should happen if those extensions are not available. If an XSLT stylesheet is careful to make use of these mechanisms, it is possible for it to take advantage of extensions and still work with any XSLT implementation.

14.1 Extension Elements

The element extension mechanism allows namespaces to be designated as **extension namespace**s. When a namespace is designated as an extension namespace and an element with a name from that namespace occurs in a template, then the element is treated as an instruction rather than as a literal result element. The namespace determines the semantics of the instruction.

> **NOTE:**Since an element that is a child of an `xsl:stylesheet` element is not occurring *in a template*, non-XSLT top-level elements are not extension elements as defined here, and nothing in this section applies to them.

A namespace is designated as an extension namespace by using an `extension-element-prefixes` attribute on an `xsl:stylesheet` element or an `xsl:extension-element-prefixes` attribute on a literal result element or extension element. The value of both these attributes is a whitespace-separated list of namespace prefixes. The namespace bound to each of the prefixes is designated as an extension namespace. It is an error if there is no namespace bound to the prefix on the element bearing the `extension-element-prefixes` or `xsl:extension-element-prefixes` attribute. The default namespace (as declared by `xmlns`) may be designated as an extension namespace by including `#default` in the list of namespace prefixes. The designation of a namespace as an extension namespace is effective within the subtree of the stylesheet rooted at the element bearing the `extension-element-prefixes` or `xsl:extension-element-prefixes` attribute; a subtree rooted at an `xsl:stylesheet` element does not include any stylesheets imported or included by children of that `xsl:stylesheet` element.

If the XSLT processor does not have an implementation of a particular extension element available, then the **element-available** function must return false for the name of the element. When such an extension element is instantiated, then the XSLT processor must perform fallback for the element as specified in [**15 Fallback**]. An XSLT processor must not signal an error merely because a template contains an extension element for which no implementation is available.

If the XSLT processor has an implementation of a particular extension element available, then the **element-available** function must return true for the name of the element.

14.2 Extension Functions

If a FunctionName in a FunctionCall expression is not an NCName (i.e. if it contains a colon), then it is treated as a call to an extension function. The FunctionName is expanded to a name using the namespace declarations from the evaluation context.

If the XSLT processor does not have an implementation of an extension function of a particular name available, then the **function-available** function must return false for that name. If such an extension function occurs in an

expression and the extension function is actually called, the XSLT processor must signal an error. An XSLT processor must not signal an error merely because an expression contains an extension function for which no implementation is available.

If the XSLT processor has an implementation of an extension function of a particular name available, then the **function-available** function must return true for that name. If such an extension is called, then the XSLT processor must call the implementation passing it the function call arguments; the result returned by the implementation is returned as the result of the function call.

15 Fallback

```
<!-- Category: instruction -->
<xsl:fallback>
  <!-- Content: template -->
</xsl:fallback>
```

Normally, instantiating an `xsl:fallback` element does nothing. However, when an XSLT processor performs fallback for an instruction element, if the instruction element has one or more `xsl:fallback` children, then the content of each of the `xsl:fallback` children must be instantiated in sequence; otherwise, an error must be signaled. The content of an `xsl:fallback` element is a template.

The following functions can be used with the `xsl:choose` and `xsl:if` instructions to explicitly control how a stylesheet should behave if particular elements or functions are not available.

Function: *boolean* **element-available**(*string*)

The argument must evaluate to a string that is a QName. The QName is expanded into an expanded-name using the namespace declarations in scope for the expression. The **element-available** function returns true if and only if the expanded-name is the name of an instruction. If the expanded-name has a namespace URI equal to the XSLT namespace URI, then it refers to an element defined by XSLT. Otherwise, it refers to an extension element. If the expanded-name has a null namespace URI, the **element-available** function will return false.

Function: *boolean* **function-available**(*string*)

The argument must evaluate to a string that is a QName. The QName is expanded into an expanded-name using the namespace declarations in

scope for the expression. The **function-available** function returns true if and only if the expanded-name is the name of a function in the function library. If the expanded-name has a non-null namespace URI, then it refers to an extension function; otherwise, it refers to a function defined by XPath or XSLT.

16 Output

```
<!-- Category: top-level-element -->
<xsl:output
  method = "xml" | "html" | "text" | qname-but-not-
ncname
  version = nmtoken
  encoding = string
  omit-xml-declaration = "yes" | "no"
  standalone = "yes" | "no"
  doctype-public = string
  doctype-system = string
  cdata-section-elements = qnames
  indent = "yes" | "no"
  media-type = string />
```

An XSLT processor may output the result tree as a sequence of bytes, although it is not required to be able to do so (see [**17 Conformance**]). The `xsl:output` element allows stylesheet authors to specify how they wish the result tree to be output. If an XSLT processor outputs the result tree, it should do so as specified by the `xsl:output` element; however, it is not required to do so.

The `xsl:output` element is only allowed as a top-level element.

The `method` attribute on `xsl:output` identifies the overall method that should be used for outputting the result tree. The value must be a QName. If the QName does not have a prefix, then it identifies a method specified in this document and must be one of `xml`, `html` or `text`. If the QName has a prefix, then the QName is expanded into an expanded-name as described in [**2.4 Qualified Names**]; the expanded-name identifies the output method; the behavior in this case is not specified by this document.

The default for the `method` attribute is chosen as follows. If

- the root node of the result tree has an element child,
- the expanded-name of the first element child of the root node (i.e. the

document element) of the result tree has local part `html` (in any combination of upper and lower case) and a null namespace URI, and

- any text nodes preceding the first element child of the root node of the result tree contain only whitespace characters,

then the default output method is `html`; otherwise, the default output method is `xml`. The default output method should be used if there are no `xsl:output` elements or if none of the `xsl:output` elements specifies a value for the `method` attribute.

The other attributes on `xsl:output` provide parameters for the output method. The following attributes are allowed:

- `version` specifies the version of the output method
- `indent` specifies whether the XSLT processor may add additional whitespace when outputting the result tree; the value must be `yes` or `no`
- `encoding` specifies the preferred character encoding that the XSLT processor should use to encode sequences of characters as sequences of bytes; the value of the attribute should be treated case-insensitively; the value must contain only characters in the range #x21 to #x7E (i.e. printable ASCII characters); the value should either be a `charset` registered with the Internet Assigned Numbers Authority [IANA], [RFC2278] or start with `X-`
- `media-type` specifies the media type (MIME content type) of the data that results from outputting the result tree; the `charset` parameter should not be specified explicitly; instead, when the top-level media type is `text`, a `charset` parameter should be added according to the character encoding actually used by the output method
- `doctype-system` specifies the system identifier to be used in the document type declaration
- `doctype-public` specifies the public identifier to be used in the document type declaration
- `omit-xml-declaration` specifies whether the XSLT processor should output an XML declaration; the value must be `yes` or `no`
- `standalone` specifies whether the XSLT processor should output a standalone document declaration; the value must be `yes` or `no`

- `cdata-section-elements` specifies a list of the names of elements whose text node children should be output using CDATA sections

The detailed semantics of each attribute will be described separately for each output method for which it is applicable. If the semantics of an attribute are not described for an output method, then it is not applicable to that output method.

A stylesheet may contain multiple `xsl:output` elements and may include or import stylesheets that also contain `xsl:output` elements. All the `xsl:output` elements occurring in a stylesheet are merged into a single effective `xsl:output` element. For the `cdata-section-elements` attribute, the effective value is the union of the specified values. For other attributes, the effective value is the specified value with the highest import precedence. It is an error if there is more than one such value for an attribute. An XSLT processor may signal the error; if it does not signal the error, if should recover by using the value that occurs last in the stylesheet. The values of attributes are defaulted after the `xsl:output` elements have been merged; different output methods may have different default values for an attribute.

16.1 XML Output Method

The `xml` output method outputs the result tree as a well-formed XML external general parsed entity. If the root node of the result tree has a single element node child and no text node children, then the entity should also be a well-formed XML document entity. When the entity is referenced within a trivial XML document wrapper like this

```
<!DOCTYPE doc [
<!ENTITY e SYSTEM "entity-URI">
]>
<doc>&e;</doc>
```

where *`entity-URI`* is a URI for the entity, then the wrapper document as a whole should be a well-formed XML document conforming to the XML Namespaces Recommendation [XML Names]. In addition, the output should be such that if a new tree was constructed by parsing the wrapper as an XML document as specified in [**3 Data Model**], and then removing the document element, making its children instead be children of the root node, then the new tree would be the same as the result tree, with the following possible exceptions:

- The order of attributes in the two trees may be different.

- The new tree may contain namespace nodes that were not present in the result tree.

> **NOTE:**An XSLT processor may need to add namespace declarations in the course of outputting the result tree as XML.

If the XSLT processor generated a document type declaration because of the `doctype-system` attribute, then the above requirements apply to the entity with the generated document type declaration removed.

The `version` attribute specifies the version of XML to be used for outputting the result tree. If the XSLT processor does not support this version of XML, it should use a version of XML that it does support. The version output in the XML declaration (if an XML declaration is output) should correspond to the version of XML that the processor used for outputting the result tree. The value of the `version` attribute should match the VersionNum production of the XML Recommendation [XML]. The default value is `1.0`.

The `encoding` attribute specifies the preferred encoding to use for outputting the result tree. XSLT processors are required to respect values of `UTF-8` and `UTF-16`. For other values, if the XSLT processor does not support the specified encoding it may signal an error; if it does not signal an error it should use `UTF-8` or `UTF-16` instead. The XSLT processor must not use an encoding whose name does not match the EncName production of the XML Recommendation [XML]. If no `encoding` attribute is specified, then the XSLT processor should use either `UTF-8` or `UTF-16`. It is possible that the result tree will contain a character that cannot be represented in the encoding that the XSLT processor is using for output. In this case, if the character occurs in a context where XML recognizes character references (i.e. in the value of an attribute node or text node), then the character should be output as a character reference; otherwise (for example if the character occurs in the name of an element) the XSLT processor should signal an error.

If the `indent` attribute has the value `yes`, then the `xml` output method may output whitespace in addition to the whitespace in the result tree (possibly based on whitespace stripped from either the source document or the stylesheet) in order to indent the result nicely; if the `indent` attribute has the value `no`, it should not output any additional whitespace. The default value is `no`. The `xml` output method should use an algorithm to output additional whitespace that ensures that the result if whitespace were to be stripped from the output using the process described in **[3.4 Whitespace Stripping]** with the set of whitespace-preserving elements consisting of just `xsl:text` would be the same when additional whitespace is output as when additional whitespace is not output.

> **NOTE:**It is usually not safe to use `indent="yes"` with document types that include element types with mixed content.

The `cdata-section-elements` attribute contains a whitespace-separated list of QNames. Each QName is expanded into an expanded-name using the namespace declarations in effect on the `xsl:output` element in which the QName occurs; if there is a default namespace, it is used for QNames that do not have a prefix. The expansion is performed before the merging of multiple `xsl:output` elements into a single effective `xsl:output` element. If the expanded-name of the parent of a text node is a member of the list, then the text node should be output as a CDATA section. For example,

```
<xsl:output cdata-section-elements="example"/>
```

would cause a literal result element written in the stylesheet as

```
<example>&lt;foo></example>
```

or as

```
<example><![CDATA[<foo>]]></example>
```

to be output as

```
<example><![CDATA[<foo>]]></example>
```

If the text node contains the sequence of characters `]]>`, then the currently open CDATA section should be closed following the `]]` and a new CDATA section opened before the >. For example, a literal result element written in the stylesheet as

```
<example>]]&gt;</example>
```

would be output as

```
<example><![CDATA[]]]><![CDATA[>]]></example>
```

If the text node contains a character that is not representable in the character encoding being used to output the result tree, then the currently open CDATA section should be closed before the character, the character should be output using a character reference or entity reference, and a new CDATA section should be opened for any further characters in the text node.

CDATA sections should not be used except for text nodes that the `cdata-section-elements` attribute explicitly specifies should be output using CDATA sections.

The `xml` output method should output an XML declaration unless the `omit-xml-declaration` attribute has the value `yes`. The XML declaration should include both version information and an encoding declaration. If the `standalone` attribute is specified, it should include a standalone document declaration with the same value as the value as the value of the `standalone` attribute. Otherwise, it should not include a standalone document declaration; this ensures that it is both a XML declaration (allowed at the beginning of a document entity) and a text declaration (allowed at the beginning of an external general parsed entity).

If the `doctype-system` attribute is specified, the `xml` output method should output a document type declaration immediately before the first element. The name following `<!DOCTYPE` should be the name of the first element. If `doctype-public` attribute is also specified, then the `xml` output method should output `PUBLIC` followed by the public identifier and then the system identifier; otherwise, it should output `SYSTEM` followed by the system identifier. The internal subset should be empty. The `doctype-public` attribute should be ignored unless the `doctype-system` attribute is specified.

The `media-type` attribute is applicable for the `xml` output method. The default value for the `media-type` attribute is `text/xml`.

16.2 HTML Output Method

The `html` output method outputs the result tree as HTML; for example,

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/
XSL/Transform">

<xsl:output method="html"/>

<xsl:template match="/">
  <html>
   <xsl:apply-templates/>
  </html>
</xsl:template>

...

</xsl:stylesheet>
```

The `version` attribute indicates the version of the HTML. The default value is `4.0`, which specifies that the result should be output as HTML conforming to the HTML 4.0 Recommendation [HTML].

The `html` output method should not output an element differently from the `xml` output method unless the expanded-name of the element has a null namespace URI; an element whose expanded-name has a non-null namespace URI should be output as XML. If the expanded-name of the element has a null namespace URI, but the local part of the expanded-name is not recognized as the name of an HTML element, the element should output in the same way as a non-empty, inline element such as `span`.

The `html` output method should not output an end-tag for empty elements. For HTML 4.0, the empty elements are `area`, `base`, `basefont`, `br`, `col`, `frame`, `hr`, `img`, `input`, `isindex`, `link`, `meta` and `param`. For example, an element written as `<br/>` or `<br></br>` in the stylesheet should be output as `<br>`.

The `html` output method should recognize the names of HTML elements regardless of case. For example, elements named `br`, `BR` or `Br` should all be recognized as the HTML `br` element and output without an end-tag.

The `html` output method should not perform escaping for the content of the `script` and `style` elements. For example, a literal result element written in the stylesheet as

```
<script>if (a &lt; b) foo()</script>
```

or

```
<script><![CDATA[if (a < b) foo()]]></script>
```

should be output as

```
<script>if (a < b) foo()</script>
```

The `html` output method should not escape < characters occurring in attribute values.

If the `indent` attribute has the value `yes`, then the `html` output method may add or remove whitespace as it outputs the result tree, so long as it does not change how an HTML user agent would render the output. The default value is `yes`.

The `html` output method should escape non-ASCII characters in URI attribute values using the method recommended in Section B.2.1 of the HTML 4.0 Recommendation.

The `html` output method may output a character using a character entity reference, if one is defined for it in the version of HTML that the output method is using.

The `html` output method should terminate processing instructions with > rather than ?>.

The `html` output method should output boolean attributes (that is attributes with only a single allowed value that is equal to the name of the attribute) in minimized form. For example, a start-tag written in the stylesheet as

```
<OPTION selected="selected">
```

should be output as

```
<OPTION selected>
```

The `html` output method should not escape a `&` character occurring in an attribute value immediately followed by a `{` character (see Section B.7.1 of the HTML 4.0 Recommendation). For example, a start-tag written in the stylesheet as

```
<BODY bgcolor='&amp;{{randomrbg}};'>
```

should be output as

```
<BODY bgcolor='&{randomrbg};'>
```

The `encoding` attribute specifies the preferred encoding to be used. If there is a `HEAD` element, then the `html` output method should add a `META` element immediately after the start-tag of the `HEAD` element specifying the character encoding actually used. For example,

```
<HEAD>
<META http-equiv="Content-Type" content="text/html;
charset=EUC-JP">
...
```

It is possible that the result tree will contain a character that cannot be represented in the encoding that the XSLT processor is using for output. In

this case, if the character occurs in a context where HTML recognizes character references, then the character should be output as a character entity reference or decimal numeric character reference; otherwise (for example, in a `script` or `style` element or in a comment), the XSLT processor should signal an error.

If the `doctype-public` or `doctype-system` attributes are specified, then the `html` output method should output a document type declaration immediately before the first element. The name following `<!DOCTYPE` should be `HTML` or `html`. If the `doctype-public` attribute is specified, then the output method should output `PUBLIC` followed by the specified public identifier; if the `doctype-system` attribute is also specified, it should also output the specified system identifier following the public identifier. If the `doctype-system` attribute is specified but the `doctype-public` attribute is not specified, then the output method should output `SYSTEM` followed by the specified system identifier.

The `media-type` attribute is applicable for the `html` output method. The default value is `text/html`.

16.3 Text Output Method

The `text` output method outputs the result tree by outputting the string-value of every text node in the result tree in document order without any escaping.

The `media-type` attribute is applicable for the `text` output method. The default value for the `media-type` attribute is `text/plain`.

The `encoding` attribute identifies the encoding that the `text` output method should use to convert sequences of characters to sequences of bytes. The default is system-dependent. If the result tree contains a character that cannot be represented in the encoding that the XSLT processor is using for output, the XSLT processor should signal an error.

16.4 Disabling Output Escaping

Normally, the `xml` output method escapes & and < (and possibly other characters) when outputting text nodes. This ensures that the output is well-formed XML. However, it is sometimes convenient to be able to produce output that is almost, but not quite well-formed XML; for example, the output may include ill-formed sections which are intended to be transformed into well-formed XML by a subsequent non-XML aware process. For this reason, XSLT provides a mechanism for disabling output escaping. An `xsl:value-of` or `xsl:text` element may have a `disable-output-escaping` attribute; the

allowed values are `yes` or `no`; the default is `no`; if the value is `yes`, then a text node generated by instantiating the `xsl:value-of` or `xsl:text` element should be output without any escaping. For example,

```
<xsl:text disable-output-escaping="yes">&lt;</xsl:
text>
```

should generate the single character <.

It is an error for output escaping to be disabled for a text node that is used for something other than a text node in the result tree. Thus, it is an error to disable output escaping for an `xsl:value-of` or `xsl:text` element that is used to generate the string-value of a comment, processing instruction or attribute node; it is also an error to convert a result tree fragment to a number or a string if the result tree fragment contains a text node for which escaping was disabled. In both cases, an XSLT processor may signal the error; if it does not signal the error, it must recover by ignoring the `disable-output-escaping` attribute.

The `disable-output-escaping` attribute may be used with the `html` output method as well as with the `xml` output method. The `text` output method ignores the `disable-output-escaping` attribute, since it does not perform any output escaping.

An XSLT processor will only be able to disable output escaping if it controls how the result tree is output. This may not always be the case. For example, the result tree may be used as the source tree for another XSLT transformation instead of being output. An XSLT processor is not required to support disabling output escaping. If an `xsl:value-of` or `xsl:text` specifies that output escaping should be disabled and the XSLT processor does not support this, the XSLT processor may signal an error; if it does not signal an error, it must recover by not disabling output escaping.

If output escaping is disabled for a character that is not representable in the encoding that the XSLT processor is using for output, then the XSLT processor may signal an error; if it does not signal an error, it must recover by not disabling output escaping.

Since disabling output escaping may not work with all XSLT processors and can result in XML that is not well-formed, it should be used only when there is no alternative.

17 Conformance

A conforming XSLT processor must be able to use a stylesheet to transform a source tree into a result tree as specified in this document. A conforming XSLT processor need not be able to output the result in XML or in any other form.

> **NOTE:**Vendors of XSLT processors are strongly encouraged to provide a way to verify that their processor is behaving conformingly by allowing the result tree to be output as XML or by providing access to the result tree through a standard API such as the DOM or SAX.

A conforming XSLT processor must signal any errors except for those that this document specifically allows an XSLT processor not to signal. A conforming XSLT processor may but need not recover from any errors that it signals.

A conforming XSLT processor may impose limits on the processing resources consumed by the processing of a stylesheet.

18 Notation

The specification of each XSLT-defined element type is preceded by a summary of its syntax in the form of a model for elements of that element type. The meaning of syntax summary notation is as follows:

- An attribute is required if and only if its name is in bold.
- The string that occurs in the place of an attribute value specifies the allowed values of the attribute. If this is surrounded by curly braces, then the attribute value is treated as an attribute value template, and the string occurring within curly braces specifies the allowed values of the result of instantiating the attribute value template. Alternative allowed values are separated by `|`. A quoted string indicates a value equal to that specific string. An unquoted, italicized name specifies a particular type of value.
- If the element is allowed not to be empty, then the element contains a comment specifying the allowed content. The allowed content is specified in a similar way to an element type declaration in XML; *template* means that any mixture of text nodes, literal result elements, extension elements, and XSLT elements from the `instruction` category is allowed; *top-level-elements* means that any mixture of XSLT elements from the `top-level-element` category is allowed.

- The element is prefaced by comments indicating if it belongs to the `instruction` category or `top-level-element` category or both. The category of an element just affects whether it is allowed in the content of elements that allow a *template* or *top-level-elements*.

A References

A.1 Normative References

XML

World Wide Web Consortium. *Extensible Markup Language (XML) 1.0.* W3C Recommendation. See http://www.w3.org/TR/1998/REC-xml-19980210

XML Names

World Wide Web Consortium. *Namespaces in XML.* W3C Recommendation. See http://www.w3.org/TR/REC-xml-names

XPath

World Wide Web Consortium. *XML Path Language.* W3C Recommendation. See http://www.w3.org/TR/xpath

A.2 Other References

CSS2

World Wide Web Consortium. *Cascading Style Sheets, level 2 (CSS2)*. W3C Recommendation. See http://www.w3.org/TR/1998/REC-CSS2-19980512

DSSSL

International Organization for Standardization, International Electrotechnical Commission. *ISO/IEC 10179:1996. Document Style Semantics and Specification Language (DSSSL)*. International Standard.

HTML

World Wide Web Consortium. *HTML 4.0 specification*. W3C Recommendation. See http://www.w3.org/TR/REC-html40

IANA

Internet Assigned Numbers Authority. *Character Sets*. See ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets.

RFC2278

N. Freed, J. Postel. *IANA Charset Registration Procedures*. IETF RFC 2278. See http://www.ietf.org/rfc/rfc2278.txt.

RFC2376

E. Whitehead, M. Murata. *XML Media Types*. IETF RFC 2376. See http://www.ietf.org/rfc/rfc2376.txt.

RFC2396

T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*. IETF RFC 2396. See http://www.ietf.org/rfc/rfc2396.txt.

UNICODE TR10

Unicode Consortium. *Unicode Technical Report #10. Unicode Collation Algorithm*. Unicode Technical Report. See http://www.unicode.org/unicode/reports/tr10/index.html.

XHTML

World Wide Web Consortium. *XHTML 1.0: The Extensible HyperText Markup Language.* W3C Proposed Recommendation. See http://www.w3.org/TR/xhtml1

XPointer

World Wide Web Consortium. *XML Pointer Language (XPointer).* W3C Working Draft. See http://www.w3.org/TR/xptr

XML Stylesheet

World Wide Web Consortium. *Associating stylesheets with XML documents.* W3C Recommendation. See http://www.w3.org/TR/xml-stylesheet

XSL

World Wide Web Consortium. *Extensible Stylesheet Language (XSL).* W3C Working Draft. See http://www.w3.org/TR/WD-xsl

B Element Syntax Summary

```
<!-- Category: instruction -->
<xsl:apply-imports />

<!-- Category: instruction -->
<xsl:apply-templates
  select = node-set-expression
  mode = qname>
  <!-- Content: (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>

<!-- Category: instruction -->
<xsl:attribute
  name = { qname }
  namespace = { uri-reference }>
  <!-- Content: template -->
</xsl:attribute>
```

```
<!-- Category: top-level-element -->
<xsl:attribute-set
  name = qname
  use-attribute-sets = qnames>
  <!-- Content: xsl:attribute* -->
</xsl:attribute-set>

<!-- Category: instruction -->
<xsl:call-template
  name = qname>
  <!-- Content: xsl:with-param* -->
</xsl:call-template>

<!-- Category: instruction -->
<xsl:choose>
  <!-- Content: (xsl:when+, xsl:otherwise?) -->
</xsl:choose>

<!-- Category: instruction -->
<xsl:comment>
  <!-- Content: template -->
</xsl:comment>

<!-- Category: instruction -->
<xsl:copy
  use-attribute-sets = qnames>
  <!-- Content: template -->
</xsl:copy>

<!-- Category: instruction -->
<xsl:copy-of
  select = expression />

<!-- Category: top-level-element -->
<xsl:decimal-format
  name = qname
  decimal-separator = char
  grouping-separator = char
  infinity = string
  minus-sign = char
  NaN = string
  percent = char
  per-mille = char
  zero-digit = char
```

```
  digit = char
  pattern-separator = char />

<!-- Category: instruction -->
<xsl:element
  name = { qname }
  namespace = { uri-reference }
  use-attribute-sets = qnames>
  <!-- Content: template -->
</xsl:element>

<!-- Category: instruction -->
<xsl:fallback>
  <!-- Content: template -->
</xsl:fallback>

<!-- Category: instruction -->
<xsl:for-each
  select = node-set-expression>
  <!-- Content: (xsl:sort*, template) -->
</xsl:for-each>

<!-- Category: instruction -->
<xsl:if
  test = boolean-expression>
  <!-- Content: template -->
</xsl:if>

<xsl:import
  href = uri-reference />

<!-- Category: top-level-element -->
<xsl:include
  href = uri-reference />

<!-- Category: top-level-element -->
<xsl:key
  name = qname
  match = pattern
  use = expression />

<!-- Category: instruction -->
<xsl:message
  terminate = "yes" | "no">
```

```
  <!-- Content: template -->
</xsl:message>

<!-- Category: top-level-element -->
<xsl:namespace-alias
  stylesheet-prefix = prefix | "#default"
  result-prefix = prefix | "#default" />

<!-- Category: instruction -->
<xsl:number
  level = "single" | "multiple" | "any"
  count = pattern
  from = pattern
  value = number-expression
  format = { string }
  lang = { nmtoken }
  letter-value = { "alphabetic" | "traditional" }
  grouping-separator = { char }
  grouping-size = { number } />

<xsl:otherwise>
  <!-- Content: template -->
</xsl:otherwise>

<!-- Category: top-level-element -->
<xsl:output
  method = "xml" | "html" | "text" | qname-but-not-ncname
  version = nmtoken
  encoding = string
  omit-xml-declaration = "yes" | "no"
  standalone = "yes" | "no"
  doctype-public = string
  doctype-system = string
  cdata-section-elements = qnames
  indent = "yes" | "no"
  media-type = string />

<!-- Category: top-level-element -->
<xsl:param
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:param>

<!-- Category: top-level-element -->
```

```
<xsl:preserve-space
  elements = tokens />

<!-- Category: instruction -->
<xsl:processing-instruction
  name = { ncname }>
  <!-- Content: template -->
</xsl:processing-instruction>

<xsl:sort
  select = string-expression
  lang = { nmtoken }
  data-type = { "text" | "number" | qname-but-not-ncname }
  order = { "ascending" | "descending" }
  case-order = { "upper-first" | "lower-first" } />

<!-- Category: top-level-element -->
<xsl:strip-space
  elements = tokens />

<xsl:stylesheet
  id = id
  extension-element-prefixes = tokens
  exclude-result-prefixes = tokens
  version = number>
  <!-- Content: (xsl:import*, top-level-elements) -->
</xsl:stylesheet>

<!-- Category: top-level-element -->
<xsl:template
  match = pattern
  name = qname
  priority = number
  mode = qname>
  <!-- Content: (xsl:param*, template) -->
</xsl:template>

<!-- Category: instruction -->
<xsl:text
  disable-output-escaping = "yes" | "no">
  <!-- Content: #PCDATA -->
</xsl:text>

<xsl:transform
```

```
  id = id
  extension-element-prefixes = tokens
  exclude-result-prefixes = tokens
  version = number>
  <!-- Content: (xsl:import*, top-level-elements) -->
</xsl:transform>

<!-- Category: instruction -->
<xsl:value-of
  select = string-expression
  disable-output-escaping = "yes" | "no" />

<!-- Category: top-level-element -->
<!-- Category: instruction -->
<xsl:variable
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:variable>

<xsl:when
  test = boolean-expression>
  <!-- Content: template -->
</xsl:when>

<xsl:with-param
  name = qname
  select = expression>
  <!-- Content: template -->
</xsl:with-param>
```

C DTD Fragment for XSLT Stylesheets (Non-Normative)

> **NOTE:**This DTD Fragment is not normative because XML 1.0 DTDs do not support XML Namespaces and thus cannot correctly describe the allowed structure of an XSLT stylesheet.

The following entity can be used to construct a DTD for XSLT stylesheets that create instances of a particular result DTD. Before referencing the entity, the stylesheet DTD must define a `result-elements` parameter entity listing the allowed result element types. For example:

```
<!ENTITY % result-elements "
```

```
    | fo:inline-sequence
    | fo:block
">
```

Such result elements should be declared to have `xsl:use-attribute-sets` and `xsl:extension-element-prefixes` attributes. The following entity declares the `result-element-atts` parameter for this purpose. The content that XSLT allows for result elements is the same as it allows for the XSLT elements that are declared in the following entity with a content model of `%template;`. The DTD may use a more restrictive content model than `%template;` to reflect the constraints of the result DTD.

The DTD may define the `non-xsl-top-level` parameter entity to allow additional top-level elements from namespaces other than the XSLT namespace.

The use of the `xsl:` prefix in this DTD does not imply that XSLT stylesheets are required to use this prefix. Any of the elements declared in this DTD may have attributes whose name starts with `xmlns:` or is equal to `xmlns` in addition to the attributes declared in this DTD.

```
<!ENTITY % char-instructions "
  | xsl:apply-templates
  | xsl:call-template
  | xsl:apply-imports
  | xsl:for-each
  | xsl:value-of
  | xsl:copy-of
  | xsl:number
  | xsl:choose
  | xsl:if
  | xsl:text
  | xsl:copy
  | xsl:variable
  | xsl:message
  | xsl:fallback
">

<!ENTITY % instructions "
  %char-instructions;
  | xsl:processing-instruction
  | xsl:comment
  | xsl:element
  | xsl:attribute
">
```

```
<!ENTITY % char-template "
 (#PCDATA
  %char-instructions;)*
">

<!ENTITY % template "
 (#PCDATA
  %instructions;
  %result-elements;)*
">

<!-- Used for the type of an attribute value that is
a URI reference.-->
<!ENTITY % URI "CDATA">

<!-- Used for the type of an attribute value that is
a pattern.-->
<!ENTITY % pattern "CDATA">

<!-- Used for the type of an attribute value that is
an
     attribute value template.-->
<!ENTITY % avt "CDATA">

<!-- Used for the type of an attribute value that is
a QName; the prefix
     gets expanded by the XSLT processor. -->
<!ENTITY % qname "NMTOKEN">

<!-- Like qname but a whitespace-separated list of
QNames. -->
<!ENTITY % qnames "NMTOKENS">

<!-- Used for the type of an attribute value that is
an expression.-->
<!ENTITY % expr "CDATA">

<!-- Used for the type of an attribute value that
consists
     of a single character.-->
<!ENTITY % char "CDATA">

<!-- Used for the type of an attribute value that is
a priority. -->
<!ENTITY % priority "NMTOKEN">
```

```
<!ENTITY % space-att "xml:space (default|preserve)
#IMPLIED">

<!-- This may be overridden to customize the set of
elements allowed
at the top-level. -->

<!ENTITY % non-xsl-top-level "">

<!ENTITY % top-level "
 (xsl:import*,
  (xsl:include
  | xsl:strip-space
  | xsl:preserve-space
  | xsl:output
  | xsl:key
  | xsl:decimal-format
  | xsl:attribute-set
  | xsl:variable
  | xsl:param
  | xsl:template
  | xsl:namespace-alias
  %non-xsl-top-level;)*)
">

<!ENTITY % top-level-atts '
  extension-element-prefixes CDATA #IMPLIED
  exclude-result-prefixes CDATA #IMPLIED
  id ID #IMPLIED
  version NMTOKEN #REQUIRED
  xmlns:xsl CDATA #FIXED "http://www.w3.org/1999/XSL/
Transform"
  %space-att;
'>

<!-- This entity is defined for use in the ATTLIST
declaration
for result elements. -->

<!ENTITY % result-element-atts '
  xsl:extension-element-prefixes CDATA #IMPLIED
  xsl:exclude-result-prefixes CDATA #IMPLIED
  xsl:use-attribute-sets %qnames; #IMPLIED
  xsl:version NMTOKEN #IMPLIED
'>
```

```
<!ELEMENT xsl:stylesheet %top-level;>
<!ATTLIST xsl:stylesheet %top-level-atts;>

<!ELEMENT xsl:transform %top-level;>
<!ATTLIST xsl:transform %top-level-atts;>

<!ELEMENT xsl:import EMPTY>
<!ATTLIST xsl:import href %URI; #REQUIRED>

<!ELEMENT xsl:include EMPTY>
<!ATTLIST xsl:include href %URI; #REQUIRED>

<!ELEMENT xsl:strip-space EMPTY>
<!ATTLIST xsl:strip-space elements CDATA #REQUIRED>

<!ELEMENT xsl:preserve-space EMPTY>
<!ATTLIST xsl:preserve-space elements CDATA
#REQUIRED>

<!ELEMENT xsl:output EMPTY>
<!ATTLIST xsl:output
  method %qname; #IMPLIED
  version NMTOKEN #IMPLIED
  encoding CDATA #IMPLIED
  omit-xml-declaration (yes|no) #IMPLIED
  standalone (yes|no) #IMPLIED
  doctype-public CDATA #IMPLIED
  doctype-system CDATA #IMPLIED
  cdata-section-elements %qnames; #IMPLIED
  indent (yes|no) #IMPLIED
  media-type CDATA #IMPLIED
>

<!ELEMENT xsl:key EMPTY>
<!ATTLIST xsl:key
  name %qname; #REQUIRED
  match %pattern; #REQUIRED
  use %expr; #REQUIRED
>

<!ELEMENT xsl:decimal-format EMPTY>
<!ATTLIST xsl:decimal-format
  name %qname; #IMPLIED
  decimal-separator %char; "."
  grouping-separator %char; ","
```

```
  infinity CDATA "Infinity"
  minus-sign %char; "-"
  NaN CDATA "NaN"
  percent %char; "%"
  per-mille %char; "&#x2030;"
  zero-digit %char; "0"
  digit %char; "#"
  pattern-separator %char; ";"
>

<!ELEMENT xsl:namespace-alias EMPTY>
<!ATTLIST xsl:namespace-alias
  stylesheet-prefix CDATA #REQUIRED
  result-prefix CDATA #REQUIRED
>

<!ELEMENT xsl:template
 (#PCDATA
  %instructions;
  %result-elements;
  | xsl:param)*
>

<!ATTLIST xsl:template
  match %pattern; #IMPLIED
  name %qname; #IMPLIED
  priority %priority; #IMPLIED
  mode %qname; #IMPLIED
  %space-att;
>

<!ELEMENT xsl:value-of EMPTY>
<!ATTLIST xsl:value-of
  select %expr; #REQUIRED
  disable-output-escaping (yes|no) "no"
>

<!ELEMENT xsl:copy-of EMPTY>
<!ATTLIST xsl:copy-of select %expr; #REQUIRED>

<!ELEMENT xsl:number EMPTY>
<!ATTLIST xsl:number
   level (single|multiple|any) "single"
   count %pattern; #IMPLIED
   from %pattern; #IMPLIED
   value %expr; #IMPLIED
```

```
   format %avt; '1'
   lang %avt; #IMPLIED
   letter-value %avt; #IMPLIED
   grouping-separator %avt; #IMPLIED
   grouping-size %avt; #IMPLIED
>

<!ELEMENT xsl:apply-templates (xsl:sort|xsl:with-
param)*>
<!ATTLIST xsl:apply-templates
  select %expr; "node()"
  mode %qname; #IMPLIED
>

<!ELEMENT xsl:apply-imports EMPTY>

<!-- xsl:sort cannot occur after any other elements
or
any non-whitespace character -->

<!ELEMENT xsl:for-each
 (#PCDATA
  %instructions;
  %result-elements;
  | xsl:sort)*
>

<!ATTLIST xsl:for-each
  select %expr; #REQUIRED
  %space-att;
>

<!ELEMENT xsl:sort EMPTY>
<!ATTLIST xsl:sort
  select %expr; "."
  lang %avt; #IMPLIED
  data-type %avt; "text"
  order %avt; "ascending"
  case-order %avt; #IMPLIED
>

<!ELEMENT xsl:if %template;>
<!ATTLIST xsl:if
  test %expr; #REQUIRED
  %space-att;
>
```

```
<!ELEMENT xsl:choose (xsl:when+, xsl:otherwise?)>
<!ATTLIST xsl:choose %space-att;>

<!ELEMENT xsl:when %template;>
<!ATTLIST xsl:when
  test %expr; #REQUIRED
  %space-att;
>

<!ELEMENT xsl:otherwise %template;>
<!ATTLIST xsl:otherwise %space-att;>

<!ELEMENT xsl:attribute-set (xsl:attribute)*>
<!ATTLIST xsl:attribute-set
  name %qname; #REQUIRED
  use-attribute-sets %qnames; #IMPLIED
>

<!ELEMENT xsl:call-template (xsl:with-param)*>
<!ATTLIST xsl:call-template
  name %qname; #REQUIRED
>

<!ELEMENT xsl:with-param %template;>
<!ATTLIST xsl:with-param
  name %qname; #REQUIRED
  select %expr; #IMPLIED
>

<!ELEMENT xsl:variable %template;>
<!ATTLIST xsl:variable
  name %qname; #REQUIRED
  select %expr; #IMPLIED
>

<!ELEMENT xsl:param %template;>
<!ATTLIST xsl:param
  name %qname; #REQUIRED
  select %expr; #IMPLIED
>

<!ELEMENT xsl:text (#PCDATA)>
<!ATTLIST xsl:text
  disable-output-escaping (yes|no) "no"
>
```

```
<!ELEMENT xsl:processing-instruction %char-template;>
<!ATTLIST xsl:processing-instruction
  name %avt; #REQUIRED
  %space-att;
>

<!ELEMENT xsl:element %template;>
<!ATTLIST xsl:element
  name %avt; #REQUIRED
  namespace %avt; #IMPLIED
  use-attribute-sets %qnames; #IMPLIED
  %space-att;
>

<!ELEMENT xsl:attribute %char-template;>
<!ATTLIST xsl:attribute
  name %avt; #REQUIRED
  namespace %avt; #IMPLIED
  %space-att;
>

<!ELEMENT xsl:comment %char-template;>
<!ATTLIST xsl:comment %space-att;>

<!ELEMENT xsl:copy %template;>
<!ATTLIST xsl:copy
  %space-att;
  use-attribute-sets %qnames; #IMPLIED
>

<!ELEMENT xsl:message %template;>
<!ATTLIST xsl:message
  %space-att;
  terminate (yes|no) "no"
>

<!ELEMENT xsl:fallback %template;>
<!ATTLIST xsl:fallback %space-att;>
```

D Examples (Non-Normative)

D.1 Document Example

This example is a stylesheet for transforming documents that conform to a

simple DTD into XHTML [XHTML]. The DTD is:

```
<!ELEMENT doc (title, chapter*)>
<!ELEMENT chapter (title, (para|note)*, section*)>
<!ELEMENT section (title, (para|note)*)>
<!ELEMENT title (#PCDATA|emph)*>
<!ELEMENT para (#PCDATA|emph)*>
<!ELEMENT note (#PCDATA|emph)*>
<!ELEMENT emph (#PCDATA|emph)*>
```

The stylesheet is:

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/
XSL/Transform"
                xmlns="http://www.w3.org/TR/xhtml1/
strict">

<xsl:strip-space elements="doc chapter section"/>
<xsl:output
   method="xml"
   indent="yes"
   encoding="iso-8859-1"
/>

<xsl:template match="doc">
 <html>
   <head>
     <title>
       <xsl:value-of select="title"/>
     </title>
   </head>
   <body>
     <xsl:apply-templates/>
   </body>
 </html>
</xsl:template>

<xsl:template match="doc/title">
  <h1>
    <xsl:apply-templates/>
  </h1>
</xsl:template>

<xsl:template match="chapter/title">
  <h2>
```

```
    <xsl:apply-templates/>
  </h2>
</xsl:template>

<xsl:template match="section/title">
  <h3>
    <xsl:apply-templates/>
  </h3>
</xsl:template>

<xsl:template match="para">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>

<xsl:template match="note">
  <p class="note">
    <b>NOTE: </b>
    <xsl:apply-templates/>
  </p>
</xsl:template>

<xsl:template match="emph">
  <em>
    <xsl:apply-templates/>
  </em>
</xsl:template>

</xsl:stylesheet>
```

With the following input document

```
<!DOCTYPE doc SYSTEM "doc.dtd">
<doc>
<title>Document Title</title>
<chapter>
<title>Chapter Title</title>
<section>
<title>Section Title</title>
<para>This is a test.</para>
<note>This is a note.</note>
</section>
<section>
<title>Another Section Title</title>
<para>This is <emph>another</emph> test.</para>
```

```
<note>This is another note.</note>
</section>
</chapter>
</doc>
```

it would produce the following result

```
<?xml version="1.0" encoding="iso-8859-1"?>
<html xmlns="http://www.w3.org/TR/xhtml1/strict">
<head>
<title>Document Title</title>
</head>
<body>
<h1>Document Title</h1>
<h2>Chapter Title</h2>
<h3>Section Title</h3>
<p>This is a test.</p>
<p class="note">
<b>NOTE: </b>This is a note.</p>
<h3>Another Section Title</h3>
<p>This is <em>another</em> test.</p>
<p class="note">
<b>NOTE: </b>This is another note.</p>
</body>
</html>
```

D.2 Data Example

This is an example of transforming some data represented in XML using three different XSLT stylesheets to produce three different representations of the data, HTML, SVG and VRML.

The input data is:

```
<sales>

        <division id="North">
                <revenue>10</revenue>
                <growth>9</growth>
                <bonus>7</bonus>
        </division>

        <division id="South">
                <revenue>4</revenue>
                <growth>3</growth>
```

```
                <bonus>4</bonus>
        </division>

        <division id="West">
                <revenue>6</revenue>
                <growth>-1.5</growth>
                <bonus>2</bonus>
        </division>

</sales>
```

The following stylesheet, which uses the simplified syntax described in [**2.3 Literal Result Element as Stylesheet**], transforms the data into HTML:

```
<html xsl:version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/
Transform"
      lang="en">
    <head>
        <title>Sales Results By Division</title>
    </head>
    <body>
        <table border="1">
            <tr>
                <th>Division</th>
                <th>Revenue</th>
                <th>Growth</th>
                <th>Bonus</th>
            </tr>
            <xsl:for-each select="sales/division">
                <!-- order the result by revenue -->
                <xsl:sort select="revenue"
                          data-type="number"
                          order="descending"/>
                <tr>
                    <td>
                        <em><xsl:value-of
select="@id"/></em>
                    </td>
                    <td>
                        <xsl:value-of
select="revenue"/>
                    </td>
                    <td>
                        <!-- highlight negative
growth in red -->
```

```
                        <xsl:if test="growth &lt; 0">
                            <xsl:attribute
    name="style">
                                <xsl:text>color:
    red</xsl:text>
                            </xsl:attribute>
                        </xsl:if>
                        <xsl:value-of
    select="growth"/>
                    </td>
                    <td>
                        <xsl:value-of select="bonus"/
    >
                    </td>
                </tr>
            </xsl:for-each>
        </table>
    </body>
</html>
```

The HTML output is:

```
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
<title>Sales Results By Division</title>
</head>
<body>
<table border="1">
<tr>
<th>Division</th><th>Revenue</th><th>Growth</
th><th>Bonus</th>
</tr>
<tr>
<td><em>North</em></td><td>10</td><td>9</td><td>7</
td>
</tr>
<tr>
<td><em>West</em></td><td>6</td><td style="color:
red">-1.5</td><td>2</td>
</tr>
<tr>
<td><em>South</em></td><td>4</td><td>3</td><td>4</td>
</tr>
</table>
```

```
</body>
</html>
```

The following stylesheet transforms the data into SVG:

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/
XSL/Transform"
                xmlns="http://www.w3.org/Graphics/
SVG/SVG-19990812.dtd">

<xsl:output method="xml" indent="yes" media-
type="image/svg"/>

<xsl:template match="/">

<svg width = "3in" height="3in">
    <g style = "stroke: #000000">
        <!-- draw the axes -->
        <line x1="0" x2="150" y1="150" y2="150"/>
        <line x1="0" x2="0" y1="0" y2="150"/>
        <text x="0" y="10">Revenue</text>
        <text x="150" y="165">Division</text>
        <xsl:for-each select="sales/division">
            <!-- define some useful variables -->

            <!-- the bar's x position -->
            <xsl:variable name="pos"
                          select="(position()*40)-
30"/>

            <!-- the bar's height -->
            <xsl:variable name="height"
                          select="revenue*10"/>

            <!-- the rectangle -->
            <rect x="{$pos}" y="{150-$height}"
                  width="20" height="{$height}"/>

            <!-- the text label -->
            <text x="{$pos}" y="165">
                <xsl:value-of select="@id"/>
            </text>

            <!-- the bar value -->
            <text x="{$pos}" y="{145-$height}">
```

```
                    <xsl:value-of select="revenue"/>
                </text>
            </xsl:for-each>
        </g>
    </svg>

    </xsl:template>
    </xsl:stylesheet>
```

The SVG output is:

```
    <svg width="3in" height="3in"
         xmlns="http://www.w3.org/Graphics/SVG/svg-
    19990412.dtd">
        <g style="stroke: #000000">
            <line x1="0" x2="150" y1="150" y2="150"/>
            <line x1="0" x2="0" y1="0" y2="150"/>
            <text x="0" y="10">Revenue</text>
            <text x="150" y="165">Division</text>
            <rect x="10" y="50" width="20" height="100"/>
            <text x="10" y="165">North</text>
            <text x="10" y="45">10</text>
            <rect x="50" y="110" width="20" height="40"/>
            <text x="50" y="165">South</text>
            <text x="50" y="105">4</text>
            <rect x="90" y="90" width="20" height="60"/>
            <text x="90" y="165">West</text>
            <text x="90" y="85">6</text>
        </g>
    </svg>
```

The following stylesheet transforms the data into VRML:

```
    <xsl:stylesheet version="1.0"
                    xmlns:xsl="http://www.w3.org/1999/
    XSL/Transform">

    <!-- generate text output as mime type model/vrml,
    using default charset -->
    <xsl:output method="text" encoding="UTF-8" media-
    type="model/vrml"/>

            <xsl:template match="/">#VRML V2.0 utf8

    # externproto definition of a single bar element
    EXTERNPROTO bar [
```

```
  field SFInt32 x
  field SFInt32 y
  field SFInt32 z
  field SFString name
  ]
  "http://www.vrml.org/WorkingGroups/dbwork/barProto.
wrl"

# inline containing the graph axes
Inline {
        url "http://www.vrml.org/WorkingGroups/
dbwork/barAxes.wrl"
        }

                <xsl:for-each select="sales/
division">
bar {
        x <xsl:value-of select="revenue"/>
        y <xsl:value-of select="growth"/>
        z <xsl:value-of select="bonus"/>
        name "<xsl:value-of select="@id"/>"
        }
                </xsl:for-each>

        </xsl:template>

</xsl:stylesheet>
```

The VRML output is:

```
#VRML V2.0 utf8

# externproto definition of a single bar element
EXTERNPROTO bar [
  field SFInt32 x
  field SFInt32 y
  field SFInt32 z
  field SFString name
  ]
  "http://www.vrml.org/WorkingGroups/dbwork/barProto.
wrl"

# inline containing the graph axes
Inline {
        url "http://www.vrml.org/WorkingGroups/
dbwork/barAxes.wrl"
```

```
        }


bar {
        x 10
        y 9
        z 7
        name "North"
        }

bar {
        x 4
        y 3
        z 4
        name "South"
        }

bar {
        x 6
        y -1.5
        z 2
        name "West"
        }
```

E Acknowledgements (Non-Normative)

The following have contributed to authoring this draft:

- Daniel Lipkin, Saba
- Jonathan Marsh, Microsoft
- Henry Thompson, University of Edinburgh
- Norman Walsh, Arbortext
- Steve Zilles, Adobe

This specification was developed and approved for publication by the W3C XSL Working Group (WG). WG approval of this specification does not necessarily imply that all WG members voted for its approval. The current members of the XSL WG are:

Sharon Adler, IBM (Co-Chair); Anders Berglund, IBM; Perin Blanchard, Novell; Scott Boag, Lotus; Larry Cable, Sun; Jeff Caruso, Bitstream; James Clark; Peter Danielsen, Bell Labs; Don Day, IBM; Stephen Deach, Adobe; Dwayne Dicks, SoftQuad; Andrew Greene, Bitstream; Paul Grosso, Arbortext; Eduardo Gutentag, Sun; Juliane Harbarth, Software AG; Mickey Kimchi, Enigma; Chris Lilley, W3C; Chris Maden, Exemplary Technologies; Jonathan

Marsh, Microsoft; Alex Milowski, Lexica; Steve Muench, Oracle; Scott Parnell, Xerox; Vincent Quint, W3C; Dan Rapp, Novell; Gregg Reynolds, Datalogics; Jonathan Robie, Software AG; Mark Scardina, Oracle; Henry Thompson, University of Edinburgh; Philip Wadler, Bell Labs; Norman Walsh, Arbortext; Sanjiva Weerawarana, IBM; Steve Zilles, Adobe (Co-Chair)

F Changes from Proposed Recommendation (Non-Normative)

The following are the changes since the Proposed Recommendation:

- The `xsl:version` attribute is required on a literal result element used as a stylesheet (see [**2.3 Literal Result Element as Stylesheet**]).
- The `data-type` attribute on `xsl:sort` can use a prefixed name to specify a data-type not defined by XSLT (see [**10 Sorting**]).

G Features under Consideration for Future Versions of XSLT (Non-Normative)

The following features are under consideration for versions of XSLT after XSLT 1.0:

- a conditional expression;
- support for XML Schema datatypes and archetypes;
- support for something like style rules in the original XSL submission;
- an attribute to control the default namespace for names occurring in XSLT attributes;
- support for entity references;
- support for DTDs in the data model;
- support for notations in the data model;
- a way to get back from an element to the elements that reference it (e. g. by IDREF attributes);
- an easier way to get an ID or key in another document;

- support for regular expressions for matching against any or all of text nodes, attribute values, attribute names, element type names;
- case-insensitive comparisons;
- normalization of strings before comparison, for example for compatibility characters;
- a function `string resolve(node-set)` function that treats the value of the argument as a relative URI and turns it into an absolute URI using the base URI of the node;
- multiple result documents;
- defaulting the `select` attribute on `xsl:value-of` to the current node;
- an attribute on `xsl:attribute` to control how the attribute value is normalized;
- additional attributes on `xsl:sort` to provide further control over sorting, such as relative order of scripts;
- a way to put the text of a resource identified by a URI into the result tree;
- allow unions in steps (e.g. `foo/(bar|baz)`);
- allow for result tree fragments all operations that are allowed for node-sets;
- a way to group together consecutive nodes having duplicate subelements or attributes;
- features to make handling of the HTML `style` attribute more convenient.


Lokalisierungspfad
<xsl:template match="">
Ersetzungsmuster
</xsl:template>

<?xml version="1.0" encoding="utf-8"?>

Person gefunden!

Person gefunden!

Person gefunden!

```
<?xml version="1.0" encoding="utf-8"?>
	<Mitarbeiter/>
	<Mitarbeiter/>
	<Mitarbeiter/>
```

```
<?xml version="1.0" encoding="utf-8"?>
    <Mitarbeiter>
        Hans
        Hinterhuber
    </Mitarbeiter>
    <Mitarbeiter>
        Franz
        Xaver
        Obermüller

          IT-Kompetenzverschiedene Betriebssysteme und
          professionelle
          Programmierung
          verschiedener Programmiersprachen
          Entwickler von 1988-1990
          Projektleiterfunktion
          von 1990-93 im X42-Projekt in Abteilung AB&amp;amp;C

    </Mitarbeiter>
    <Mitarbeiter>
        Fritz
        Meier
    </Mitarbeiter>
```

<?xml version="1.0" encoding="utf-8"?>
<Mitarbeiter><Vorname xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Hans</Vorname><Nachname xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Hinterhuber</Nachname></Mitarbeiter>
<Mitarbeiter><Vorname xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Franz</Vorname><Vorname xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Xaver</Vorname><Nachname xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Obermüller</Nachname><Qualifikationsprofil xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<u>IT-Kompetenz</u>verschiedene Betriebssysteme und
<Leistungsstufe>professionelle</Leistungsstufe>
<Qualifikation>Programmierung</Qualifikation>
verschiedener Programmiersprachen
<u><Qualifikation>Entwickler</Qualifikation></u> von 1988-1990
<u><Qualifikation>Projektleiterfunktion</Qualifikation></u>
von 1990-93 im X42-Projekt in Abteilung AB&C
</Qualifikationsprofil></Mitarbeiter>
<Mitarbeiter><Vorname xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Fritz</Vorname><Nachname xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Meier</Nachname></Mitarbeiter>

```
<?xml version="1.0" encoding="utf-8"?><?xml-stylesheet type="text/xsl" href="projektverwaltung.
xsl"?><ProjektVerwaltung xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
noNamespaceSchemaLocation="http://www.jeckle.de/vorlesung/xml/examples/projektverwaltung.
xsd">
    <Mitarbeiter>
        <Vorname>Hans</Vorname>
        <Nachname>Hinterhuber</Nachname>
    </Mitarbeiter>
    <Mitarbeiter>
        <Vorname>Franz</Vorname>
        <Vorname>Xaver</Vorname>
        <Nachname>Obermüller</Nachname>

    </Mitarbeiter>
    <Mitarbeiter>
        <Vorname>Fritz</Vorname>
        <Nachname>Meier</Nachname>
    </Mitarbeiter>
    <Projekt id="Prj01" Projektleiter="Pers01" Mitarbeiter="Pers01"/>
    <Projekt id="Prj02" Projektleiter="Pers02" Mitarbeiter="Pers03"/>
</ProjektVerwaltung>
```

xml.apache.org

PDF

Welcome to the Apache XML Project

The goals of the *Apache XML Project* (part of The Apache Software Foundation) are:

- Get Involved
- CVS Repositories
- Mailing Lists
- Reference
- Legal Stuff
- Contact Info

- to provide commercial-quality standards-based XML solutions that are developed in an open and cooperative fashion,
- to provide feedback to standards bodies (such as IETF and W3C) from an implementation perspective, and
- to be a focus for XML-related activities within Apache projects

The *Apache XML Project* currently consists of the following sub- projects, each focused on a different aspect of XML:

- *Xerces* - XML parsers in Java, C++ (with Perl and COM bindings)
- *Xalan* - XSLT stylesheet processors, in Java and C++
- *AxKit* - XML-based web publishing, in mod_perl
- *FOP* - XSL formatting objects, in Java
- *Forrest* - XML/XSLT-based framework for project documentation and website development, based on Cocoon
- *Xang* - Rapid development of dynamic server pages, in JavaScript
- *SOAP* - Simple Object Access Protocol
- *Batik* - A Java based toolkit for Scalable Vector Graphics (SVG)
- *Crimson* - A Java XML parser derived from the Sun Project X Parser.
- *XML Security* - Java and C++ implementations of the XML signature and encryption standards
- *Xindice* - A native XML database.
- *XML Commons* - focussed on common code and guidelines for xml projects

Xerces: XML parsers in Java and C++ (plus Perl and COM)

Xerces (named after the Xerces Blue butterfly) provides world-class XML parsing and generation. Fully-validating parsers are available for both Java and C++, implementing the W3C XML and DOM (Level 1 and 2) standards, as well as the de facto SAX (version 2) standard. The parsers are highly modular and configurable. Initial

support for XML Schema (draft W3C standard) is also provided.

A Perl wrapper is provided for the C++ version of Xerces, which allows access to a fully validating DOM XML parser from Perl. It also provides for full access to Unicode strings, since Unicode is a key part of the XML standard.

A COM wrapper (also for Xerces-C) provides compatibility with the Microsoft MSXML parser.

Xalan: XSL stylesheet processors in Java & C++

Xalan (named after a rare musical instrument) provides high-performance XSLT stylesheet processing. Xalan fully implements the W3C XSLT and XPath recommendations. The stylesheet processor is feature-rich and robust. The XPath Processor is useable as a stand-alone unit. Xalan uses the Bean Scripting Framework (BSF) to implement Java and script extensions, features EXSLT extensions, nodeset extension, multiple document output extensions and SQL extension.

Xalan is currently available in Java, and C++.

AxKit: XML-based web publishing in mod_perl

AxKit brings the power of XML web publishing and dynamic XML based applications to the Apache web server using the mod_perl framework and the Apache C API to combine the power of XML with the performance of a native Apache solution.

FOP: XSL Formatting Object processor in Java

FOP is the world's first print formatter driven by XSL formatting objects. It is a Java 1.2 application that reads a formatting object

tree and then turns it into a PDF document. The formatting object tree, can be in the form of an XML document (output by an XSLT engine like Xalan) or can be passed in memory as a DOM Document or (in the case of Xalan) SAX events.

Forrest: Standards-based documentation framework

Forrest is an XML/XSLT-based, skinnable project documentation and website development system. Content can be written in presentation-neutral XML, and then converted to HTML/PDF by Forrest. As Forrest is based on Cocoon, sites can scale seamlessly from simple static affairs to full-blown dynamic webapps. Forrest is the system used to generate this site (xml. apache.org).

Xang: Rapid development of dynamic server pages in JavaScript

Apache Xang lets you quickly build data-driven, cross-platform Web applications that integrate disparate data sources. The Xang architecture cleanly separates data, logic and presentation. It is based on open industry standards such as HTTP, XML, XSL, DOM and ECMAScript (JavaScript).

SOAP: Simple Object Access Protocol

The Apache Soap project is an implementation of the draft W3C protocol by the same name. It is based on, and supersedes, the IBM SOAP4J implementation.

From the draft W3C specification: SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote

procedure calls and responses.

Crimson: A Java XML parser derived from the Sun Project X Parser

The source code for Crimson is available under the xml-crimson CVS module. Please visit the Crimson pages for more information.

XML-Security: providing security functionality for XML data

XML-Security is a collection of Java and C++ libraries that provide security functionality for XML data. Both libraries provide an implementation of the W3C XML Digital Signature standards, and implementations of the W3C XML Encryption standard are currently being developed.

Xindice: A native XML database

Apache Xindice is a database designed from the ground up to store XML data or what is more commonly referred to as a native XML database.

XML Commons: focussed on common code and guidelines for xml projects

The first focus will be to organize and have common packaging for the various externally-defined standards code relating to XML - things like the DOM, SAX, and JAXP interfaces.

As the xml-commons community forms, we also hope to serve as a holding area for other common xml-related utilities and code, and to help promulgate common packaging, testing, documentation, and other guidelines across all xml.apache.org subprojects.

Anyone Can Participate

The *xml.apache.org* Project is composed of developers from all around the world, both

individuals and engineers from major corporations. All interested developers are welcome to join and participate. Learn more about how to get involved.

Projektverwaltung

Projekte

Projektnummer	Projektleiter
Prj01	Hinterhuber
Prj02	Obermüller

. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .

Mitarbeiter

Name	Projekt
Hans Hinterhuber	Prj01
Franz Xaver Obermüller	Prj02

Fritz Meier	Prj02

XSLTransformations (XSLT)

XSLT ist eine XML-Sprache, welche die Übersetzung von beliebigen XML-Dokumenten in allgemeine Unicode-Streams erlaubt. Häufigste Anwendung dürfte jedoch die Erzegung von anderen XML-Formaten aus bestehenden sein. Vom Sprachgesichtspunkt aus handelt es sich bei XSLT um eine funktionale Programmiersprache.

News

- Aus der XSLT-Arbeitsgruppe beim W3C: Es gibt Pläne v1.1 nicht weiterzuverfolgen, und stattdessen XSLT v2.0 vorzuziehen.
  v2.0 soll dabei einige -- nicht alle! -- der für v1.1 geplanten Erweiterungen beinhalten.
  Hierdurch soll insbesondere eine engere Synchronisation mit der XPath-Arbeitsgruppe, die ebenfalls derzeit an der Nachfolgeversion arbeitet, erreicht werden.
- Erste XSLT Konferenz
- Mozilla unterstützt ab M18 XSLT

Spezifikation und sonstige W3C-Dokumente zu XSLT

- Neuester XSLT v2.0 Working Draft vom 2002-11-15
- verabschiedete XSLT Specification v1.0
- XSLT v1.0 Specification Errata
- XSLT v1.1 Requirements
- nicht mehr weiterverfolgter XSLT v1.1 WD 2000-12-12

Links

- (W3C) XSL Transformation Requirements v1.1
- Allgemeine Turingmaschine in XSLT
- Buch zu XSLT
- Emacs Makros für XSLT
- Henning Behmes XSLT-Tutorial
- Improve your XSLT coding five ways --Tips to make you a better XSLT programmer
- John Lam: XSL Transformations: XSLT Alleviates XML Schema Incompatibility
- Mailing Liste
- Performance-Vergleich verschiedener XSLT-Prozessoren
- Steve Muench: Building Oracle XML Applications
- Trainingsmaterial
- Tutorial: *XMLT by example*
- XML-Schema for XSLT
- XSLT @ perfectxml.com
- XSLT Conformance Tests @ NIST
- XSLT-Portal @ bayes.co.uk
- XSLTracer, ein Werkzeug zur Verfolgung der Ausführung von Transformationen.
- XSLT Surgery -- Q&A @ xml.com
- XSLT Tricks
- XSLT Tutorial: Transforming XML
- XSLT Tutorial @ VBXML
- XSLT-Tutorials @ Tutorialfind.com

Mailinglisten

- XSL(T)-Mailingliste @ Mulberrytech.com
- XSLT-Mailingliste des W3C

XSLT Transformationen und Tricks

- Dave Pawson's XSLT FAQ
- EXST.org
- Gallery of Stupid XSL and XSLT Tricks
- Homepage von Jenni Tennison
- Homepage von Oliver Becker
- XSLT @ SourceForge

XSLT-Prozessoren

XSLT-Prozessoren-Benchmark
XSLTMark Compliance Test für XSLT-Prozessoren
XSLT Processor Benchmark

- 4XSLT Open-Source-Implementierung in Python
- Altova XSLT Engine
- Apache's *Xalan*
- EZ/X Java-Implementierung
- Infoteria's iXSLT
- James Clark's *XT*

... da durch den Autor nicht mehr weiterentwickelt mittlerweile als OpenSource unter 4xt.org verfügbar.

- jd.xslt
- LotusXSL
- Microsoft's MSXML
  - Joshua Allen's unofficial MS XSLT FAQ
- Napa
- Oracle J
- Sabletron
- Sablotron (Linux, Windows, Unix)
- Saxon, unterstützt ab Version 7 bereits den aktuellen Working Draft zu XSLT v2.0
  - Artikel über Saxon-Implementierung
- SUNs XSLT Compiler
- TransforMiix
- TransforMix Open-Source XSLT-Prozessor des Mozialla-Projekts
- UnicornXSL
- UWOBO -- Ein Web-basierter Stylsheetprozessor
- XESALT
- XSLJIT -- Just in Time Compiler für XSL
- XSLT Plugin für Netscape v4
- XSLT Prozessor für Mac
- Überblick

Sonstige Werkzeuge

- 4XDebug -- XSLT Debugger
- catchXSL!
  Profiler für XSLT
  XSLT-Editor und Debugger
- eXcelon Stylus
- SUN's *Translet* Compiler
- XFinity Designer graphisches Tool zur XSLT-Entwicklung
- XML Spy
  XML IDE die installierte XSLT-Engine mitbenutzen kann
- Xserator -- XSL(T) IDE incl. Debugger
- XSLAtHome
- XSLDebugger
- XSL Editor
- XSLTScript
  Kompakte Notation zur Formulierung von Stylesheets, die in XSLT übersetzt wird
- Yanam
  XSLT-Editor, der Transformationen während der Eingabe testweise ausführt

Literatur

- Bob DuCharme: XSLT Quickly, Manning, ISBN 1-930110-11-1
  Guter praxisorientierter Einstieg
- Bücherliste @ perfectxml.com
- Building Oracle XML Applications
  Das Kapitel *Transforming XML Using XSLT* ist vollständig online verfügbar
- Jeni Tennison: Beginning XSLT, Wrox, ASIN 1-861005-94-6
  Ebenfalls guter Einstieg, vermittelt kompakt und hervorragend geschrieben

die wichtigsten Basiskonzepte

- Jeni Tennison: XSLT and XPath On The Edge, Unlimited Edition, John Wiley & Sons, ISBN 0-764547-76-3
  Vertiefendes Werk. Deckt auch Besonderheiten einzelner XSLT-Engines ab.
- Khun Yee Fung: XSLT: Working with XML and HTML
- Michael Fitzgerald: Learning XSLT, O'Reilly, 2003, ISBN 0-596-00327-7
  Umfassendes einführendes Werk. Deckt bereits XSLT v2.0 und XPath v2.0 ab.
- Michael Kay: XSLT Programmer's Reference, Wrox Press, ISBN: 1-861003-12-9
  Beste Referenz zu XSLT!
- Practical Transformation Using XSLT and XPath
- XSLT-Kapitel meiner XML-Vorlesung

Konversion proprietärer Formate nach XML

... kann **nicht** mit XSLT erreicht werden. Jedoch sind inzwischen einige Werkzeuge verfügbar die sich dieser Aufgabenstellung angenommen haben.

- GoXML Transform
- VorteXML
- XML Junction

▶ TOPICS
Business
Databases
Graphics
Metadata
Mobile
Programming
Schemas
Style
Web
Web Services

▶ ESSENTIALS
Annotated XML
What is XML?
What is XSLT?
What is XSL-FO?
What is XLink?
What is XML Schema?
What is XQuery?
What is RDF?
What is RSS?
What are Topic Maps?
What are Web Services?
What are XForms?
XSLT Recipe of the Day

Manage Your Account
Forgot Your Password?

▶ FIND
Search
Article Archive

Add the latest news from XML.com to your web site
Click here!

▶ COLUMNS
<taglines/>
Dive into XML

What is XSLT?

by G. Ken Holman

August 16, 2000

Print
Email article link

Introduction

Now that we are successfully using XML to mark up our information according to our own vocabularies, we are taking control and responsibility for our information, instead of abdicating such control to product vendors. These vendors would rather lock our information into their proprietary schemes to keep us beholden to their solutions and technology.

But the flexibility inherent in the power given to each of us to develop our own vocabularies, and for industry associations, e-commerce consortia, and the W3C to develop their own vocabularies, presents the need to be able to transform information marked up in XML from one vocabulary to another.

Two W3C Recommendations, XSLT (the Extensible Stylesheet Language Transformations) and XPath (the XML

Hacking the Library
Jon Udell
Perl and XML
Practical XQuery
Python and XML
Rich Salz
Sacré SVG
Standards Lowdown
Transforming XML
XML Q&A
XML-Deviant

GUIDES
XML Resources
Buyer's Guide
Events Calendar
Standards List
Submissions List

TOOLBOX
Syntax Checker

Path Language), meet that need. They provide a powerful implementation of a tree-oriented transformation language for transmuting instances of XML using one vocabulary into either simple text, the legacy HTML vocabulary, or XML instances using any other vocabulary imaginable. We use the XSLT language, which itself uses XPath, to specify how an implementation of an XSLT processor is to create our desired output from our given marked-up input.

XSLT enables and empowers interoperability. This XML.com introduction strives to overview essential aspects of understanding the context in which these languages help us meet our transformation requirements, and to introduce substantive concepts and terminology to bolster the information available in the W3C Recommendation documents themselves.

Since April 1999 Crane Softwrights Ltd. has published commercial training material titled Practical Transformation Using XSLT and XPath, covering the entire scope of the W3C XSLT and XPath through working drafts and the final 1.0 recommendations. This material is delivered by Crane in instructor-led sessions and is licensed to other training organizations around the world needing to teach these exciting technologies.

Crane has rewritten the first two chapters of this material into prose. These prose-oriented chapters are published on XML.com correspondingly as two main sections. The material assumes no prior knowledge of XSLT and XPath and guides the reader through background, context, structure, concepts and introductory terminology.

Table of Contents

About SAXON

From version 6.4.2 onwards, Saxon releases can be found at the new home page at SourceForge:

This site is no longer the place to find the latest version, though it still holds older versions.

The following versions of Saxon are available for download from this site:

Status	Full Saxon	Instant Saxon
Elderly	5.5.1	5.5.1
Stable	6.2.2	6.2.2
Stable	6.3	6.3 (withdrawn)
Not Recommended	6.4	6.4.1

Versions 6.4 and 6.4.1 have been superseded by 6.4.2, which will be found at sourceforge.

Saxon 5.5.1

No longer the recommended version, but retained for users running applications that use old interfaces. It includes documentation, source code, Java executables, and sample code.

- Information
- Download (2.3 Mbytes)

For a list of known bugs in Saxon 5.5.1,

consult the Saxon 6.3 change log, which gives details of all bugs fixed in subsequent releases. (It's not necessarily true that all these bugs were present in Saxon 5.5.1, but it's a reasonable working assumption.)

Instant Saxon 5.5.1

Instant Saxon contains identical functionality to the full product, but packaged as a Windows executable for ease of installation and running. This package includes only basic documentation, and no source code or sample applications. Note that Saxon performs considerably better using Sun JDK 1.3 than using the Microsoft Java VM, so for production use, the full product should be downloaded.

- Information
- Download (300 Kbytes)

For a list of known bugs in Instant Saxon 5.5.1, consult the Saxon 6.3 change log, which gives details of all bugs fixed in subsequent releases. (It's not necessarily true that all these bugs were present in Instant Saxon 5.5.1, but it's a reasonable working assumption.)

Saxon 6.2.2

The full Saxon product. This version has proved very reliable, and is currently the recommended version unless you need the new features or bug-fixes in Saxon 6.3.

The download includes documentation, source code, Java executables, and sample code.

- Information
- Download (2836 Kbytes)

For a list of known bugs in Saxon 6.2.2,

consult the Saxon 6.3 change log, which gives details of all bugs fixed in Saxon 6.3.

Instant Saxon 6.2.2

Instant Saxon contains identical functionality to the full product, but packaged as a Windows executable for ease of installation and running. This package includes only basic documentation, and no source code or sample applications. Note that Saxon performs considerably better using Sun JDK 1.3 than using the Microsoft Java VM, so for production use, the full product should be downloaded.

- Information
- Download (356 Kbytes)

For a list of known bugs in Instant Saxon 6.2.2, consult the Saxon 6.3 change log, which gives details of all bugs fixed in Saxon 6.3.

Saxon 6.3

The latest version of the full Saxon product. For details of new features and bug fixes in this release, consult the Saxon 6.3 change log

The download includes documentation, source code, Java executables, and sample code.

- Information
- Download (2893 Kbytes)

Saxon 6.3 requires JDK 1.2 or later. It will therefore not run with the Microsoft Java VM. This change was not intentional, and I plan to support Microsoft Java again in a future version (though not necessarily for ever...)

Instant Saxon 6.3

Instant Saxon 6.3 is withdrawn: the product was incorrectly built, and does not work under the current Microsoft Java VM, which it requires.

Saxon 6.4

The latest version of the full Saxon product. Includes internal changes to provide a 20% performance boost, and for the first time supports JDOM.

The download includes documentation, source code, Java executables, and sample code.

- Information
- Details of changes
- Download (2895 Kbytes)

Instant Saxon 6.4.1

Instant Saxon contains identical functionality to the full product, but packaged as a Windows executable for ease of installation and running. This package includes only basic documentation, and no source code or sample applications. Note that Saxon performs considerably better using Sun JDK 1.3 than using the Microsoft Java VM, so for production use, the full product should be downloaded.

Instant Saxon 6.4 failed to install on machines without the default JAXP software installed. It is replaced by 6.4.1.

- Information
- Download (386 Kbytes)

Michael H. Kay
3 July 2001

Personal home page: http://users.iclway.co.uk/mhkay

Home

Dialog

Fragen, Anregungen, Wünsche? Die Feedback-Seite!

Hyperlinks

- ○ Ungültiger Hyperlink ...
- ○ Überflüssiger Hyperlink ...
- ○ fehlender hyperlink ...

... von Seite:

http://www.jeckle.de/

... nach Seite:

http://

Bemerkungen:

Seiteninhalt

- ○ Nicht mehr aktueller Inhalt ...
- ○ Inkorrekter Inhalt ...
- ○ Irreführender Inhalt ...

... auf Seite:

http://www.jeckle.de/

Bemerkung:

Anregungen, Wünsche, Fragen

Bitte tragen Sie weitere Anregungen, Wünsche und Fragen hier ein:

Und nun...

Ihr Name:

Ihre Email-Adresse:

ab die Post! Eingaben löschen

Service provided by Mario Jeckle

Generated: 2004-05-24T13:35:55+01:00

Feedback SiteMap

This page's original location: http://www.jeckle.de/feedback.html

RDF description for this page

Unified Modeling Language (UML)

- UML Resource Center
- UML FAQs @ jeckle.de
- UML Links
  - Generelles
  - Tutorials
  - Frequently Asked Questions (FAQ) und Mailinglisten
  - UML @ OMG
  - Austauschformate
  - Werkzeugbezogenes ...
  - Objektorientierung allgemein
  - UML Autoren
  - UML auf gut Deutsch(Vorschlag einer Gruppe deutschsprachiger Autoren zur einheitlichen deutschen Übersetzung der UML-Begriffe)
- UML Bücher
- UML Official Specification Documents / Offizielle UML Spezifikation
  - General Information
  - UML Profiles und sonstige UML-bezogene Aktivitäten
  - UML v2.0
  - UML v1.5
  - UML v1.4
  - UML v1.3
  - UML v1.2
  - UML v1.1
  - Unterschiede zwichen den UML-Versionen
- UML Online Publications / Netz Resourcen zur UML
- UML Tools
- UML2SVG-Transformationsdienst

eXtensible Markup Language (XML)

- XML Resource Center
  - Was ist die XML?
  - Links
  - Parser
  - XML-Editoren
  - XML (Einsteiger-)Informationen & Tutorials
  - Mailinglisten und User Groups
  - Konferenzen
  - FAQs und Antworten darauf
  - Vorlesung XML
- Entwurf von XML-Sprachen
- XML und Sicherheit (Digitale Signatur, Verschlüsselung, ...)
- Eine Übersicht existierender XML-Sprachen
  - Channel Description Format (CDF)
  - Chemical Markup Language (CML)
  - Customer Profile Exchange (CPEX)
  - Mathematical Markup Language (MathML)
  - Querying XML (XQuery)
  - RDF Site Summary (RSS)
  - Resource Description Framework (RDF)
  - Simple Markup Language (SML) und verwandte Ansätze (TMML, Minimal XML, YAML, ...)
  - Voice ML
  - Wireless Markup Language (WML)
  - XML Hypertext Markup Language (XHTML)
  - XML Linking Language (XLink)
  - XML Path Language (XPath)
  - XML Pointer Language (XPointer)
  - XML Protocols (XML-RPC, SOAP, W3C's XML-Protocols, ...)
  - XML-Schema (XSD, DCD, DT4DTD, Schematron, RELAX, ...)
    - Projekt zur Übersetzung der XML-Schema-Spezifikation ins Deutsche
      - XSD-DE Chat vom 2001-08-16
      - XSD-DE Chat vom 2001-09-11
      - XSD-DE Chat vom 2001-10-08
  - XML Stylesheet Language (XSL)
  - XSL Transformations (XSLT)
    - Konversion nativer Formate nach XML
- Vorlesung *eXtensible Markup Language*

▲

XML Metadata Interchange (XMI)

- XMI Resource Center
- Beispiele
  - Einfaches UML-Diagramm in XMI v1.0, v1.1 und als Rational Rose Export
  - Umfangreicheres UML-Diagramm in XMI v1.0
  - Generierung einer eigenen XMI-Sprache
  - Nutzung der MOF-DTD: UML v1.4 Metamodell in XMI v1.1
  - Generierung eines XSD-Schemas aus einem UML-Klassendiagramm unter Nutzung von XMI[UML]
- XMI News
- Werkzeuge mit XMI-Support
- Wozu ein Metadaten-Austauschformat?
- Ziele von XMI
- Hintergründe zur Entwicklung von XMI
- Offizielle Spezifikationsdokumente
- XMI Production for XML Schema
- Präsentationen, Vorträge und Veröffentlichungen
- Links
- Entwurf von XML-Sprachen mit XMI
- Modellaustausch mit dem XML Metadata Interchange Format
- UML2SVG-Transformationsdienst

▲

Vorträge und Publikationen

Vorträge und Publikationen

Abstracts und Kurzfassungen

▲

Vorlesungen

- Einführung in das .NET-Framework mit C# New
- Systemarchitekturen
  - Vorlesungsinhalt

- Mailingliste zur Vorlesung
- Literatur
- Aktuelles XML
- Standardisierung
  - Abstract
  - Ziel der Veranstaltung
  - Durchgeführtes Projekt
  - Links
  - Mailingliste zur Vorlesung
  - Aktuelles XML
- DB-Anwendungen
  - Vorlesungsinhalt
  - Mailingliste zur Vorlesung
  - Literatur
  - Links
  - Aktuelles XML
  - Alte Klausuren
- e-Business Engineering
  - Vorlesungsinhalt
  - Mailingliste zur Vorlesung
  - FAQs zur Vorlesung
  - Aktuelles XML
  - Alte Klausuren
- Integration Engineering
  - Vorlesungsinhalt
  - Mailingliste zur Vorlesung
  - Aktuelles XML
- Datenbanken
  - Vorlesungsinhalt
  - Mailingliste zur Vorlesung
  - Übungsaufgaben
  - FAQs zur Vorlesung
  - Aktuelles XML
  - Alte Klausuren
- eXtensible Markup Language
  - Teleteaching
  - Vorlesungsinhalt
  - Gliederung
  - Termine
  - FAQs zur Vorlesung
  - Anmeldung zum Praktikum im WS2002/03
  - Alte Klausuren
  - Übungsaufgaben:
  - 1. Übungsaufgabe
  - 2. Übungsaufgabe
    - Hinweise, Anmerkungen und FAQs zu dieser Übung
  - 3. Übungsaufgabe
    - Hinweise, Anmerkungen und FAQs zu dieser Übung
  - 4. Übungsaufgabe
    - Hinweise, Anmerkungen und FAQs zu dieser Übung
  - 5. Übungsaufgabe
    - Hinweise, Anmerkungen und FAQs zu dieser Übung
- Alte Klausuren
- Script
- Hinweise zum Script
- Empfohlene Literatur
- Archiv: Praktikumsaufgaben und -Lösungen aus dem WS2001/02

- Java Threads
  - Inhaltsübersicht
  - Script
  - Klausur vom Sommersemester 2002
  - Literatur
  - Internetressourcen zum Thema
- Java
  - Script
  - Gliederung
  - Abstract
- Systemnahe Software
- Software Engineering I
  - Hands on Java
  - Objektorientierung
  - Applikationserstellung
  - Anwendungsaspekte
- Chat zur Vorlesung
- Vorlesungsevaluierung

▲

Studien- und Abschlußarbeiten

- Mögliche noch zu vergebende Themen
- Gegenwärtig laufende Arbeiten
- Abgeschlossene Arbeiten
- HOWTO: Hilfestellung bei der Anfertigung von Arbeiten

▲

Web Services

- Web Services -- eine technisch orientierte Einführung
- Sicherheitsaspekte XML-basierter Web-Services
- Web Service Workshop *WS-RSD'02*
- Web Services @ Berliner XML-Tage

▲

GOOAL.net

Deutsche Mailingliste rund um die Objekttechnologie

direkt zu GOOAL.net ...

▲

Internet Search Engines / Suchmaschinen

- Deutsche Suchmaschinen
- Meta-Suchmaschinen
- Internationale Suchmaschinen
- E-Mail Finder
- Dictionaries und Übersetzungshilfen
- File Finder
- Books & Journals
- Page Registrierung

▲

Mersennesche Primzahlen

- Mersennesche Primzahlen
- SETI @ Home
- Rätsel der Proteinfaltung

▲

Web Services Workshop WS-RSD'02

- Workshop Homepage
- Call for Papers
- Workshop Program, Schedule, and Presentation Slides Workshop Program, Schedule, and Presentation Slides

▲

Free Stuff --- Software & Downloads

- XInclude Task for Ant
- Java version of Linpack benchmark
- Timer Task for Ant
- SOAPing New
- Java Assembler Playground New

▲

RSS-Newsfeeds

- jeckle.de Newsfeed (HTML-Version)
- Linux Kernel News
- Call for Papers Corner (HTML-Version)
- DB-Anwendungen Newsfeed (HTML-Version)
- e-Business Engineering Newsfeed (HTML-Version)
- Datenbanken Newsfeed (HTML-Version)
- Mehr Informationen zu RSS

▲

Feedback

direkt zu Seite ...

Rotkreuz Mitgliederverwaltung

direkt zu Seite ...

Dialog

direkt zu Seite ...

Über diese Seiten ...

direkt zu Seite ...

Suchen ...

direkt zu Seite ...

Mario Jeckle ...

direkt zu Seite ...

PGP public key

Service provided by Mario Jeckle

Generated: 2004-05-24T13:36:06+01:00

Feedback SiteMap

This page's original location: http://www.jeckle.de/sitemap.html

RDF description for this page